C - 4a - Tablice (Wprowadzenie)

4a.1 Tablice jednowymiarowe - sposób deklaracji i inicjowania

Prawie wszystkie algorytmy komputerowe wymagają przechowywania danych w jakiejś uporządkowanej formie. Taką formą są, między innymi, tablice (ang. **array**s). Tablice mogą być 1-, 2- i więcej-wymiarowe. Zaczniemy od tablic jednowymiarowych (coś w rodzaju wektora).

Jeżeli chcemy zadeklarować 6-elementową tablicę przechowującą liczby całkowite, to napiszemy:

```
int mojaPierwszaTablica[6];
```

Tu są nawiasy **kwadratowe**, a nie jakieś inne !!!

Numeracja tablic zaczyna się od zera, a nie od jedynki, ale 90% studentów o tym zapomina :(

Jeżeli chcielibyśmy przypisać do tablicy wartości np. 1, 4, 11, 22, 38 oraz 91 , to napisalibyśmy:

```
mojaPierwszaTablica[0] = 1;   // (*) Ciekawostka: tablice w Pascalu zaczynają się od 1
mojaPierwszaTablica[1] = 4;
mojaPierwszaTablica[2] = 11;
mojaPierwszaTablica[3] = 22;
mojaPierwszaTablica[4] = 38;
mojaPierwszaTablica[5] = 91;
```

Średnik, średnik, średnik !!!

lub krócej, od razu przy deklaracji tablicy:

```
int mojaPierwszaTablica[6] = { 1 , 4 , 11 , 22 , 38 , 91 };
```

Jeżeli kolejnym liczbom w naszej tablicy da się przyporządkować jakiś wzór, to można taką tablicę zainicjować w pętli:

```
for(int i=0 ; i<6 ; i++){
        mojaPierwszaTablica[i] = i*i+2*i-19;
}
```

Od zera do ... „prawie sześciu" . Zapis:
`for(int i=0 ; i<=5 ; i++){ ...`
byłby nieprofesjonalny. Tak po prostu nikt nie pisze!!!

Tu jest **i-ty** element tablicy.

Jak można dobrać się do konkretnego elementu?

```
printf("Czwarty element mojej tablicy: %d",mojaPierwszaTablica[3]);
```

Oczywiście można zrobić coś takiego:

```
mojaPierwszaTablica[2] = mojaPierwszaTablica[4]*5;
```

i coś takiego:

```
mojaPierwszaTablica[3] = mojaPierwszaTablica[3] + 17;
```

a nawet coś takiego:

```
mojaPierwszaTablica[4] ++;
```

Jedna komórka tablicy całkowitoliczbowej to normalna zmienna typu **int** i można na niej wykonywać wszelkie możliwe operacje. Analogicznie będzie dla typów **char**, **double** i **bool**.

Zadanie 0. Zadeklaruj 10-elementową tablicę liczb całkowitych. W pętli **for** do **i**-tej komórki tablicy wpisz $\mathbf{i^2}$. Również za pomocą pętli **for** wypisz zawartość swojej tablicy na ekranie – każdą liczbę w osobnej linijce. Zauważ, że jeżeli w drugim **for**ze ponownie wpiszesz **for(int i=** ... , to wyświetli się błąd typu *"redefinicja i"*. Skorzystaj z innej zmiennej lub rozpocznij drugiego **for**a po prostu od **for(i=** ...

Wykonaj ponownie to zadanie korzystając z pętli **while** oraz **do-while**.

W celach czysto eksperymentalnych popełnij błąd: wypisz zawartość tablicy w pętli **for** od **i**=1 do **i**<11, najprawdopodobniej na ekranie wyświetli się na końcu jakaś dziwna wartość – pętla wydrukowała zawartość tej komórki w pamięci RAM, która jest zaraz po tablicy.

Powyższy eksperyment pokazał, że w pamięci komputera mogą znajdować się „śmieci". Jeżeli nie inicjujesz tablicy podczas jej deklarowania, lecz np. za pomocą jakiegoś algorytmu, to przed podjęciem jakichkolwiek działań dobrym nawykiem jest *wyzerowanie* tablicy – wpisanie zer do jej komórek.

