Wykład nr 1

Temat: Wprowadzenie do języków programowania oraz programowania w języku C++.

Cytaty:

Programy nie spadają z nieba,
najpierw tym niebem potrząść trzeba.

gemGreg

Każdy działający program jest przestarzały.

pierwsze prawo Murphy'ego o programowaniu

Mój cel wspaniały osiągnę na czas.

W.S. Gilbert

Wykład nr 1

Temat: Wprowadzenie do języków programowania oraz programowania w języku C++.

Zakres wykładu:

- przegląd technik programowania
- przegląd języków programowania
- ogólne informacje o programowaniu strukturalnym, obiektowym i języku C++
- biblioteka standardowa C++
- tworzenie programu
- z czego składa się program? – pierwszy program
- deklaracja vs. definicja, inicjalizacja vs. przypisanie
- słowa kluczowe języka C++
- typy danych (zmiennych, obiektów)
- operatory
- instrukcje sterujące
- podsumowanie
- ćwiczenia powtórzeniowe i sprawdzające
- następny wykład

Przegląd technik programowania

Istnieje kilka **technik** (metod, wzorców) **programowania**:

1. programowanie liniowe
2. programowanie proceduralne
3. programowanie z ukrywaniem danych
4. programowanie bazujące na obiektach
5. programowanie (z)orientowane obiektowo

Powyższe techniki programowania zostaną szczegółowo omówione na Wykładzie 6.

W pierwszej części wykładu (Wykłady 1, 2, 3) przedstawimy **programowanie strukturalne**.

W drugiej części wykładu (Wykłady 4, 5, 6) przedstawimy **programowanie obiektowe**.

Przegląd języków programowania

Języki programowania mogą być podzielone na trzy ogólne grupy:

1. języki maszynowe
2. języki asemblera
3. języki wysokiego poziomu

Język maszynowy jest „naturalnym językiem" każdego komputera. Jest on zdefiniowany przez projekt sprzętu tego komputera. Języki maszynowe składają się z łańcuchów cyfr (ostatecznie sprowadzonych do 0 i 1), które instruują komputer o przeprowadzaniu najbardziej elementarnych działań w każdej chwili. Programowanie w językach maszynowych jest wolne, nudne i niewygodne dla ludzi ze względu na maszynowy kod programu, np.
(płaca podstawowa + płaca za nadgodziny = wynik):

```
+1300042774
+1400593419
+1200274027
```

Język asemblera używa angielsko-podobnych skrótów do przedstawiania elementarnych działań. **Asemblery** to programy tłumaczące, które przekształcają programy języka asemblera do języka maszynowego. Kod języka asemblera jest zrozumiały dla ludzi i niezrozumiały dla komputera, i musi być przetłumaczony na język maszynowy.

```
LOAD        PODSTAWOWA
ADD         NADGODZINY
STORE       BRUTTO
```

Język wysokiego poziomu realizuje konkretne działania za pomocą pojedynczych instrukcji. **Kompilatory** to programy tłumaczące, które przekształcają programy w językach wysokiego poziomu na język maszynowy. Języki wysokiego poziomu umożliwiają programistom pisanie instrukcji, które wyglądają niemal jak codzienny angielski i zawierają ogólnie używane zapisy matematyczne.

```
placaBrutto = placaPodstawowa + placaNadgodziny
```

Z punktu widzenia programisty, języki wysokiego poziomu są bardziej pożądane niż języki maszynowe lub języki asemblera. Z punktu widzenia komputera, języki maszynowe i asemblera są bardziej pożądane, gdyż proces kompilacji programu w języku wysokiego poziomu na język maszynowy może zabrać znaczną ilość czasu komputera.

Spośród bardzo dużej liczby ogólnych języków programowania, jedne nastawione są na szybkość, inne na rozmiar kodu, jeszcze inne na przejrzystość itp. Najważniejsze używane obecnie języki programowania to:

1. **FORTRAN** (*FORmula TRANslator* – ang. tłumacz wzorów)
Zaprojektowany do naukowych i inżynierskich aplikacji, które wymagają złożonych obliczeń matematycznych. FORTRAN jest wciąż szeroko stosowany, szczególnie w aplikacjach inżynierskich.

