Laboratorium Informatyki II
Ćwiczenie 1

1 Wstęp

Język C++ powstał jako nadzbiór języka C. Składniowo jest on w znacznym stopniu zgodny. Język C jest językiem nastawionym na programowanie strukturalne, natomiast C++ jest językiem wspierającym programowanie obiektowe. Kurs programowania w C++ w ramach laboratorium informatyki rozpocząć należy od podstaw czyli programowania strukturalnego w C++.

Każdy program w C++ analogicznie jak w C rozpoczyna się od funkcji `main()`. Programowanie strukturalne polega na podziale zadaniowym algorytmu na pojedyncze, możliwie wąskie zadania i implementacja ich w funkcjach, które następnie są wywoływane bezpośrednio lub pośrednio z funkcji `main()`.

2 Stałe, zmienne i tablice

Zmienna stałe i tablice statyczne deklaruje się w języku C++ analogicznie jak w języku C. Do typów podstawowych dochodzi jeszcze typ logiczny `bool`. Zmienne można deklarować oraz inicjować, w sposób identyczny jak w języku C. W C++ dochodzi możliwość inicjacji za pomocą identyfikatora `auto` który identyfikuje typ przypisanej wartości i automatyczne przypisuje optymalny typ danych:

```
auto x =1;
auto y = true;
auto z = 'A';
auto w = 3.1415;
```

3 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.

4 Operacje wejścia wyjścia

Podstawową biblioteką dostarczającą narzędzi do obsługi strumieni wejścia i wyjścia jest biblioteka `<iostream>`. W języku C++ można także korzystać z wszystkich bibliotek języka C. Zgodnie z przyjętą nomenklaturą w kodzie C++ nazwy bibliotek z języka C poprzedza się literą `c` i pomija rozszerzenie `.h`: zamiast `<stdio.h>` należy zapisać `<cstdio>`.

W języku C++ za operacje wejścia i wyjścia odpowiadają obiekty obsługi strumienia, wejściowego `cin` i wyjściowego `cout` oraz operatory wstawienia określające kierunek przepływu danych `>>` i `<<`. Stosowana jest także funkcja przeniesienia kursora do nowego wiersza `endl`. W języku C++ stosowane jest pojęcie przestrzeni nazw, pozwalające na grupowanie zmiennych, funkcji, klas. Większość podstawowych narzędzi dostępna jest w standardowej przestrzeni nazw `std`. Powoduje to że wywołanie określonego polecenia możliwe jest po wcześniejszym odwołaniu się do właściwej przestrzeni nazw. Obiekty obsługi strumienia wejścia i wyjścia przynależą do standardowej przestrzeni nazw.

```
std::cout << Podaj wartość x:"
int x;
std::cin >> x;
```

Możliwe jest zadeklarowanie domyślnego stosowania określonej przestrzeni nazw lub określenia domyślnego stosowania określonego elementu z podanej przestrzeni nazw. W przypadku stosowania w kodzie rożnych przestrzeni nazw zalecane jest to drugie rozwiązanie.

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
int main()
{
cout << "Podaj wartość x:";
    int x;
    cin >> x;
    cout << "x=" << x << endl;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Podaj wartość x:";
    int x;
    cin >> x;
    cout << "x=" << x << endl;
}
```

Strumień wyjściowy zawiera szereg metod umożliwiających sterowanie sposobem wyświetlania danych na ekranie. metoda `.put` umożliwia wysłanie pojedynczego znaku na ekran poprzez przypisanie do niej argumentu w postaci znaku lub wartości kodowej znaku. Natomiast metoda `.write` umożliwia wysłanie ciągu tekstowego o określonej długości.

```
#include <iostream>
using namespace std;
int main()
{
    int tab1[] = {45,55,66,77,88};
    char tab2[] = "Przykładowy ciąg tekstowy";
    cout.put('A').put('\t').put(134).put('\n');
    cout.write(tab2,sizeof(tab2)).put('\n');
    for(int i=0;i<5;i++)
        cout.write((char*)&tab1[i],sizeof(int)).put('\t');
}
```

Obiekt strumienia wyjściowego pozwala na określenie systemu liczbowego za pomocą którego wyświetlane będą liczby całkowite. Wyboru można dokonać pomiędzy systemami dziesiętnym `dec`, ósemkowym `oct` i szesnastkowym `hex`. Określenie wybranego systemu może być realizowane na dwa sposoby: `cout << hex;` lub `hex(cout)`. Aktywowany system wyświetlania liczb całkowitych obowiązuje do końca programu lub aktywacji innego systemu.

