Wykład 8

8. Tablice

8.1. Definiowanie tablic

8.2. Dostęp do elementów tablicy

8.3. Inicjacja tablic

8.4. Przekazywanie tablic do funkcji

8.5. Tablice znakowe

8.6. Przekazywanie tablic znakowych do funkcji

8. Tablice

Tablica to złożona struktura danych składająca się z elementów tego samego typu. Zajmuje ona ciągły obszar pamięci o rozmiarze niezbędnym do zapamiętania wszystkich jej elementów. Dostęp do elementu tablicy jest realizowany przez podanie nazwy tablicy oraz indeksu (indeksów – w przypadku tablic wielowymiarowych) określającego położenie elementu w tablicy.

8.1. Definiowanie tablic

Definicja tablicy N-wymiarowej (zmiennej tablicowej) o wymiarach w_1, w_2, ... , w_N ma następującą postać:

typ_elementu **nazwa_tablicy**[w_1][w_2] ... [w_N];

Elementy tablicy mogą być dowolnego typu z wyjątkiem referencyjnego oraz typu void. W szczególności, tablica może zawierać:
- elementy typu skalarnego (całkowitego lub rzeczywistego),
- elementy typu wyliczeniowego (enum),
- wskaźniki, struktury, unie, inne tablice, obiekty.

Nie ma ograniczeń liczby wymiarów tablicy, ale *całkowity rozmiar* zmiennej tablicowej jest ograniczony, np. nie może przekroczyć 64 KB w systemie 16-bitowym BC++3.1 (można obejść to ograniczenie stosując tablice typu huge, np. long huge tab[30000]). W systemach 32-bitowych VC++ i BBuilderC++ dopuszczalne rozmiary tablic zależą od rozmiaru dostępnej pamięci wirtualnej.

```
typedef int ttab[7];   // typ tablicowy ttab – tablica 7 liczb całkowitych

ttab tablica;      // definicja zmiennej tablicowej typu ttab

char st[20];       // tablica 20 znaków

unsigned long w[7];    // tablica 7 liczb typu unsigned long

double x[5];       // tablica 5 liczb typu double

int  *ws[7];       // tablica 7 wskaźników na obiekty typu int

float x[3][2];     // dwuwymiarowa tablica liczb typu float
                   // tablica 3 elementów typu tablica 2 elementowa
char *twsk[3][3]; // dwuwymiarowa tablica wskaźników na znaki
```

8.2. Dostęp do elementów tablicy

Dostęp do elementu tablicy jest realizowany poprzez podanie jego pozycji (indeksu (ów)) w nawiasach kwadratowych, np. tab[5], t[2][3].

W języku C/C++ numeracja elementów tablicy zaczyna się od zera. Pierwszy element tablicy ma wszystkie indeksy równe zero. Dostęp do pierwszego elementu tablicy wielowymiarowej tab wygląda następująco:

```
tab[0][0] ... [0].
```

Dla tablicy tab[wym1][wym2] ... [wymN]

pozycja ostatniego elementu jest określona przez

```
tab[wym1 - 1][wym2 - 1] ... [wymN - 1].
```

W szczególności, tablica int t[4] składa się z elementów t[0], t[1], t[2], t[3]. Obowiązek kontroli zakresów tablicy spoczywa na programiście. W przypadku przekroczenia zakresów tablicy dane są umieszczane poza obszarem zajmowanym przez tablicę. Może to prowadzić do zniszczenia innych danych. W efekcie powstają trudne do wykrycia błędy.

Zapełnianie tablicy danymi można zrealizować następująco:

```
void main()
{
long t[4];
int i;

for (i = 0; i < 4; i++)  t[i] = 2*i;

cout << "Zawartość tablicy:  \n" << endl;

for (i = 0; i < 4; i++)  cout << " t[ " << i << " ] = " << t[i] << endl;
}
```

8.3. Inicjacja tablic

Tablicę można zainicjować danymi w momencie podania jej definicji. Lista inicjatorów tablicy jest ujęta w nawiasy klamrowe. Np.

```
const int N=4;   // rozmiar tablicy
int t[N] = { 2, 3, 4, 5 }.
```

Wówczas, t[0] = 2, t[1] = 3, t[2] = 4, t[3] = 5.