2. **COBOL** (COmmon Busines Oriented Language – ang. powszechny język o orientacji ekonomicznej)
Zaprojektowany przez twórców oraz rządowych i przemysłowych użytkowników komputerów. COBOL jest wykorzystywany najczęściej do komercyjnych aplikacji, które wymagają precyzyjnej i wydajnej manipulacji dużymi ilościami danych. Obecnie więcej niż połowa całego oprogramowania ekonomicznego jest wciąż projektowana w COBOL-u.

3. **ADA** (na cześć Lady Ady Lovelace, córki poety Lorda Byrona)
Zaprojektowany pod patronatem Departamentu Obrony USA, który pragnął jednego języka, wypełniającego większość jego potrzeb, głównie do utworzenia jednego silnego systemu oprogramowania do zarządzania i kontroli Departamentu Obrony. Jedną z ważniejszych możliwości ADY jest **wielozadaniowość**, która umożliwia programistom określenie, jak wiele działań może wystąpić równolegle. Inne szeroko używane języki wysokiego poziomu (włączając C i C++), umożliwiają zasadniczo pisanie programów, które przeprowadzają tylko jedno działanie w danej chwili.

4. **Visual Basic**
Jest to następca popularnego swego czasu języka **BASIC**. Zgodnie z nazwą (*basic* znaczy prosty), był on przede wszystkim łatwy do nauki. Visual Basic pozwala na tworzenie programów dla środowiska Windows w sposób wizualny, tzn. poprzez konstruowanie okien z takich elementów jak przyciski czy pola tekstowe.
Język ten posiada dosyć spore możliwości, jednak ma również jedną, za to bardzo poważną wadę. Programy w nim napisane nie są kompilowane w całości do kodu maszynowego, ale **interpretowane** podczas działania. Z tego powodu są znacznie wolniejsze od tych kompilowanych całkowicie.

5. **Pascal**
Zaprojektowany i przeznaczony głównie do nauczania **programowania strukturalnego** w środowisku akademickim. Szybko stał się preferowanym językiem w większości uniwersytetów. Niestety brakuje w nim wielu cech potrzebnych do uczynienia go użytecznym w komercyjnych, przemysłowych i rządowych aplikacjach, więc nie został szerzej zaakceptowany poza uniwersytetami.

6. **Delphi (Object Pascal)**
Delphi wywodzi się od popularnego języka **Pascal**. Podobnie jak VB jest łatwy do nauczenia, jednakże oferuje znacznie większe możliwości zarówno jako język programowania, jak i narzędzie do tworzenia aplikacji. Jest całkowicie kompilowany, więc działa tak szybko, jak to tylko możliwe. Posiada również możliwość wizualnego konstruowania okien. Dzięki temu jest to obecnie chyba najlepsze środowisko do budowania programów użytkowych.

7. **C++**
C++ jest chyba najpopularniejszym językiem do wszelakich zastosowań. Powstało do niego bardzo wiele kompilatorów pod różne systemy operacyjne i dlatego jest uważany za najbardziej przenośny. O C++ nie mówi się zwykle, że jest łatwy – być może ze względu na dosyć skondensowaną składnię (na przykład odpowiednikiem pascalowych słów **`begin`** i **`end`** są po prostu nawiasy klamrowe { i }). To jednak dosyć powierzchowne przekonanie, a sam język jest spójny i logiczny. Jeżeli chodzi o możliwości, to w przypadku C++ są one bardzo duże – w sumie można powiedzieć, że nieco większe niż **Delphi**. Jest on też chyba najbardziej elastyczny – niejako dopasowuje się do preferencji programisty. Języki C i C++ to jedne z najlepszych i najczęściej stosowanych języków wysokiego poziomu.

