Tablice - podstawowa organizacja danych

Tablice - sens wprowadzania kolejnego "typu"

Aby wyjaśnić Ci do czego służą tablice, musisz sobie najpierw przypomnieć wcześniejsze programy. W niektórych z nich chcieliśmy, aby użytkownik podał jedną liczbę za pomocą klawiatury, w innych chcieliśmy, żeby podał 2 liczby za pomocą klawiatury. Czy sprawiało nam to jakąś trudność? Nie, było to bardzo proste i łatwe.

Wyobraź sobie jednak, co by było, gdybyśmy chcieli, aby użytkownik z jakiegoś powodu musiał podać 100 lub nawet 1000 liczb. Co prawda nikt normalny aż tylu danych za pomocą klawiatury by podać nie chciał, ale my przyjmiemy, że tak właśnie jest, bowiem nie znasz na razie innych sposobów pobierania danych (z pliku).

Czy w przypadku tak dużej liczby danych byłoby Ci trudniej napisać program? Podejrzewam, że nie. Wszystko działoby się dokładnie w ten sam sposób. Problemem byłoby jednak to, jak nazywać poszczególne zmienne. Gdyby zmienne były nazywane a, b, c itd. wkrótce zabrakłoby Ci liter alfabetu. Poza tym czy w dalszej części programu łatwo byłoby Ci odgadnąć która z danych jest np. daną, która została podana jako dwudziesta?

Sprytniejsza osoba by sobie pewnie pomyślała tak - skoro mamy dużą liczbę danych, możemy je sobie nazywać na przykład tak: zmienna1, zmienna2, zmienna3, zmienna4. I rzeczywiście jest to jakieś rozwiązanie. Dzięki temu nawet w znacznie bardziej odległej części programu wiadomo by było, że np. dana pobrana jako dwudziesta z klawiatury to zmienna20.

Mimo, że sam pomysł jest dość dobry, ma on jednak duże wady. Przede wszystkim każda taka dana zostaje traktowana jako wartość pewnej zmiennej. Dane te nie są ze sobą powiązane w żaden sposób. Poza naszą wygodą nie zmienia się tak naprawdę nic - jeśli będziemy musieli pobrać bardzo dużo danych z klawiatury będziemy musieli niestety postąpić tak samo jak wtedy, gdy zmienne nazywały się a, b, c itd.

Tablice są w pewnym sensie rozwinięciem ostatniego pomysłu. Umożliwiają wygodniejsze nazewnictwo danych, a oprócz tego mają dodatkowe cechy, których nie udałoby się nam osiągnąć zwykłymi metodami. Czas zatem dowiedzieć się o tablicach nieco więcej.

Tablice - czym są i jak się ich używa

Tablice są metodą organizacji danych tego samego typu.

Co dokładnie oznacza to stwierdzenie? Oznacza ono, że w przypadku każdej tablicy musimy się zdecydować, jakiego typu będziemy przechowywać w niej dane. Musimy jasno określić, czy będziemy przechowywać w takiej tablicy liczby np. typu unsigned int lub double czy znaki, czy może napisy.

Jeśli pomyślisz, że to duże ograniczenie, jesteś w błędzie. Przede wszystkim czy rozsądnie jest pośród grupy liczb przechowywać jakąś literę? Raczej nie - nie dość, że należałoby o tym pamiętać, to tak naprawdę takie działanie mija się z celem.

Tablicę możesz sobie wyobrazić jako karton, w którym znajdują się pudełka - wszystkie takie same, wszystkie o takich samych wymiarach, a w każdym z tych pudełek znajduje się konkretna dana - w jednym na przykład znajduje się liczba 23, w drugim liczba 15. Wszystkie pudełka są jednak takie same (czyli wszystkie zmienne w tablicy są jednego typu), bowiem inaczej nie udałoby nam się

umieścić pudełek w kartonie.