Zadanie 1. Zadeklaruj tablicę 10-elementową liczb całkowitych, korzystając z jakiejkolwiek pętli do **i**-tej komórki wpisz:
3*i+1, jeżeli **i** jest nieparzyste,
i/2, jeżeli **i** jest parzyste.
wyświetl tablicę na ekranie. Następnie, w jakiejkolwiek pętli, podwój zawartość każdej komórki.

Zadanie 2. Zadeklaruj tablicę 20-elementową liczb całkowitych. Do pierwszej i drugiej komórki wpisz „ręcznie" jedynki. Następnie, w jakiejkolwiek pętli, do każdej następnej komórki wpisz sumę dwóch poprzednich komórek.

Zadanie 3. Zadeklaruj 10-elementową tablicę typu **double**, wykonaj na niej jakieś interesujące operacje.

Zadanie 4. Zadeklaruj 10-elementową tablicę typu **char**, wpisz do tej tablicy jakieś duże literki już podczas deklaracji, np: **char tab[10]={'L','U','D','O','L','F','I','N','K','A'};** - nie zapomnij o apostrofach dookoła literek. Wyświetl tablicę. Następnie w pętli zwiększ zawartość każdej komórki tablicy o **32**. Ponownie wyświetl tablicę (jako literki, więc **%c**). W jaki sposób zmieniła się wielkość liter i dlaczego? Sprawdź w kodach ASCII.

Wiele algorytmów wymaga przesuwania lub zawijania zawartości tablic. Poniższy kod przesuwa liczby w dół, gubiąc niestety pierwszą liczbę:

```
int tab[10]={1,8,9,2,97,33,76,2,-3,4};
for(int i=9;i>=0;i--) tab[i]=tab[i-1];
tab[0]=0;
for(i=0;i<10;i++) printf("%d ",tab[i]);
```

Zadanie 5. Pozmieniaj powyższy kod tak, by tablica była przesuwana w drugim kierunku.

Zadanie 6. Pozmieniaj powyższy kod tak, by tablica była przesuwana o dwa pola w dowolnym kierunku.

Zadanie 7*. Pozmieniaj zadanie **6** w taki sposób, by wartości, które obecnie są gubione, były „zawijane" z drugiej strony tablicy – np. dla powyższych danych końcowym wynikiem byłoby **-3,4,1,8,9,2,97,33,76,2**. Przyda się jakaś zmienna pomocnicza oraz/lub operacja **%** (modulo).

Zadanie 8. Zadeklaruj tablicę typu **int** lub **double** z dowolnymi wartościami. Przeglądając tablicę zliczaj jej elementy, na końcu wyświetl sumę liczb oraz średnią liczb.

Zadanie 9. Zadeklaruj tablicę typu **int** lub **double** z dowolnymi wartościami. Przeglądając tablicę znajdź minimum i maksimum jej wartości– przydadzą się **if**y oraz dwie zmienne pomocnicze.

Poniższy kod wczytuje tablicę z klawiatury:

```
int tab[10];
int licznik = 0;

while(licznik<10){
        printf("Podaj liczbę całkowitą: ");
        scanf("%d",&tab[licznik]);
        licznik++;
}

for(int i=0;i<10;i++) printf("%d  ",tab[i]);
```

Zadanie 10. Przerób powyższy kod tak, by można było wczytywać tylko liczby dodatnie – jeżeli podana liczba była ujemna, to nie zwiększaj licznika. Inaczej mówiąc, zwiększaj go tylko po dodatnich liczbach.

4a.2 Podstawowy algorytm sortowania – sortowanie bąbelkowe

Tablice są dobrą okazją do przećwiczenia różnych algorytmów sortowania. Najbardziej popularnym i najprostszym sposobem jest **sortowanie bąbelkowe** (ang. **bubble-sort**). Polega ono na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim sortuje się tablicę. Można napisać ten algorytm na dwa sposoby:

1) Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany:

```
int tab[10]={-3,4,1,8,9,2,97,33,76,2};
bool zamienionoCos = true;

while(zamienionoCos){
        zamienionoCos = false;
        for(int i=0;i<9;i++){
                if(tab[i]>tab[i+1]){
                        int pom=tab[i];
                        tab[i]=tab[i+1];
                        tab[i+1]=pom;
                        zamienionoCos = true;
                }
        }
}

for(int i=0;i<10;i++) printf("%d ",tab[i]);
```

Żeby można było wejść do pętli **while**.