8. **Java**
Ostatnimi czasy Java stała się niemal częścią kultury masowej – wystarczy choćby wspomnieć o telefonach komórkowych i przeznaczonych doń aplikacjach. Ilustruje to dobrze główny cel Javy, a mianowicie przenośność – i to nie kodu, lecz skompilowanych programów! Java jest wykorzystywana do pisania niewielkich programów umieszczanych na stronach WWW, tak zwanych **apletów**. Ceną za tą przenośność jest rzecz jasna szybkość – Javy działa znacznie wolniej niż zwykły kod maszynowy, w dodatku jest strasznie pamięciożerny. Ponieważ jednak zastosowaniem tego języka nie są wielkie i wymagające aplikacje, lecz proste programy, nie jest to aż tak wielki mankament. Java jest oparty na C i C++ (podobna składnia) i łączy cechy wielu innych języków zorientowanych obiektowo. Java zawiera rozszerzone biblioteki klas ze składnikami oprogramowania do tworzenia multimediów, sieci, wielowątkowości, grafiki, baz danych, obliczeń itp.

9. **PHP** (*Hypertext Preprocessor*)
Jest językiem używanym przede wszystkim w zastosowaniach internetowych, dokładniej na stronach WWW. Pozwala dodać im znacznie większą funkcjonalność niż ta oferowana przez zwykły HTML. Możliwości PHP są całkiem duże, nie można tego jednak powiedzieć o szybkości – jest to język interpretowany. Jednakże w przypadku głównego zastosowania PHP, czyli obsłudze serwisów internetowych, nie ma ona większego znaczenia.

Ogólne informacje o programowaniu strukturalnym, obiektowym i języku C++

Programowanie strukturalne polega na pisaniu programów, przez właściwe łączenie struktur sterujących, przy czym istnieje tylko jedne sposób wejścia/wyjścia do/z każdej struktury sterującej, a punkt wyjściowy jednej struktury jest dołączany do punktu wejściowego następnej i struktury sterujące są umieszczane jedna pod drugą w programie. Reguły programowania strukturalnego pozwalają także na zagnieżdżanie struktur sterujących.

Programowanie strukturalne tworzy programy, które są bardziej zrozumiałe niż programy niestrukturalne, łatwiejsze do sprawdzania i usuwania błędów oraz do modyfikacji.

Wszystkie programy mogą być napisane bez instrukcji skoku `goto`, a jedynie w oparciu o **trzy struktury sterujące**: ***strukturę sekwencji*** (która jest wbudowana w język C++ i polega na wykonywaniu wyrażeń jedno po drugim w porządku, w jakim zostały napisane), ***strukturę wyboru*** (`if`, `if/else`, `switch`) i ***strukturę powtórzenia*** (`while`, `do/while`, `for`). Programowanie strukturalne jest niemal równoważne z eliminacją instrukcji `goto`.

Programowanie orientowane obiektowo polega na intuicyjnym, naturalnym pisaniu programów, poprzez modelowanie obiektów świata rzeczywistego, ich atrybutów i zachowań odpowiednikami oprogramowania. Programowanie zorientowane obiektowo modeluje także komunikację pomiędzy obiektami przez wiadomości.

Obiekty mają atrybuty (dane) i wykazują zachowanie (funkcje, metody). Obiekty są elementami oprogramowania do ponownego użycia, tworzone są z „planów" zwanych klasami. Różne obiekty mogą mieć wiele takich samych atrybutów i podobne zachowania.

Język C++ jest językiem hybrydowym – umożliwia zarówno **programowanie strukturalne**, jak i **obiektowo orientowane**. To dlatego język C++ jest tak popularny.

Biblioteka standardowa C++

Programy w C++ składają się z części zwanych *klasami* i *funkcjami*. Większość programistów korzysta z bogatej kolekcji istniejących klas i funkcji w **bibliotece standardowej C++**. Nauczenie się „świata C++" składa się wtedy głównie do: 1) poznania samego języka i 2) używania klas i funkcji z biblioteki standardowej.

