Laboratorium Informatyki II
Ćwiczenie 2

1 Tablice

Tablica to złożona struktura danych pozwalająca na przechowywanie określonej liczby wartości danego typu. Metody deklaracji i używania tablic są identyczne jak te stosowane w języku C. W tym celu należy podać typ wartości jakie będą w tablicy przechowywane, jej nazwę oraz w nawiasie kwadratowym liczbę elementów jaką będzie przechowywała. W przypadku tablicy znakowej (`char`) nalezy pamiętać o zarezerwowaniu dodatkowego miejsca dla znako końca ciagu tekstowego `'\0'`.

```
int tab1[23];
float tab2[15];
char tab3[50];
```

Tablicę tak jak każdą zmienną można inicjować podając wartości jakie zostaną do niej przypisane w momencie jej utworzenia. Przy inicjacji nie trzeba podawać jej rozmiaru, zostanie on automatycznie określony na podstawie listy przypisanych elementów. Inna metodą jest przypisanie niepełnej listy elementów przy podanym rozmiarze tablicy. W tym przypadku możliwe są dwa sposoby przypisania wartości początkowych. Podanie niepełnej listy określając początkowe wartości tablicy, lub przypisać wartości pod konkretne pozycje. Nieokreślone elementy automatycznie zostaną zapełnione zerami.

```
int tab1[] = {1,2,3,4,5,6,7}; //Tablica siedmioelementowa
int tab2[] = {[2]=3,[6]=5}; //tablica siedmioelementowa {0,0,3,0,0,0,5}
int tab3[7] = {[2]=3,[4]=5}; //tablica siedmioelementowa {0,0,3,0,5,0,0}
char tab4[] = "Ciąg testowy"; //tablica czternastoelementowa
```

char tab4[ ]=	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	C	i	ą	g		t	e	k	s	t	o	w	y	\0

W przypadku tablic dwu i więcej wymiarowych należny postępować analogicznie jak w języku C. Przy inicjacji tablicy wielowymiarowej pominąć można tylko pierwszy z wymiarów, pozostałe **muszą** być podane.

```
int tab1[3][3];
int tab2[][3] = {1,2,3,4,5,6,7,8,9};
int tab3[][3] = {{1,2,3},{4,5,6},{7,8,9}};
int tab4[3][3] = {1,2,3}};
int tab5[][3] = {{1},{4,5,6},{7,8}};
int tab6[][3] = {[0][0]=3,[2][2]=7};//niezgodny z C++
```

W języku C++ podobnie jak w C tablica (**jej nazwa**) jest wskaźnikiem na początek obszaru pamięci zarezerwowanego dla tablicy. Powoduje to że tablica gdy jest atrybutem wejściowym funkcji, jej zawartość nie jest kopiowana do tablicy lokalnej, a jest przekazywany wskaźnik na początek obszaru. Tablica nie może być także zdefiniowana jako wartość zwracana przez funkcję. Funkcja może za to zwracać wskaźnik na tablicę dynamiczną.

2 Dynamiczna alokacja pamięci

W języku C++ można dynamicznie alokowac tablice w sposób analogiczny jak w języku C za pomocą instrukcji `malloc()`, `calloc()`, `realloc()` i `free()`. Metoda ta jest ograniczona wyłącznie do typów podstawowych. W języku C++ do dynamicznej alokacji zmiennych, a w sposób szczególny przy typach złożonych jak struktury, unie i klasy należny stosować funkcje `new` i `delete`.

Podstawowym elementem języka stosowanym przy dynamicznej alokacji pamięci jest wskaźnik. Jest to specyficzna zmienna określonego typu która umożliwia przechowanie adresu do komórki pamięci w której jest przechowywana wartość danego typu. Wskaźnik deklaruje się podając typ jaki będzie wskazywał oraz jego nazwę poprzedzoną znakiem `*`. Do przypisania adresu często wykorzystuje się operator referencji `&`, który "wyciąga" adres w pamięci pod którym dana zmienna przechowuje swoją wartość. W celu "wyciągnięcia" wartości wskazywanej przez wskaźnik należy jego nazwę poprzedzić operatorem wyłuskania `*`.

```
#include <iostream>
using namespace std;
int main()
{
    int x = 25;
    int *wsk = &x;
    cout << "x=" << x << "\t*wsk=" << *wsk << "\twsk=" << wsk << endl;
    *wsk = 34;
    cout << "x=" << x << "\t*wsk=" << *wsk << "\twsk=" << wsk << endl;
}
```