Jeszcze nic nie zamieniliśmy miejscami, bo nie weszliśmy jeszcze do **for**a.

Tylko do **9**, bo za chwilkę sprawdzimy element o numerze **i+1**.

Sortujemy rosnąco, więc jeżeli dwa kolejne elementy są posortowane malejąco, to je zamieniamy i wstawiamy odpowiednią wartość do zmiennej **zamienionoCos**.

Zadanie 11a. Prześledź działanie tego algorytmu na kartce papieru dla jakichś krótszych danych, np. dla tablicy pięcioelementowej.

2) Nie patrzymy na to, czy ciąg jest już posortowany, czy nie - sortujemy „na siłę”:

```
int tab[10]={-3,4,1,8,9,2,97,33,76,2};

for(int i=0;i<10;i++){
        for(int j=0;j<10;j++){
                if(tab[i]<tab[j]){
                        int pom=tab[i];
                        tab[i]=tab[j];
                        tab[j]=pom;
                }
        }
}

for(int k=0;k<10;k++)  printf("%d ",tab[k]);
```

Zadanie 11b. Zasymuluj również ten algorytm na kartce papieru dla jakiejś tablicy pięcioelementowej.

Drugi algorytm wydaje się mieć bardziej zwięzły zapis, lecz pierwszy algorytm szybciej działa. Algorytm wykonuje faktyczną pracę podczas zamieniania zawartości komórek. Wpisz w pierwszym algorytmie wewnątrz **if**a instrukcję:

```
printf(”Uwaga, ja pracuję !\n”);
```

Uruchom program i policz, ile razy dla danych **-3,4,1,8,9,2,97,33,76,2** został wypisany komunikat o pracy. Powtórz ten sam eksperyment dla drugiego algorytmu sortowania bąbelkowego i wyciągnij wnioski.

Zadanie 12. Przerób oba algorytmy sortowania bąbelkowego tak, by sortowały malejąco. Sprawdź, że algorytmy te zadziałają też dla sortowania literek (typ danych **char**).

Zadanie 13. Dodaj w jednym z algorytmów sortowania podawanie danych z klawiatury.

Zadanie 14*. Przerób zadanie **13** tak, by użytkownik podawał też ile (<=10) danych chce wprowadzić. Jeżeli użytkownik poda np. 7 danych, to posortuj i wyświetl tylko te 7 danych ignorując puste miejsca na końcu tablicy.

Zadanie 15. W posortowanym ciągu znajdź jego medianę. Jeżeli ciąg ma parzystą liczbę elementów, to wypisz dwa środkowe. Czy do znajdowania mediany trzeba używać pętli?

4a.3 Zastosowanie stałych w tablicach

W obu algorytmach sortowania bąbelkowego korzystaliśmy z tablic 10-elementowych. Może się jednak zdarzyć, że ulepszając program zechcemy powiększyć rozmiar tablicy (np. do 15). W pierwszym algorytmie musielibyśmy zamienić 10 na 15 w trzech miejscach (w deklaracji, w pętli **for** – zamiana z 9 na 14, przy wyświetlaniu). W drugim algorytmie – aż w czterech miejscach. Jeżeli w programie trzeba coś ręcznie poprawiać, to prawie zawsze przegapi się którąś poprawkę - z reguły w najważniejszym miejscu. Jest to bardzo uciążliwe przy pisaniu np. gier planszowych.

Przerobimy drugi algorytm sortowania - pod includami a nad funkcją **main** wpisz:

```
#define MAX 10
```

Tu wyjątkowo nie ma średnika !

Następnie w czterech miejscach zamień liczbę **10** na zmienną **MAX**. Uruchom program i sprawdź, że zadziała tak samo dobrze. Teraz zwiększ **MAX** do **15** - nie za pomocą **MAX=MAX+15;** (i tak by się nie dało – możesz to sprawdzić), lecz w sekcji **#define**. Następnie dopisz jeszcze pięć różnych liczb podczas deklarowania tablicy (wewnątrz klamerek).