Biblioteka standardowa C++ zawiera bogaty zbiór funkcji do przeprowadzania powszechnych operacji matematycznych, manipulacji napisami i znakami, funkcji wejścia/wyjścia, sprawdzania błędów i wielu innych użytecznych operacji. To ułatwia pracę programisty, ponieważ funkcje te dostarczają wiele pomocnych i potrzebnych metod i narzędzi.

Funkcje i klasy biblioteki standardowej są udostępniane jako część środowiska programistycznego C++ i zazwyczaj dostarczane przez dostawców kompilatorów. Wiele bibliotek klas specjalnego przeznaczenia jest dostarczanych przez niezależnych dostawców oprogramowania.

Każda biblioteka standardowa zawiera odpowiedni **plik nagłówkowy** zawierający prototypy funkcji dla wszystkich funkcji w tej bibliotece i definicje różnych typów danych oraz stałych niezbędnych dla tych funkcji.

UWAGA: Najnowszy projekt standardu ANSI/ISO określa nowe nazwy dla wielu starszych plików nagłówkowych C++. Większość nowych nazw plików nagłówkowych nie kończy się już rozszerzeniem .h.

```
#include <iostream.h>       //plik nagłówkowy starego stylu
#include <iostream>         //plik nagłówkowy nowego stylu
```

UWAGA: Wiele kompilatorów nie obsługuje jeszcze ostatniego projektu standardu C++.

Programista może tworzyć **własne pliki nagłówkowe**, które powinny kończyć się rozszerzeniem .h. Plik nagłówkowy zdefiniowany przez programistę jest dołączany – podobnie jak pliki nagłówkowe bibliotek standardowych – przez użycie dyrektywy preprocesora `#include`, np.

```
#include ”moja_biblioteka.h”
```

Pliki nagłówkowe biblioteki standardowej „starego i nowego stylu”

„stary styl”	„nowy styl”
<assert.h>	<cassert>
<ctype.h>	<cctype>
<float.h>	<cfloat>
<limits.h>	<climits>
<math.h>	<cmath>
<stdio.h>	<cstdio>
<stdlib.h>	<cstdlib>
<string.h>	<cstring>
<time.h>	<ctime>
<iostream.h>	<iostream>
<iomanip.h>	<iomanip>
<fstream.h>	<fstream>

Funkcje matematyczne z biblioteki standardowej

W pliku nagłówkowym **<cmath>**

FUNKCJE TRYGONOMETRYCZNE:

cos(x) **sin(x)** **tan(x)** **ctan(x)** **- x** *w radianach*

FUNKCJE WYKŁADNICZE I LOGARYTMICZNE:

exp(x)	*funkcja ekspotencjalna* e^x
log(x)	*logarytm naturalny z x (podstawa e)*
log10(x)	*logarytm dziesiętny z x (podstawa 10)*

POTĘGI I PIERWIASTKI:

pow(x,y)	*potęgowanie* x^y
pow10(x)	*potęgowanie* 10^x
sqrt(x)	*pierwiastek kwadratowy*
cbrt(x)	*pierwiastek sześcienny*

INNE FUNKCJE:

hypot(a,b)	*długość przeciwprostokątnej trójkąta*
ceil(x)	*zaokrąglenie liczby x w górę*
floor(x)	*zaokrąglenie liczby x w dół*
fabs(x)	*moduł x (wartość bezwzględna x)*
fmod(x,y)	*reszta z dzielenia x/y (modulo)*

Wybrane i przydatne funkcje z biblioteki standardowej

W pliku nagłówkowym **<cstdlib>**

rand()	generacja liczb całkowitych pseudolosowych
srand()	generacja liczb całkowitych losowych

W pliku nagłówkowym **<ctime>**

time()	zwraca bieżący czas kalendarzowy w sekundach

W pliku nagłówkowym **<cstring>**

strcpy()	do kopiowania napisów
strcat()	do łączenia napisów
strcmp()	do porównywania napisów
strlen()	do określania długości napisów
size()	do określania liczby znaków w napisie