Zasady inicjacji:

- lista inicjatorów nie może zawierać więcej elementów niż wynosi rozmiar tablicy (kompilator zasygnalizuje błąd),
- w języku C inicjatorem może być stała lub identyfikator zmiennej,
- jeśli inicjatorów jest mniej niż wynosi rozmiar tablicy, wówczas w przypadku *zewnętrznych i statycznych* tablic liczb lub znaków pozostałe elementy są inicjowane zerami, natomiast w przypadku tablicy wskaźników są inicjowane wskazaniami pustymi (NULL);
- w przypadku tablic automatycznych – jeśli nie są zainicjowane to mogą mieć dowolne wartości.

Przykłady.

```
double A[10] = {1.25, 2, 15 };      // A[0]=1.25, A[1]=2.0, A[2] =15.0

long B[4][2] = { {1,2}, {3,4} };      // B[0] = {1,2}, B[1] = {3,4}

long B[4][2] = { 1, 2, 3, 4 };    // B[0][0]=1, B[0][1]=2, B[1][0]=3, B[1][1]=4,
```

W przypadku tablicy zewnętrznej A zainicjowano pierwsze trzy elementy, pozostałe zostaną ustawione na zero. Definicja tablicy zewnętrznej B pokazuje możliwość użycia nawiasów klamrowych dla wygodniejszej orientacji w strukturze inicjowanej tablicy.

W przypadku tablicy wielowymiarowej jej elementy zajmują pamięć w kolejności zmiany *najbardziej skrajnego prawego* indeksu. Np.

```
int t[2][3][2];       // tablica 3-wymiarowa; 12 elementów typu int
```

Elementy tablicy (wierszami):

```
t[0][0][0],   t[0][0][1],   t[0][1][0],   t[0][1][1],   t[0][2][0],   t[0][2][1],
t[1][0][0],   t[1][0][1],   t[1][1][0],   t[1][1][1],   t[1][2][0],   t[1][2][1].
```

Istnieje również sposób inicjalizacji tablic, w którym nie podaje się rozmiaru tablicy, np.

```
int  tab[ ] = { 1, 2, 3 };   // tablica tab[3];
                             // rozmiar tablicy n = sizeof(tab) / sizeof(int) = 3
                             // lub n = sizeof(tab) / sizeof(tab[0]) = 3
```

Kompilator sam przelicza ile podano liczb w klamrze i rezerwuje pamięć dla wszystkich elementów.

8.4. Przekazywanie tablic do funkcji

Do funkcji możemy przekazywać pojedyncze elementy tablicy przez wartość – tak jak zwykłe zmienne. Przekazywanie wszystkich elementów tablicy jako pojedynczych zmiennych wymagałoby użycia zbyt dużej liczby zmiennych.

```
void funkcja(int &x, int y);
int t[5];
funkcja( t[1], t[2] );       // przekazanie t[1] przez referencję
                             // przekazanie t[2] przez wartość
```

W praktyce przekazujemy do funkcji adres początku tablicy oraz liczbę elementów tablicy (oprócz tych przypadków, w których koniec tablicy oznaczony jest znakiem specjalnym, np. NULL).

```
void pisz(int tab[5], int licz_elem)
{
      for (int i = 0; i < licz_elem; i++)  cout << tab[i] << ' – ';
}
```

Kompilator potraktuje definicję funkcji pisz tak jakby do funkcji przekazywany był wskaźnik na element typu int, czyli podana definicja funkcji pisz jest równoważna następującej:

```
void pisz(int *tab, int licz_elem)
{
      for (int i = 0; i < licz_elem; i++)  cout << tab[i] << ' – ';
}
```

Do funkcji void pisz(int tab[5], int licz_elem) można przekazywać tablice jednowymiarowe typu int o dowolnym wymiarze (mimo iż podano tab[5]). W wywołaniu funkcji należy przekazać adres pierwszego elementu tablicy.