Teraz najważniejsza część - w jaki sposób deklarujemy tablice. Robimy to tak:

nazwa_typu nazwa_tablicy[rozmiar];

przy czym:

nazwa_typu - nazwa przechowywanego typu danych np. double, int, char.

nazwa_tablicy - nazwa tablicy - zasady takie same jak przy nazwach zmiennych

rozmiar - ilość elementów, które chcemy przechowywać w tablicy

Jak więc widzisz utworzenie tablicy nie jest takie trudne. Bardzo charakterystyczną cechą jest to, że rozmiar tablicy (poza jednym wyjątkiem, o którym wspomnę później) musi zostać określony w momencie utworzenia tablicy.

Oto kilka przykładowych deklaracji tablic:

```
int calkowita[20];
char znaki[5];
double liczby[1000];
string napisy[5];

// tablica o nazwie calkowita - przechowuje 20 liczb typu int
// tablica o nazwie znaki - przechowuje 5 znaków
// tablica o nazwie liczby - przechowuje 1000 liczb typu double
// tablica o nazwie napisy - przechowuje 5 napisów
```

Widzisz już zapewne, że nie jest to takie trudne - zwróć jednak uwagę, że za każdym razem został podany rozmiar tablicy. Czas dowiedzieć się o tablicach czegoś więcej.

Jak używać tablic

Czas teraz dowiedzieć się, w jaki sposób odwołać się do poszczególnych danych. Załóżmy, że mamy taką taką tablicę:

```
int calkowita[7];
```

Mamy zatem tablicę siedmiu liczb całkowitych typu int (jak już wiemy int to to samo co signed int). Tablicę nazwaliśmy **calkowita**.

W jaki sposób odwołać się do poszczególnych zmiennych? Wbrew pozorom nie jest to takie trudne. Do pierwszej zmiennej odwołujemy się pisząc **calkowita[0]**, do drugiej **calkowita[1]**, a do ostatniej (czyli siódmej) odwołamy się pisząc **calkowita[6]**.

Schematycznie, aby odwołać się do danego elementu tablicy piszemy:

nazwa_tablicy [numer_elementu]

Poniższe stwierdzenie jest jednym z tych, które najlepiej jeśli przepiszesz na dużą kartkę dużymi czerwonymi literami i powiesisz obok swojego stanowiska pracy:

W języku C++ tablice indeksujemy od 0

Zauważ - pierwszy element w naszym przykładzie to nie jest calkowita[1], tylko całkowita[0]. Ostatni element tablicy to całkowita[6] a nie całkowita[7]. Zatem ostatni element ma indeks o jeden mniejszy niż liczba elementów w tablicy. U nas tablica miała 7 elementów, a ostatni, czyli siódmy element miał indeks 6.

To, że tablice są indeksowane od zera jest bardzo ważną sprawą. Wszyscy początkujący programiści popełniają w tym miejscu błąd, więc Tobie radzę sobie to dobrze zapamiętać - już tutaj uzyskasz przewagę nad innymi.

Poza tym, o czym już wspomniałem, na poszczególnych elementach tablic operujemy jak na zwykłych zmiennych. Nie ma tutaj żadnych nowości. Nowością pozostaje jedynie nieco inny zapis odwołania do zmiennej. Poza tym wszystko przebiega według dotychczas poznanych zasad.

Oto przykładowy program z wykorzystaniem tablicy:

```
#include<iostream>

using namespace std;

int main ()
{
  double calkowita[6]; // tablica 6 liczb typu double

cout <<"Podaj pierwsza liczbe: ";

cin >>calkowita[0]; // pobieramy pierwszy element do tablicy

cin.ignore();

cout <<"Podaj druga liczbe (rozna od zera): ";

cin >>calkowita[1]; // pobieramy drugi element do tablicy

 cin.ignore();

  //do pozostalych elementow tablicy przypiszemy wyniki dzialania na elementach

calkowita[2]=calkowita[0]+calkowita[1];

calkowita[3]=calkowita[0]-calkowita[1];

calkowita[4]=calkowita[0]*calkowita[1];

calkowita[5]=calkowita[0]/calkowita[1];

  //calkowita[6]=calkowita[0]+calkowita[1]; blad - calkowita[6] nie istnieje!!!
```

```
  cout <<"Podane liczby to: "<<calkowita[0]<<' '<<calkowita[1]<<'\n'
   <<"Ich suma wynosi: "<<calkowita[2]<<'\n'
   <<"Ich roznica wynosi: "<<calkowita[3]<<'\n'
   <<"Ich iloczyn wynosi: "<<calkowita[4]<<'\n'
   <<"Ich iloraz wynosi: "<<calkowita[5]<<'\n';
  cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';
  getchar();
return 0;
}
```