W pliku nagłówkowym **<cctype>**

islower()	do określania czy argument jest małą literą
isupper()	do określania czy argument jest wielką literą
tolower()	do zamiany wielkiej litery na małą
toupper()	do zamiany małej litery na wielką

Wskazówka praktyczna

Staraj się używać „podejścia blokowego” przy tworzeniu programów. Dzięki temu unikniesz ponownego wynalezienia koła. Korzystaj z istniejących elementów, gdzie tylko możliwe – jest to nazywane „*wielokrotnym użyciem oprogramowania*” i jest to główne założenie w programowaniu zorientowanym obiektowo.

Wskazówka praktyczna

Programy w C++ są zazwyczaj tworzone przez łączenie nowych funkcji i klas, napisanych przez programistę, z funkcjami i klasami z biblioteki standardowej C++ oraz z różnych innych niestandardowych bibliotek klas.

Wskazówka dotycząca wydajności i przenośności

Używanie funkcji i klas biblioteki standardowej zamiast pisania swoich własnych, porównywalnych wersji, może polepszyć wykonywanie programu i zwiększyć jego przenośność, gdyż są one dość uważnie pisane, aby zapewnić jak najlepszą wydajność i przenośność.

Tworzenie programu

Tworzenie programu odbywa się w dwóch etapach:

1. opracowanie kodu źródłowego
2. generowanie kodu wynikowego

Ad.1. Zapis za pomocą instrukcji języka (np. C++) kodu programu i jego edycja w pliku tekstowym.

Ad.2. Operacja ta składa się z 2 faz:

kompilacja – kompilator sprawdza czy instrukcje są zgodne z regułami języka programowania,

linkowanie – linker konsoliduje (łączy) wszystkie pliki i biblioteki tworzące program.

Program po kompilacji i linkowaniu jest w postaci wykonywalnej, tj. ma rozszerzenie **.EXE**, np. **program1.exe** i jest gotowy do uruchomienia.

Z czego składa się program? – pierwszy program

Przykład:

```
#include <iostream>              /*dyrektywa preprocesora
                                 włączająca plik nagłówkowy nowego stylu
                                 iostream do programu*/

using namespace std;             //użycie nazw z bibl. standard.

int main()
{
   cout<<"Pierwszy program"<<endl;    //wyświetlenie na ekran tekstu

   cout<<"Pierwszy program\t";        // \t - poziomy tabulator

   std::cout<<"Pierwszy program\n"<<std::endl;       // \n - nowa linia

   int a,b;                           //definicje zmiennych całkowitych

   cout<<"Podaj dwie liczby: ";
   cin>>a>>b;                         //wczytanie danych z klawiatury

   cout<<"Liczba a="<<a<<" ,liczba b="<<b<<" , a ich suma="<<a+b<<endl;

   return 0;                          //program zakończył się pomyślnie
}
```

DO ZAPAMIĘTANIA:

- Programy w C++ rozpoczynają wykonywanie od funkcji głównej programu **`main()`**
- Wszystkie zmienne w C++ muszą być zadeklarowane przed ich użyciem
- **Każda instrukcja w języku C++ MUSI kończyć się średnikiem;**
- Nazwa zmiennej jest dowolnym dozwolonym identyfikatorem, który jest serią znaków składającą się z liter, cyfr i znaków podkreślenia (_), która nie rozpoczyna się cyfrą
- C++ rozróżnia wielkość znaków (małe/wielkie litery), a więc `a5` i `A5` to różne zmienne
- Język C++ jest językiem o tzw. wolnym formacie, tzn. kod programu może się znaleźć w każdym miejscu linii, lub może być nawet rozpisany na wiele linii. Poza nielicznymi sytuacjami, w dowolnym miejscu instrukcji można przejść do nowej linii i tam kontynuować pisanie. To dlatego, że **każda instrukcja kończy się średnikiem;**

Typowy błąd programisty

Brak dołączonego pliku **<iostream>** w programie, który wprowadza dane z klawiatury lub wysyła je na ekran. Kompilator wygeneruje komunikat o błędzie.