Ponieważ nazwa tablicy jest *stałą* będącą adresem pierwszego jej elementu wywołanie funkcji dla tablicy int t[5] ma postać: **pisz(t, 5)**.
Poprawne jest również wywołanie: **pisz(&t[0], 5);**
Inne tablice jednowymiarowe typu int również można przekazywać do funkcji, np. dla int tabliczka[20] wywołanie postaci **pisz(tabliczka, 20)** lub jeśli chcemy wypisać mniej elementów niż 20: **pisz(tabliczka, 4).**

Ponieważ do funkcji jest przekazywany adres pierwszego elementu tablicy wszelkie zmiany zawartości tablicy są dokonywane bezpośrednio na jej elementach (a nie na ich kopiach).

Podczas przekazywania tablic jednowymiarowych do funkcji nie jest istotny ich rozmiar. Do funkcji *pisz* możemy zatem przekazać tablicę o elementach typu int o dowolnym rozmiarze. Dlatego w ogólnym przypadku wystarczy określić jakiego typu tablicy funkcja powinna się spodziewać:

```
void pisz1(int tab[ ], int licz_elem);        // prototyp funkcji
```

Podczas przekazywania tablic do funkcji można posłużyć się specyfikatorem typu *typedef*.

```
typedef  double ttab[3][2];    // typ tablicowy o nazwie ttab
                               // tablica – 3 wiersze, 2 kolumny
void pisz(ttab tablica, int n_wierszy, int n_kolumn);
```

Tablica wielowymiarowa jest tablicą, której liczba elementów jest określona przez pierwszy wymiar. W szczególności, tablica typu double tb[3][20] jest 3-elementową tablicą, której każdy element jest 20-elementową tablicą liczb typu double. Tablica składa się z 3 wierszy i 20 kolumn. Wiersze mają numery od 0 do 2, natomiast kolumny mają numery od 0 do 19.

W przypadku tablic wielowymiarowych pierwszy wymiar (skrajny lewy) zawsze może być pominięty podczas specyfikacji typu tablicowego, gdyż na podstawie znajomości pozostałych wymiarów można odnaleźć każdy element.

```
void pisz(double tb[ ][20], int wym1, int wym2)
{
  for (int i=0; i < wym1; i++)
    {   printf("\n");       for (int j=0; j < wym2; j++) printf("%6.2f",tb[i][j]);   }
}
```

Na przykład element tb[2][5] (wiersz o numerze 2, kolumna o numerze 5) jest oddalony od początku tablicy tb o (2 ∗ 20) + 5 elementów typu double. Znajomość drugiego wymiaru wystarczy do określenia położenia elementu tb[i][j] względem początku tablicy. Element tb[i][j] (wiersz i, kolumna j) jest przesunięty względem tb o (i ∗ 20) + j elementów typu double.

Dopuszczalna jest definicja typu w postaci:

typedef int tx[ ][7];

```
int tablica[3][7];            // przykładowa tablica o 7 kolumnach
int tablica1[9][7];           // inna tablica o 7 kolumnach

void pisz(tx tab, int wym1, int wym2);
```

Wywołanie: pisz(tablica,3, 7); pisz(tablica1, 9, 7);

8.5. Tablice znakowe

W języku C teksty (łańcuchy) przechowywane są w tablicach typu char, np.

```
 char tekst[20];
```

Teksty w tablicach znakowych są zakończone znakiem o kodzie zero ('\0' = NULL). W ten sposób można rozpoznać koniec ciągu znaków. Większość standardowych funkcji operujących na tekstach zakłada, że są one łańcuchami ASCIIZ. W przypadku wprowadzania tekstu za pomocą funkcji standardowych jest on automatycznie kończony znakiem NULL (np. gets(tekst)).