Jak więc widzisz, rzeczywiście na poszczególnych elementach tablicy możemy wykonywać dowolne operacje i posługujemy się nimi jak zwykłymi zmiennymi.

Przy okazji zwróć uwagę, że żądamy, aby użytkownik podał drugą liczbę taką, żeby nie była ona zerem. Czy wiesz czemu? Tak naprawdę nie ma to żadnego związku z tablicami. Chodzi jedynie o to, że później w programie dokonujemy dzielenia. Gdyby użytkownik podał zero, operacja byłaby niezdefiniowana, bo jak zapewne pamiętasz z lekcji matematyki, przez zero się nie dzieli.

W prawdziwym programie, raczej nie należy zakładać, że użytkownik nas posłucha. Do sprawdzenia, czy użytkownik nie podał czasem wartości zero, należałoby wykorzystać instrukcję warunkową if .

Dodatkowe informacje o tablicach

Wartości poszczególnych elementów tablicy możemy albo pobierać tak jak do tej pory, albo możemy przypisać wartość tych elementów w momencie powstania tablicy (jest to tak zwana inicjalizacja - dokładniej omówię to zagadnienie w jednej z następnych lekcji).

Aby przypisać wartości poszczególnym elementom tablicy należy umieścić te wartości w nawiasie klamrowym.

Jeśli wartości takich będzie dokładnie tyle ile elementów, wówczas każdy element będzie miał określoną wartość. Taki przypadek ilustruje poniższy przykład:

```
#include<iostream>

using namespace std;

int main ()
{
 int calkowite[4]={5,67,2, -3}; // kazdy element ma okreslona wartosc
cout <<calkowite[0]<<' '<<calkowite[1]<<' '<<calkowite[2]<<' '<<calkowite[3]<<'\n';
  cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';
  getchar();
return 0;
}
```

Jeśli wartości będzie mniej niż rozmiar tablicy, wówczas elementy, dla których "zabraknie" wartości, będą miały przypisaną wartość zerową (w przypadku typów liczbowych). Weź jednak pod uwagę, że w każdym innym wypadku (tzn. w przypadku braku inicjalizacji) wartości elementów tablicy będą przypadkowe. Te sytuacje ilustruje przykład:

```
#include<iostream>

using namespace std;

int main ()
{
 int calkowite[4]={5,67,2}; // ostatni element nie ma okreslonej wartosci
cout <<calkowite[0]<<' '<<calkowite[1]<<' '<<calkowite[2]<<' '<<calkowite[3]<<'\n';


  int przypadkowa[4];
  /* Nie przypisalismy wartosci. Do momentu pobrania wartosci z klawiatury lub pliku lub innych
operacji na tablicy, wartosci elementow sa PRZYPADKOWE!!!*/
cout <<przypadkowa[0]<<' '<<przypadkowa[1]<<' '<<przypadkowa[2]<<' '<<przypadkowa[3]<<'\n';


  int zerowa[4]={ };
  /*Nie przypisalismy zadnej wartosci, ale uzylismy nawiasow tak jakbysmy chcieli tego dokonac.
Wszystkie elementy beda mialy wartosc zerowa*/
cout <<zerowa[0]<<' '<<zerowa[1]<<' '<<zerowa[2]<<' '<<zerowa[3]<<'\n';


  cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';
  getchar();
return 0;
}
```