Jeżeli przewidujemy, że rozmiar tablicy będzie się zmieniał podczas pisania programu, to na prawdę warto stosować stałe jako rozmiar tablicy.

4a.4 Podawanie parametrów z linii poleceń

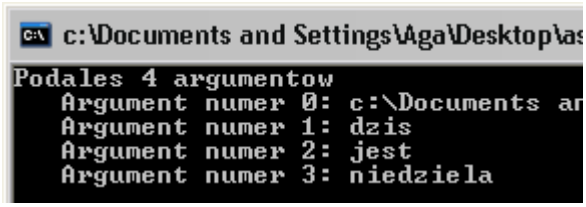
Nie zawsze chce się nam podawać dane z klawiatury. Można je też podawać z linii poleceń. Wpisz treść programu:

```
#include "stdafx.h"
#include "conio.h"

int main(int argc, char* argv[]){

        printf("Podales %d argumentow\n",argc);
        for(int i=0;i<argc;i++)
                printf("   Argument numer %d: %s\n",i,argv[i]);

        getch();   // żeby okienko od razu nie mignęło i nie znikło
        return 0;
}
```

Skompiluj i uruchom program. Zauważ, że wypisał lokalizację pliku **exe**.

Otwórz TotalCommandera. Doklikaj się do katalogu ze swoim projektem, a właściwie do podkatalogu **Debug**. Na dole TotalCommandera wpisz nazwę pliku wykonywalnego **exe** oraz jakieś dodatkowe wyrazy i naciśnij ENTER, np:

Program powinien wypisać:



Parametr **argc** podaje liczbę argumentów wpisanych w linii poleceń – jeżeli nic nie wpiszemy, to parametr ten wynosi **1**, bo nazwa pliku **exe** też jest przecież jakimś argumentem. Parametr **argv** jest tablicą tych argumentów. **char*** oznacza wskaźnik do danych typu **char** (o wskaźnikach będzie później), czyli do łańcuchów znaków (ang. **string** – stąd **%s**).

4a.5 Tablica w tablicy – sortowanie przez zliczanie

Załóżmy, że mamy dwie tablice: **int tab[5]={4,1,5,3,2};** oraz **int a[5]={2,3,1,5,4};**.

Zadanie 16. Jakie wartości zawierają: **tab[a[1]]**, **tab[a[2]+a[0]]**, **a[tab[2]-1]** i dlaczego właśnie takie?

Korzystanie z dodatkowej tablicy w tablicy przydaje się w **sortowaniu przez zliczanie** (ang. **counting-sort**). Stosujemy ten algorytm, gdy nasz zbiór danych do posortowania zawiera się w jakichś określonych granicach (np. od 0 do 17) i gdy w zbiorze danych te same wartości występują wielokrotnie, na przykład: **{1,2,17,2,2,3,1,3,17,...itd...}**.

Tablice: wejściowa (z ciągiem) oraz pomocnicza – jeżeli wejściowa zawiera liczby od **0** do np. **17**, to pomocnicza ma **18** (nie ważne, że **18** – ważne, że **17+1**) elementów.

Sposób działania:

1. Wyzeruj tablicę pomocniczą.
2. Przeglądaj w pętli tablicę wejściową – jeżeli napotkasz na element **α**, to zwiększ o jeden zawartość komórki *pomocnicza*[**α**].
3. Przeglądaj tablicę pomocniczą w pętli od **0** do ... (w naszym przykładzie: **<18**) – w **i**-tym obiegu pętli wydrukuj liczbę **i** aż *pomocnicza*[**i**] razy.

Poniższy kod realizuje algorytm sortowania przez zliczanie dla danych **1,5,1,0,5,17,14,3,4,7,8,7,1,4**:

```
int wej[14]={1,5,1,0,5,17,14,3,4,7,8,7,1,4};
int pom[18];

for(int i=0;i<18;i++) pom[i]=0;

for(i=0;i<14;i++) pom[ wej[i] ]++;

// jeżeli chcesz, to wydrukuj w celach testowych
// zawartość tablicy pomocniczej:
/*printf("\n");
  for(i=0;i<18;i++)  printf("%d wystepuje %d razy\n",i,pom[i]);
  printf("\n"); */

for(i=0;i<18;i++){
        for(int j=0;j<pom[i];j++) printf("%d ",i);
}
```

Zadanie 17. Prześledź działanie powyższego algorytmu na kartce dla danych **1,5,1,0,5,17,14,3,4,7,8,7,1,4**.