Przykład. 8.1. Wprowadzanie danych do tablicy znaków.

```
void main(void)
{   int i,k;
   char slowo[11];
   clrscr();   randomize();
   for (k=0; k < 7; k++)      // 7 tekstów
   {   for (i=0; i < 10; i++)   slowo[i] = 65 + random(20);   // losowanie
       slowo[i] = NULL;     // i = 10  dodajemy znak końca
       cout << "Slowo nr " << k << " = " << slowo << endl;  }
}
```

Tablice znakowe można inicjować na etapie definicji.

W przypadku stałych tekstowych ujętych w cudzysłów zawsze na końcu dołączany jest znak NULL.

```
char tekst[20] = { "Borland C++" };
char tekst1[20] = "Borland1";
```

Tablice tekst i tekst1 mają ostatni znak równy NULL.

W przypadku inicjacji tablicy za pomocą ciągu stałych znakowych znak NULL nie jest dołączany automatycznie. Ponieważ jednak wolne pozycje w tablicy *zewnętrznej lub statycznej* są wypełniane zerami w definicji:

```
      static char tekst2[20] = { 'B','o','r','l', 'a', 'n','d' };
```

łańcuch również zostanie zakończony znakiem NULL.
Należy jednak pamiętać, że w przypadku definicji, w której nie podajemy rozmiaru tablicy

```
      static char tekst[ ] = { 'B','o','r','l', 'a', 'n','d' };
```

zarezerwowanych zostanie dokładnie tyle bajtów pamięci ile podano znaków inicjujących. W rozpatrywanym przypadku powstanie tablica znaków składająca się z 7 elementów (nie wiadomo czy zero będzie na końcu). Jeżeli jednak napiszemy,

```
      char tekst1[ ] = { "Borland" };
```

to powstanie tablica, w której ostatnim (ósmym) znakiem będzie NULL.

W szczególności,

```
      cout << sizeof(tekst)   << endl;      // wartość 7
      cout << sizeof(tekst1) << endl;       // wartość 8
```

W przypadku definicji { 'B','o','r','l', 'a', 'n','d',"x" } kompilator zgłosi błąd.

Należy zapamiętać, że

- "a" jest łańcuchem złożonym z dwóch znaków 'a' oraz '\0';
- 'a' jest stałą znakową.

Na etapie definicji dopuszczalne jest podstawienie: char w[2] = "a";

Wpisywanie całego łańcucha do tablicy znakowej możliwe jest tylko na etapie inicjalizacji. Podstawienia

```
char tekst[20], kopia[20];

tekst[20] = "Borland";        tekst = "Borland";
```

są błędne.

Jeżeli wczytamy tekst z klawiatury do tablicy, to możemy przepisać go do innej tablicy za pomocą standardowej funkcji *strcpy* lub za pomocą przepisywania znak po znaku.

Przykład. 8.2. Kopiowanie zawartości tablicy znakowej (wykorzystanie gets)

```
 char  dana[80],  kopia[80],  kp[80];

 gets(dana);        // pobierz daną; enter kończy wprowadzanie;
                    // zamiast znaku enter ('\n') wstawiany znak '\0'
// brak kontroli liczby wczytywanych danych!
// dlatego wykorzystywana tablica dana[80] powinna być większa niż
// spodziewana liczba wprowadzanych znaków
// można wykorzystać funkcję cgets, w której określa się maksymalną
// liczbę wprowadzanych znaków

 strcpy(kopia, dana);          // przepisz daną
 cout << kopia << endl;        // wypisz
```

Przepisywanie znak po znaku (zero koniec):

```
  for (int i=0;  ; i++)
  {  kp[i] = dana[i];      if (dana[i] == NULL) break;  }
  cout << kp << endl;
```

Inny sposób kopiowania tablic znakowych zakończonych zerem:

```
int i=0;
while ( kp[i] = dana[i] ) i++;    // kopiuj dopóki nie pojawi się zero
cout << kp << endl;
```

lub inaczej:

```
while ((kp[i] = dana[i])!=0) i++;   // zero kończy kopiowanie
  cout << kp << endl;               // kopia nr 3
```