Typowy błąd programisty

Brak średnika na końcu instrukcji jest błędem składni. Kompilator nie może rozpoznać instrukcji. Jest to tzw. **błąd kompilowania**, gdyż jest wykrywany już na etapie kompilacji, w przeciwieństwie do tzw. **błędu wykonywania**, które są trudniejsze do wykrycia.

Typowy błąd programisty

Typowym błędem wykonywania jest dzielenie przez zero, które jest ogólnie błędem krytycznym, np. może spowodować bezpośrednie wyjście z programu.
UWAGA: w niektórych systemach, dzielenie przez zero nie jest błędem krytycznym.

Typowy błąd programisty

Błędem składni jest rozdzielanie identyfikatorów przez wstawianie znaków odstępu (białych spacji) w ich nazwach, np. pisanie **ma in** zamiast **main**.

Dobry styl programisty

Pisz swoje programy w prosty i bezpośredni sposób (KIS, ang. „Keep It Simple”). Nie nadużywaj języka przez próby „dziwacznego” stosowania.

Dobry styl programisty

Każdy program powinien rozpoczynać się komentarzem opisującym swoje przeznaczenie. Ustal preferowaną przez Ciebie konwencję pisania programów, tj. rozmiary wcięć, spacji, znaki nowych linii, stosowanie komentarzy itp. Stosuj tę konwencję i zwiększ czytelność programów.

Dobry styl programisty

Twój komputer i kompilator są dobrymi nauczycielami. Eksperymentuj używając „programu testowego” i patrz co się dzieje. Czytaj uważnie ostrzeżenia kompilatora i komunikaty o błędach. Poprawiaj kod źródłowy aby eliminować uwagi kompilatora.

Dobry styl programisty

Nazywaj swoje zmienne w programie w taki sposób, aby oddawały ich przeznaczenie. Pomaga to programowi być „samodokumentującym”, przez co łatwiejsze jest zrozumienie programu przez jego samo czytanie.

Dobry styl programisty

Unikaj nazw zmiennych (identyfikatorów), które rozpoczynają się znakiem pojedynczego lub podwójnego podkreślenia, gdyż kompilator może używać takich nazw dla swoich celów.

Deklaracja vs. definicja, inicjalizacja vs. przypisanie

PLIK program.cpp

```
#include <iostream>                  //dyrektywa preprocesora
#include "naglowek.h"                //dyrektywa preprocesora
using namespace std;                 //użycie nazw z bibl. standard.

int b;                               //DEFINICJA zmiennej b (b=0)
int dodaj(int x,int y);              //DEKLARACJA (prototyp) funkcji

main()
{
   int a=2,c,d;                      //DEFINICJA zmiennych a,c,d oraz INICJALIZACJA a
   b=4;                              //PRZYPISANIE wartości 4 zmiennej b
   c=dodaj(a,b);                     //PRZYPISANIE do zmiennej c rezultatu funkcji
   d=odejmij(c,b);                   //PRZYPISANIE do zmiennej d rezultatu funkcji
   cout<<c<<" "<<d<<endl;
}
////////////////////////////////////////////////
int dodaj(int x,int y)               //DEFINICJA funkcji (nagłówek definicji)
{
   return x+y;                       //ciało funkcji
}
```

PLIK naglowek.h

```
extern int b;                        //DEKLARACJA zmiennej b
int odejmij(int x,int y)             //DEFINICJA (i jednocześnie DEKLARACJA) funkcji
{
   return x-y;
}
```

Deklaracja informuje kompilator, że dana nazwa reprezentuje zmienną (obiekt) jakiegoś typu. Można deklarować obiekt WIELE RAZY w programie.

vs.