Zadanie 18. Przerób powyższy kod dla mniejszej liczby danych – wyrzuć **17** i **14**. Ile komórek będzie teraz potrzebne w tablicy **pom**?

4a.6 Tablice jednowymiarowe typu char.

Pojawił się już przykład, w którym zadeklarowano i zainicjowano tablicę znaków w następujący sposób:

```
char tab[10]={'L','U','D','O','L','F','I','N','K','A'};
```

Można to zapisać o wiele wygodniej:

```
char tab[11]="LUDOLFINKA";
```

Nie 10, ponieważ wyrazy deklarowane w cudzysłowach muszą mieć oznaczony koniec – służy to tego specjalny znaczek \0 (czyli NULL). Kompilator sam go dokleja, nie piszemy więc ręcznie "LUDOLFINKA\0";

Sprawdź, że jeżeli zamienisz 11 na 10, to podczas kompilacji pojawi się error:
error C2117: 'LUDOLFINKA' : array bounds overflow
czyli „przekroczenie zakresu tablicy".

Zadanie 19. Sprawdź, że znaczek **\0** jest „niewidzialny" – zamiast
`for(int i=0;i<10;i++) printf("%c",tab[i]);` wpisz `for(int i=0;i<11;i++) printf("%c",tab[i]);`

Tablicę typu **char** można wyświetlać też w krótszy sposób: `printf("%s",tab);`

%s pochodzi od słowa string, czyli „łańcuch znaków"

Zadanie 20. Wyrzuć ostatnie 5 liter z wyrazu (ale nie zmieniaj rozmiaru tablicy) i znowu go wyświetl na ekranie (w ten krótszy sposób) – zauważ, że kompilator sam wie, ile znaków powinien wyświetlić.

Zadanie 21. Zadeklaruj dwie tablice **char**owe z dwoma różnymi wyrazami. Następnie, w jakiejkolwiek pętli zamień je miejscami literka po literce.

Zadanie 22. Zadeklaruj tablicę **char**ową. W pętli przejrzyj każdy znak w tablicy i wyświetl, czy jest to mała literka, duża literka, liczba, czy coś innego.
Duże literki mają kody ASCII od 65 do 90, małe – od 97 do 122, cyfry – od 48 do 57.

Zadanie 23. Zadeklaruj tablicę **char**ową i zainicjuj ją małymi i dużymi literami. W pętli przejrzyj każdy znak – zamień małą literę na dużą i vice versa.

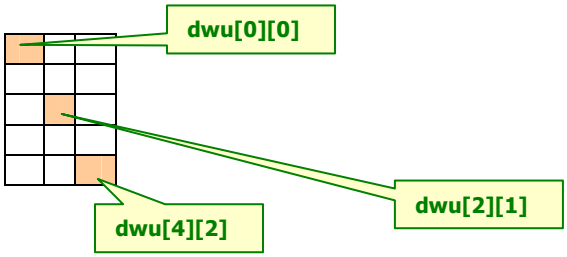
Zadanie 24. Zadeklaruj tablicę **char**ową i zainicjuj ją małymi literami. W pętli przejrzyj każdy znak – zamień każdą samogłoskę na dużą literę, a spółgłoskę na gwiazdkę.

4a.7 Tablice wielowymiarowe

Aby zadeklarować tablicę dwuwymiarową o 5 wierszach i 3 kolumnach dla liczb całkowitych należy napisać:

```
int dwu [5][3];
```

[wiersze] [kolumny] a nie na odwrót !!! Tutaj też obowiązuje numeracja od zera – zarówno w wierszach, jak i w kolumnach. Będziemy więc numerować komórki: {0,1,2,3,4}x{0,1,2}.