```
#include <iostream>
using namespace std;
int main()
{
    int tab[] = {45,55,66,77,88};
    cout << "DEC\tHEX\tOCT" << endl;
    for(int i=0;i<5;i++){
        cout << dec << tab[i] << "\t";
        hex(cout);
        cout << tab[i] << "\t";
        cout << oct << tab[i] << "\n";
    }
}
```

Dla danych wyświetlanych za pomocą obiektu strumienia wyjściowego `cout` można określić minimalną liczbę znaków za pomocą których dana zmienna lub wartość będzie wy-

świetlona. Realizuje się to za pomocą metody `.width(int)`, podając jako parametr minimalną liczbę znaków. Niestety ustawienie aktywne jest tylko dla pierwszego odwołania do obiektu `cout`. W przypadku wartości rzeczywistych (typy `float` i `double`) liczbę znaków określa się za pomocą metody `.precision(int)`, o analogicznym jak metoda `.width(int)` zakresie działania.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "ułamek zwykły | ułamek dziesiętny" << endl;
    for(int i=2;i<10;i++){
        cout.width(12);
        cout << "1/" << i << " | ";
        cout.width(17);
        cout.precision(3);
        cout << 1/(float)i << endl;
    }
}
```

5 Instrukcje sterujące

Instrukcje sterujące są jednymi z najważniejszych konstrukcji w każdym języku programowania. Dzięki nim możliwe jest zaprogramowanie kodu realizującego podjęcie decyzji i wielokrotne wykonanie określonego fragmentu kodu. W języku C++ dostępne są te same instrukcje co w języku C. Ich składnia i zasady użycia są identyczne. Jedynie w przypadku pętli iteracyjnej `for()` dochodzi dodatkowa składnia realizująca iteracje po elementach tablicy. W składni wykorzystuje się identyfikator `auto`. W tak zdefiniowanej pętli zmienną iteracyjną pętli stają się po kolei elementy tablicy. W odróżnieniu od klasycznej pętli nie ma konieczności określania liczby elementów. Pętla tego typu może operować wyłącznie na tablicach statycznych.

```
#include <iostream>
using namespace std;
int main()
{
    int tab1[] = {3,8,12,36,45,77,99};
    for(auto i:tab1)
        cout << "tab1 = " << i << endl;
    char tab2[] = " Testowy ciąg znaków";
    cout << "Tekst: ";
    for(auto i:tab2)
        cout << i;
    cout << endl;
}
```

6 Programowanie funkcyjne

Funkcje są podstawową jednostką kodu pisanego w języku C/C++. Każdy program musi zawierać przynajmniej funkcję `int main()`. Funkcja zawiera wydzielony fragment kodu realizujący określone działanie. Definicja każdej funkcji poza `main()`, składa się z dwóch

części, inicjacji umieszczanej w obszarze nagłówkowym oraz kodu źródłowego zazwyczaj umieszczanego po kodzie funkcji `main()`. Inicjacja składa się z trzech elementów zakończonych średnikiem:

- deklaracji typu zwracanego przez funkcję,
- nazwy funkcji,
- listy typów atrybutów funkcji.

```
#include <iostream>
using namespace std;
  int dane(char *);
  void drukuj(char *, int);
int main()
{
    int A = dane("A");
    int B = dane("B");
    drukuj("A*B",A*B);
}
void drukuj(char *nazwa, int wartosc)
{
    cout << nazwa << " = " << wartosc << endl;
}
int dane(char *nazwa)
{
    cout << "Podaj " << nazwa << " = ";
    int x; cin >> x;
    return x;
}
```

Funkcje mogą być osadzane w kodzie na cztery sposoby:

a) w głównym pliku źródłowym (`*.cpp`) w obszarze nagłówkowym - istotna jest kolejność osadzania funkcji

b) w głównym pliku źródłowym (`*.cpp`) w obszarze nagłówkowym deklaracje funkcji, poniżej funkcji `main()` kod funkcji - kolejność osadzania funkcji nie jest istotna

c) w pliku nagłówkowym głównego pliku źródłowego (`*.hpp`) deklaracje funkcji, a kod funkcji w głównym pliku źródłowym (`*.cpp`)

d) W dedykowanym funkcji(iom) zestawie plików nagłówkowego i źródłowego (odpowiednio deklaracja i kod funkcji), w obszarze nagłówkowym lub pliku nagłówkowym głównego pliku źródłowego umieszcza się dyrektywą `#include "nazwa.hpp"` podpinającą pliki z kodem funkcji do programu

W trakcie kompilacji projektu wieloplikowego w miejscu osadzenia dyrektywy `#include` wstawiany jest kod wskazanego w niej pliku. W zależności od stosowanego środowiska programistycznego procedura tworzenia i kompilacji programu wieloplikowego będzie przebiegała odmiennie. Opcja d) spośród wymienionych powyżej wydaje się najbardziej uniwersalna. W sposób znaczący ułatwia wielokrotne wykorzystanie kodu, dzięki możliwości podpinania plików `*.hpp` i `*.cpp` z kodem funkcji do kolejnych projektów.