Jeśli natomiast w nawiasie klamrowym umieścimy więcej wartości niż to wynika z rozmiaru tablicy,kompilator zasygnalizuje błąd. Taka sygnalizacja błędu w przypadku tablic występuje tylko w tym wypadku. W każdym innym przypadku, kiedy utworzymy tablicę o określonym rozmiarze i będziemy się odwoływać do nieistniejącego elementu, żaden błąd nie zostanie zasygnalizowany, jednak takie działania mogą być katastrofalne w skutkach. Oto przykład:

```
#include<iostream>

using namespace std;

int main ()
```

```
{
// int calkowite[4]={5,67,2,-3,7};
/*
Powyzsza linia spowodowalaby blad - mamy o jedna wartosc za duzo w nawiasie. Kompilator
ZAPROTESTUJE
*/
  int tablica[4];
// cout <<tablica[4];
/*
To jest tez BLAD - element tablica[4] nie istnieje. Kompilator jednak NIE zaprotestuje. TRZEBA
SAMEMU UWAZAC!!!
*/
  cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';
  getchar();
return 0;
}
```

Ponadto warto dodać, że mimo że rozmiar tablicy musi zostać jawnie podany, to w przypadku gdy podajemy wartości elementów tablicy, możemy pominąć rozmiar tablicy. W takim wypadku kompilator jest w stanie sam wyliczyć rozmiar tablicy na podstawie listy wartości elementów. Ważne jednak, aby nie zapomnieć o nawiasach kwadratowych. Inaczej program się nie skompiluje. Oto przykładowy program:

```
#include<iostream>

using namespace std;

int main ()
{
  int calkowite[]={5,67,2, -3}; // 4 wartosci - czyli chcemy tablice o 4 elementach
// int tablica={5,67,2, -3}; // BLAD - brak nawiasow sugerujacych ze to tablica


  cout <<calkowite[0]<<' '<<calkowite[1]<<' '<<calkowite[2]<<' '<<calkowite[3]<<'\n';
  cout <<"Nacisnij ENTER aby zakonczyc"<<'\n';
  getchar();
return 0;
}
```

Ostrożnie z tablicami

Mimo, że tablice są bardzo przydatną metodą organizacji danych, to jednak trzeba być w posługiwaniu się nimi bardzo ostrożnym.

Wiąże się to z tym, że w C++ (i również w języku C) nie ma kontroli poprawności odwołania do tablicy. Możemy utworzyć tablicę 2 elementową i odwoływać się do jej 100-ego elementu, mimo że taki element nie istnieje.

Tego typu błędy mogą być trudne do wykrycia, tym bardziej, że gdy spróbujemy zapisać coś do tego 100-ego elementu tablicy dwuelementowej, możemy nadpisać wartość jakiejś innej zmiennej i będziemy szukać błędu nie w tym miejscu co należy.

Jeśli uważasz, że to wada języka, musisz wiedzieć, że tak naprawdę jest to zaleta. To programista musi się zatroszczyć o odpowiednie odwołania. W zamian uzyskujemy bardzo szybkie odwołania do tablicy, co jest niewątpliwie dużą zaletą.

Cechy tablic w C++

Oto najważniejsze cechy tablicy w języku C++:

- **może przechowywać elementy tylko jednego typu**
- **musi mieć określony stały rozmiar (lub rozmiar ten zostaje wyliczony przez kompilator, gdy określimy wartości poszczególnych elementów tablicy podczas inicjalizacji)**
- **jest indeksowana od zera (pierwszy element ma indeks zerowy)**
- **umożliwia szybki dostęp do dowolnego elementu tablicy**
- **nie jest sprawdzana poprawność odwołań do elementów - to na programiście spoczywa odpowiedzialność za nieprzekroczenie dopuszczalnego zakresu tablicy**
- **umożliwia łatwiejsze operowanie na danych, łatwiejsze ich wypisywanie i pobieranie**

O ile wszystkie z tych przedstawionych punktów powinny być oczywiste, ostatni taki nie jest.

O tym, jak łatwo można przeprowadzać operacje na tablicach, przekonasz się dopiero, kiedy dowiesz się w jaki sposób połączyć znajomość pętli z tablicami..