Możemy inicjować tablicę komórka po komórce:

```
dwu[0][0] = 1;
dwu[0][1] = 13;
...
dwu[4][2] = 6;
```

lub za pomocą pętli:

```
for(int i=0;i<5;i++)
        for(int j=0;j<3;j++)
                dwu[i][j] = i*i-i*i+j*j;
```

Można oczywiście zapisać to wszystko w jednej linijce, ale wtedy nie za bardzo byłoby widać „co jest co".

lub od razu podczas deklaracji:

```
int dwu[5][3] = { {1 , 2 , 3},
                  {16 , 7 , 18},
                  {9 , 0 , -2},
                  {80 , 2132 , 1},
                  {0 , 0 , 1}
                };
```

Po ostatnim wierszu nie ma przecinka !!! Po poprzednich są przecinki. Każdy wiersz jest otoczony klamerkami.

ŚREDNIK, o którym prawie wszyscy zapominają :(

Podobnie jak to było dla tablic jednowymiarowych, warto korzystać ze stałych przechowujących rozmiary tablicy, np:

```
#define RZAD 5
#define KOL 3
....
int dwu[RZAD][KOL];
```

Osobna sekcja #define dla każdej zmiennej.

Poniższy kod pobiera z klawiatury dane do tablicy typu **double** (3 rzędy, 2 kolumny):

```
double tab[3][2];

for(int rzad=0;rzad<3;rzad++){
        for(int kol=0;kol<2;kol++){
                printf("Podaj tab[%d][%d]: ",rzad,kol);
                scanf("%lf",&tab[rzad][kol]);
                getchar();
        }
}

for(rzad=0;rzad<3;rzad++){
        for(int kol=0;kol<2;kol++) printf("%.2lf ",tab[rzad][kol]);
        printf("\n");   // enter dopiero po całym wierszu !!!
}
```

Zadanie 25. Zapytaj użytkownika, czy chce obliczyć wyznacznik macierzy 2x2, czy 3x3. W zależności od jego wyboru pobierz z klawiatury odpowiednią ilość liczb, oblicz i wyświetl wyznacznik macierzy. Do tego zadania wystarczy użyć jednej tablicy – w przypadku wyboru macierzy 2x2 najwyżej nie będzie ona wypełniona w całości (ale to przecież w niczym nie przeszkadza).

Zadanie 26. Utwórz tablicę o dwóch kolumnach i o dowolnej liczbie wierszy. Następnie zmodyfikuj algorytm sortowania bąbelkowego w taki sposób, by sortował zawartość tablicy względem drugiej kolumny. Jeżeli w drugiej kolumnie trafią się dwie takie same wartości, to wtedy możesz rozpatrzyć wartości z pierwszej kolumny.
A więc dane:

2	**3**
4	**2**
1	**2**
2	**0**

po takim posortowaniu będą miały postać:

2	**0**
1	**2**
4	**2**
2	**3**

Zadanie 27*. Utwórz tablicę typu **double** o dwóch kolumnach i o dowolnej liczbie wierszy. Wyobraź sobie, że w jednym wierszu znajdują się część rzeczywista i część urojona liczby zespolonej – każda w osobnej kolumnie. Posortuj tablicę bąbelkowo rosnąco względem modułu liczby zespolonej – a więc jeżeli dla wiersza o numerze **i** wyjdzie Ci większy moduł, niż dla wiersza o numerze **i+1**, to zamień te wiersze miejscami.

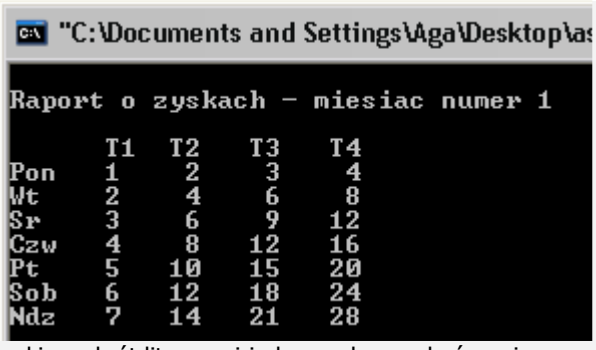
Tablice mogą mieć jeszcze więcej wymiarów, chociaż powyżej trzeciego wymiaru trudno to sobie wyobrazić. Oto przykład użycia tablicy trójwymiarowej:

```
#include "stdafx.h"

#define TYDZIEN 4
#define DZIEN 7
#define MIESIAC 2

int main(int argc, char* argv[]){

        int zyski[TYDZIEN][DZIEN][MIESIAC];

        for(int mies=0;mies<MIESIAC;mies++)
                for(int rzad=0;rzad<TYDZIEN;rzad++)
                        for(int kol=0;kol<DZIEN;kol++) zyski[rzad][kol][mies] = (rzad+1)*(kol+1)*(mies+1);

        for(mies=0;mies<MIESIAC;mies++){
                printf("\nRaport o zyskach - miesiac numer %d\n\n",mies+1);
                printf("       Pon  Wt   Sr   Cz   Pt   Sb   Ndz\n");
                for(int rzad=0;rzad<TYDZIEN;rzad++){
                        printf("Tydz.%d) ",rzad+1);
                        for(int kol=0;kol<DZIEN;kol++) printf("%4d ",zyski[rzad][kol][mies]);
                        printf("\n");
                }
        }

        printf("\n");
        return 0;
}
```