Ze względu na sposób osadzania kodu z pliku nagłówkowego w trakcie kompilacji należy jego treść zabezpieczyć przed wielokrotnym wykonaniem. Wynika to z tego że plik nagłówkowy ze względu na konieczność przeprowadzania weryfikacji kodu przez narzędzia wspomagające programowanie może być w projekcie osadzony kilkukrotnie. Może się to więc wiązać z wielokrotnym deklarowaniem typów, zmienny lub funkcji co prowadziło by do błędów kompilacji. Kod pliku nagłówkowego powinien być zamknięty w układzie warunkowym sprawdzającym czy dany plik został już wcześniej załadowany do projektu. Jeżeli tak to pomijany jest jego kod, jeżeli nie to jest on wykonywany. Realizuje się to dzięki dyrektywom preprocesora warunkowej i deklaracyjnej, jak pokazano poniżej. W edytorze **code::blocks** odpowiedni szablon jest automatycznie generowany.

```
#ifndef NAZWA_HPP
#define NAZWA_HPP
    kod;
#endif
```

Instrukcja z pierwszego wiersza sprawdza czy znana jest nazwa podana po spacji. Zgodnie z wytycznymi standardu języka nazwę tą zapisuje się dużymi literami, a sama nazwa jest tożsama z nazwą pliku nagłówkowego z kropką zastąpioną kreską dolną. Jeżeli nazwa jest nieznana to wykonywana jest następna linia kodu, w przeciwnym wypadku następuje przeskok do instrukcji zamykającej układ warunkowy (`#endif`).

Kolejny wiersz definiuje `NAZWA_HPP` dzięki czemu przy kolejnej próbie załadowania pliku nagłówkowego będzie już znana i kod nie zostanie wykonany. Po dyrektywie `#define` umieszcza się właściwy kod pliku nagłówkowego.

program.cpp

```
#include <iostream>
#include "input.hpp"
#include "output.hpp"
int main()
{
    int x = liczba("x");
    int y = liczba("y");
    drukuj("x+y",x+y);
    drukuj("x*y",x*y);
    drukuj("4*x-2*y",4*x-2*y);
}
```

input.hpp

```
#ifndef INPUT_HPP
#define INPUT_HPP
#include <iostream>
int liczba(const char*);
#endif
```

output.hpp

```
#ifndef OUTPUT_HPP
#define OUTPUT_HPP
#include <iostream>
void drukuj(const char*,int);
#endif
```

input.cpp

```
#include "input.hpp"
int liczba(const char *nazwa)
{
    int L;
    std::cout << "Podaj " << nazwa << ": ";
    std::cin >> L;
    return L;
}
```

output.cpp

```
#include "output.hpp"
void drukuj(const char *nazwa, int L)
{
    std::cout << "Działanie " << nazwa << " wynosi: " << L << std::endl;
}
```

Funkcja `main()` tak jak każda funkcja może mieć atrybuty wejściowe. Są one przypisywane w momencie wywołania programu w konsoli. Parametry te są umieszczane po nazwie programu za pomocą listy elementów rozdzielonych spacją. Jeżeli parametr jest ciągiem tekstowym zawierającym spacje lub tabulacje należy go zamknąć w cudzysłowach. Funkcja `main()` ma dwa parametry, jeden przechowuje informacje o liczbie parametrów a drugi w tablicy 2D typu char przechowuje poszczególne parametry. Należy pamiętać że parametrem zerowym jest nazwa programu.

```
#include <iostream>
using namespace std;
int main(int argv, char **argc) //lub *argc[]
{
    if(argv>1){
        for(int i=0;i<argv;i++)
            cout << "Parametr " << i+1 << ": --> " << argc[i] << endl;
    }else cout << "Nie podano parametrów wywołania." << endl;
}
```

7 Typy deklaratywne - struktury i unie

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;
```

7.1 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

W oparciu o informacje wykładowe i treść niniejszej instrukcji wykonać poniższe zadania. Kod programów oprzeć o zasady programowania funkcyjnego. Wydzielić kod do pliku nagłówkowego.

a) Napisać program wyznaczający pierwiastki podanego przez użytkownika równania kwadratowego

b) Napisać program obliczający średnia arytmetyczną i geometryczną liczb z zadanego przez użytkownika przedziału <A,B> liczb.

c) Napisać program obliczający sumę liczb parzystych i iloczyn liczb nieparzystych zestawu `n` liczb wygenerowanych losowo z zadanego przez użytkownika przedziału <A,B> liczb.

d) Napisać program wyświetlający na ekranie tablę przeliczania temperatury w stopniach Celsjusza w zadanym przez użytkownika przedziale na wartości odpowiadających temperatur w Kelwinach i stopniach Fahrenheita.

e) Napisać program kalkulator wykonujący wprowadzone przez użytkownika działanie matematyczne (np.: `3+3.15` lub `4^12`).