Przykład. 8.3. Kopiowanie zawartości tablicy znakowej (wykorzystanie cgets)

```
char dana[80], kopia[80];

printf("Wprowadz tekst: ");
// gets(dana);

 char *p;   // wskaźnik do początku wczytanego tekstu,
            // który zaczyna się od dana[2], tj. p == &dana[2]

 /*  Miejsce na 77 znaków i 78 znak NULL */
  dana[0] = 78;       // ograniczenie liczby wczytywanych znakow
                      // CR\NL zastępowane przez '\0'

  p = cgets(dana);  // dana[1] - liczba rzeczywiście wczytanych znaków;
                    // p - adres wczytanego tekstu

printf("\nWczytano %d znakow: \"%s\"\n",dana[1], p);  //wydruk tekstu

printf("Adres tekstu %p, dana[0] zaczyna sie od %p\n", p, &dana[0]);

printf("Wczytany tekst począwszy od dana[2] = %s\n", &dana[2]);

int len = int(dana[1])+1;                 // liczba znaków + znak '\0'
memmove(&dana[0], &dana[2], len);  // przepisywanie z dana[2] do
                                          //  dana[0]
// memmove dobre nawet przy obszarach zachodzących na siebie
strcpy(kopia, dana);
cout << kopia << endl;  //kopia
```

8.6. Przekazywanie tablic znakowych do funkcji

Jeżeli tablica znakowa kończy się znakiem NULL, to wystarczy przekazać do funkcji adres jej początku. Nie jest konieczne przekazywanie liczby elementów.

```
void pisz( char tekst[] )
{   cout << tekst << endl;  }

int len( char tekst[] )     // długość tekstu bez znaku \0
{
   for  (int i=0; tekst[i]; i++);   return i;
}
void main()
{       pisz("Borland");   cout << len("Borland") << endl;    }
```

Przykład. 8.4. Operacje na tablicy liczb (inicjacja i sortowanie).

Zdefiniować tablicę liczb całkowitych typu int o rozmiarze N, gdzie N - stała. Opracować funkcje:
a) inicjującą tablicę losowymi liczbami należącymi do przedziału [-100, 100);
b) wyprowadzająca zawartość tablicy na ekran - wierszami po 10 elementów;
c) znajdującą minimalny i maksymalny element w tablicy;
d) sortującą elementy tablicy rosnąco w oparciu o wybrany algorytm.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// definicje funkcji
void inicjuj(int t[], int k)
{
  randomize();
  for(int i=0; i<k; i++) t[i]=-100+random(200);
  printf("Inicjacja tablicy\n");
}

void wyprowadz(int t[], int k)
{
  for(int i=0; i<k; i++) {
    if (i % 10) printf("%8d",t[i]); else printf("\n%4d",t[i]);
    }
  printf("\n");     getch();   }
```

```
void znajdz_min_max(int t[], int k, int &mn, int &mx)
{
  mn=t[0];   mx=t[0];
  for (int i=0; i<k; i++)  {  t[i] < mn ? mn=t[i] : 1;
                              t[i] > mx ? mx=t[i] : 1;  }
}

void sortuj(int t[], int k)
 {
  int temp;
  printf("\nSortowanie tablicy - prosta zamiana (bąbelkowe)\n");
  for (int i=0; i<k; i++) {
    for (int j=0; j<k -1-i; j++) {
      if (t[j]>t[j+1]) {
        temp=t[j];
        t[j]=t[j+1];
        t[j+1]=temp;
      }
    }
  }
  printf("Tablica posortowana\n");
}

void main()   {              // program główny

  const int N=70;
  int tab[N];
  int min, max;

  clrscr();
  inicjuj(tab, N);
  wyprowadz(tab, N);
  znajdz_min_max(tab, N, min,max);
  printf("\nElement minimalny  > %d",min);
  printf("\nElement maksymalny > %d\n",max);
  sortuj(tab, N);
  wyprowadz(tab, N);
}
```