Zadanie 28.
Zmodyfikuj powyższy kod w taki sposób, by dla każdego miesiąca tabelka drukowała się w postaci „pionowej”.

Wskazówki: przyda się pomocnicza tablica **char**owa dwuwymiarowa na przechowywanie nazw dni tygodnia:

```
char dniTygodnia[DZIEN][4]=
    {"Pon","Wt ","Sr ","Czw","Pt ","Sob","Ndz"};
```



Na każdy dzień zużyjemy **4** znaki – maksymalnie trzy znaki na skrót literowy i jeden znak na zakończenie wyrazu przez **\0**. Aby wydrukować nazwę np. dla **wtorku**, trzeba napisać: **printf(”%s ”,dniTygodnia[1]);** - zauważ, że nie podajemy tutaj drugiego wymiaru!

Zadanie 29*. Napisz program, w którym użytkownik poda, od którego dnia tygodnia zaczyna się dany rok. Program ma za zadanie wypisać kalendarz na dany rok – należy uwzględnić różne długości miesięcy i zostawić puste miejsca, jeżeli miesiąc zaczyna się lub kończy w środku tygodnia. Jeden miesiąc nie może być wypisany w jednej linijce, musi zajmować „prostokąt”. Wybierz sam, czy wolisz dni tygodnia umieszczać w kolumnach, czy w rzędach. Czy w tym zadaniu potrzebne są tablice?

4a.8 Tablice równoległe

Jedna tablica przechowuje dane jednego typu – albo **int**, albo **double**, albo **char** albo **bool**. Nie jest, niestety, możliwe przechowywanie w jednej kolumnie np. nazwy samochodu, a w drugiej pojemności silnika, gdyż są to dane różnych typów. Dopóki nie zapoznamy się ze strukturami (typ **struct** – będzie dużo później), będziemy takie dane przechowywać w tablicach równoległych. Przykład:

```
#include "stdafx.h"

#define PLANETY 9

int main(int argc, char* argv[]){

char nazwaPlanety[PLANETY][7]= {"Merkury","Wenus","Ziemia","Mars","Jowisz","Saturn","Uran","Neptun","Pluton"};

int odlegloscOdSlonca[PLANETY]={58,108,150,228,778,1427,2869,4498,5900};

// rok względem roku ziemskiego
double rok[PLANETY]={0.2408 , 0.6152 , 1.0 , 1.8808 , 11.8637 , 29.4484 , 84.0711 , 164.8799 , 248.09};

printf("Nazwa    Odleglosc    Rok\n");

for(int i=0;i<PLANETY;i++) printf("%7s %10d    %6.4lf\n",nazwaPlanety[i],odlegloscOdSlonca[i],rok[i]);

return 0;
}
```

Jeżeli wyrazy są różnych długości, to możemy je wyrównać do prawej nakazując im zajęcie np. 7 znaków.

Tablice równoległe nie są zbyt wygodne – usuwając wpis z jednej tablicy można łatwo zapomnieć o uaktualnieniu pozostałych tablic.