Dynamiczna alokacja tablic polega na przypisaniu do wskaźnika danego typu obszaru pamięci pozwalającego przechować określoną liczbę wartości danego typu. Funkcje `calloc()` i `malloc()` realizują takie przypisanie. Funkcja `realloc()` pozwala na zmianę rozmiaru tablicy, natomiast funkcja `free()` zwalnia przydzielony do wskaźnika obszar. Zastosowanie wymienionych funkcji wymaga podpiecia do programu biblioteki `<cstdlib>`.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    int *tab;
    tab = (int*)malloc(5*sizeof(int));
    cout << "tab(" << tab << ")=";
    for(int i=0;i<5;i++)
        cout << tab[i] << ',';
    tab = (int*)realloc(tab,8*sizeof(int));
    cout << endl << "new tab(" << tab << ")=";
    for(int i=0;i<8;i++)
        cout << tab[i] << ',';
    free(tab);
}
```

Funkcje `calloc()` i `malloc()` realizują podobne zadanie, funkcja `calloc()` dodatkowo ustawia wszystkie wartości tworzonej tablicy na wartość `0`.

3 Typy deklaratywne

W języku C++ dostępne są także typy deklaratywne struktura i unia znane z języka C. Pozwalają one na zdefiniowanie własnego złożonego typu danych. Definicję typu zazwyczaj umieszcza się w obszarze nagłówkowym, dzięki temu zmienną strukturalną lub typu unia można definiować w dowolnym miejscu kodu.

3.1 Typ strukturalny

Struktura pozwala na zdefiniowanie szablonu złożonej zmiennej składającej się z zdefiniowanych w deklaracji typów. W uproszczeniu konstruuje się szablon tabeli w której określa się nazwę i typ wartości każdej z kolumn. w zmiennej utworzonej na bazie takiego szablonu można przechowywać zestaw danych odpowiadających zdefiniowanym parametrom. Na bazie zdefiniowanej struktury można utworzyć zmienną lub tablicę. Nowy typ może zostać wykorzystany przy konstruowaniu funkcji, jej atrybutów i zwracanej wartości. Przykład definicji i deklaracji struktury, zmiennej strukturalnej pokazano poniżej. Odwołanie do poszczególnych elementów zmiennej strukturalnej realizuje się poprzez podanie jej nazwy i nazwy elementu rozdzielonych kropką.

```
struct nazwa{
   int element1;
   float element2;
   char *element3;
   bool element5[10];
};

nazwa x, y[3], *z;
x.element1=34;
y[0].element1=3;
y[1].element5[3]=true;
```

3.2 Typ unia

Unia jest typem deklaratywnym o konstrukcji zbliżonej do struktury. Procedury związane z jej definiowaniem i wykorzystaniem są takie same. Różnica pomiędzy nimi sprowadza się do organizacji pamięci. W przypadku unii poszczególne jej elementy współdzielą ten sam obszar pamięci. Oznacza to że w danej chwili korzystać można tylko z jednego elementu unii. Zmiana wartości innego elementu powoduje nadpisanie współdzielonej pamięci i utratę informacji o wartości wcześniej używanego elementu. Kontrola aktualnie używanego elementu nie jest zapewniona przez unię, musi zostać zaimplementowana w kodzie przez programistę.

Zadania

a) W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać program zbierający informację o punktach nawigacyjnych wycieczki górskiej oraz wyświetlający zestawienie podsumowujące. Program powinien gromadzić następujące dane:

- Id. punktu - jego numer
- Nazwę punktu - o dowolnej długości, może składać się z kilku wyrazów
- Współrzędne punktu - x,y

- Wysokości punktu npm

Na podstawie zebranych danych program wyświetla zestawienie trasy:

- Tabelę z zestawieniem przebiegu trasy zawierającą informacje:
  (a) Numer punktu - Id.
  (b) Nazwę punktu
  (c) Odległość od poprzedniego punktu
  (d) Pokonaną różnice wysokości od poprzedniego punktu
- Długość pokonanej trasy
- Suma pokonanych przewyższeń, osobno na + i -
- Największe pokonane przewyższenie, osobno na + i -

b) Wykorzystując dynamiczna alokację pamięci napisać program generujący tablicę 2D spełniający poniższe kryteria:

- liczba wierszy określana jest przez użytkownika
- liczba elementów w każdym wierszu określana jest przez
- informacja o liczbie elementów w każdym wierszu zapisywana jest w tablicy
- w programie dostępne są funkcje:
  - wprowadzania danych do tablicy
  - wyświetlania tablicy na ekranie
  - zmiany rozmiaru tablicy (liczby wierszy i komórek w wierszu)
- program demonstruje zastosowanie zdefiniowanych funkcji