4a.9 Zadanie na sobotę – bardzo mała baza danych

Przekształć kod programu z planetami: dopisz menu, w którym użytkownik będzie miał do wyboru:
- posortowanie planet malejąco/rosnąco według pierwszej litery nazwy/odległości/roku
- usunięcie planety o danym numerze w tablicy – jeżeli użytkownik wybierze planetę ze środka tablicy, to planety położone w dalszym końcu tablicy trzeba „podciągnąć” o jeden wiersz do góry.
- dodanie nowej planety (wypadałoby już na początku dać trochę większy rozmiar tablicy)
- zmiana danych – użytkownik podaje numer w tablicy i nową długość roku lub odległość od słońca.

4a.10 Zadanie na niedzielę – tablice boolowskie

Tablice mogą być typu **bool**. Poniższy kod programu wypełnia tablicę 8x3 (3 zmienne, 2^3 możliwości) wartościami **true** lub **false** i sprawdza prawdziwość zdań logicznych. Przerób program tak, by działał dla 4 zmiennych i odpowiedniej liczby możliwości – podaj własne zdanie logiczne zawierające 4 zmienne. Postaraj się uogólnić zawartość pierwszego **for**a – dobrym pomysłem będzie dodanie w środku jakiegoś wewnętrznego **for**a.

```
#include "stdafx.h"
#include "math.h"

#define COL 3

int main(int argc, char* argv[]){

	bool tab[8][COL];
	int ROW = (int)pow(2,COL);

	for(int i=0;i<ROW;i++){
		// poniższe trzy linijki można ładnie uogólnić jednym forem :)
		tab[i][0]= ( (i%8) >= 4 );
		tab[i][1]= ( (i%4) >= 2 );   // <- w nawiasie znajduje się tak naprawdę true lub false
		tab[i][2]= ( (i%2) > 0 );
	}

	printf("Sprawdzimy wartosc zdania (!x || y) && z\n\n");
	printf("x y z\n\n");

	for(i=0;i<ROW;i++){

		for(int j=0;j<COL;j++) printf("%d ",tab[i][j]);

		if(  (!tab[i][0] || tab[i][1]) && tab[i][2] == true )
			printf(" T \n");
		else
			printf(" F \n");

	}

	printf("\n");

	return 0;
}
```

Dlaczego program będzie tak samo dobrze działał, jeżeli wykasujemy fragment == true ?

(*) Ciekawostka: tablice w Javie mogą być „szarpane" – ich wiersze mogą mieć różną długość (co jest bardzo przydatne w wielu metodach numerycznych). Tablicę dwuwymiarową zadeklarowalibyśmy w Javie w następujący sposób: **int[][] mojaTablica = new int[3][5];**

4a.11 Najczęstsze błędy

Mylenie rodzajów nawiasów: rozmiar tablicy podajemy w **[]** a zawartość w **{}**.
Brak średnika po klamerkach **{}** zawierających zawartość tablicy.
Pisanie średnika po sekcji **#define**.
Numerowanie elementów tablicy od jedynki i wychodzenie poza tablicę.
Nieprawidłowe wykonywanie operacji na pojedyńczych komórkach.
Zapominanie o apostrofach w tablicach **char**owych.
Nierozróżnianie rzędów i kolumn w tablicach dwuwymiarowych.
Brak przecinków/klamerek/średników w inicjowaniu tablic wielowymiarowych.
Tablice równoległe: zapominanie o uaktualnianiu wszystkich tablic przy różnych zmianach/kasowaniu.

4a.12 Quiz

1. Która instrukcja deklaruje tablicę w prawidłowy sposób?
a) int tab; b) tab{10}; c) int tab[10]; d) array int tab[10];

2. Jaki numer ma ostatni element w tablicy złożonej z 29 elementów?
a) 29 b) 28 c) 30 d) Nie da się tego określić.

3. Która z podanych tablic jest dwuwymiarowa?
a) int tab[20,20]; b) int tab[20*20] ; c) int tab[20][20] ; d) char tab[20];

4. Która instrukcja zwiększa o dwa zawartość siódmej komórki tablicy?
a) tab[7]+=2; b) tab[6]+=2; c) tab[8]+=2; d) tab +=2;

5. Jaki znak kończy wyraz zapisany w cudzysłowach?
a) **.** b) **\n** c) **;** d) **\0**

6. Ile komórek zawiera tablica zadeklarowana jako **int tab[2][4][2][5];** ? _____ .