Definicja rezerwuje (przydziela) miejsce w pamięci dla zmiennej (obiektu). Definicja jest równocześnie deklaracją, ale nie odwrotnie. Definiować obiekt w programie można TYLKO JEDEN RAZ.

Inicjalizacja to nadanie obiektowi wartości w momencie jego narodzin (podczas jego definicji). Inicjalizować obiekt można TYLKO JEDEN RAZ.

vs.

Przypisanie to podstawienie wartości do obiektu w jakimkolwiek późniejszym momencie. Przypisywać wartość do obiektu można WIELE RAZY.

Poprawna jest następująca sekwencja w programie:

```
extern int a;              //deklaracja
extern int a;              //deklaracja
int a;                           //definicja
extern int a;              //deklaracja
extern int a;              //deklaracja
```

Nie jest poprawna następująca sekwencja w programie:

```
extern int a;              //deklaracja
extern int a;              //deklaracja
int a;                           //definicja
int a;                           //definicja
extern int a;              //deklaracja
```

Słowa kluczowe języka C++

SŁOWO	OPIS
`asm`	Wstawianie kodu w asemblerze
`auto`	Klasa zmiennej lokalnej
`bool`	Typ zmiennej
`break`	Przerywa wykonywanie pętli (**for**, **while**) oraz instrukcji **switch**
`case`	Wskazuje na warunek instrukcji **switch**
`catch`	Wyłapuje wyjątek
`char`	Typ zmiennej
`class`	Deklaracje, definicje klas
`const`	Klasa zmiennych, deklaracja stałych funkcji
`const_cast`	
`continue`	Wykonanie kolejnej iteracji pętli
`default`	Wskazuje na domyślny warunek instrukcji **switch**
`delete`	Zwalnianie pamięci przydzielonej dynamicznie
`do`	Tworzenie pętli **do-while**
`double`	Typ zmiennej
`dynamic_cast`	
`else`	Alternatywa dla instrukcji **if**, gdy warunek nie jest spełniony
`enum`	Typ zmiennej
`explicit`	
`extern`	Klasa zmiennej
`false`	Wartość zmiennej **bool**
`float`	Typ zmiennej
`for`	Rodzaj pętli

SŁOWO	OPIS
`friend`	Wskazuje zaprzyjaźnioną klasę lub funkcję
`goto`	Skok bezwarunkowy
`if`	Instrukcja warunkowa
`inline`	Wstawia kod funkcji w miejscu jej wywołania
`int`	Typ zmiennej
`long`	Kwalifikator zmiennej
`mutable`	
`namespace`	Przestrzeń nazw
`new`	Przydziela pamięć dynamicznie
`operator`	Przeciążanie operatorów
`private`	Stopień ochrony danych w klasie
`protected`	Stopień ochrony danych w klasie
`public`	Stopień ochrony danych w klasie
`register`	Klasa zmiennej
`reinterpret_cast`	
`return`	Zwracanie wartości przez funkcje
`short`	Kwalifikator zmiennej
`signed`	Kwalifikator zmiennej
`sizeof`	Zwraca rozmiar obiektu (typu) w bajtach
`static`	Klasa zmiennej, funkcje statyczne
`static_cast`	
`struct`	Deklaracje, definicje struktur

SŁOWO	OPIS
`switch`	Rodzaj instrukcji warunkowej
`template`	Tworzenie wzorców
`this`	Wskaźnik dla klas
`throw`	Rzucanie wyjątkiem
`true`	Wartość zmiennej **bool**
`try`	Przechwytywanie wyjątków
`typedef`	Tworzenie nowego typu danych
`typeid`	
`typename`	
`union`	Deklaracja do definiowania unii
`unsigned`	Kwalifikator zmiennej
`using`	Wybór przestrzeni nazw
`virtual`	Deklarowanie metod wirtualnych klasy
`void`	Typ zmiennej
`volatile`	Klasa zmiennej
`wchar_t`	Typ zmiennej
`while`	Rodzaj pętli

UWAGA:
Nazwy Twoich zmiennych nie mogą być identyczne z żadnym z powyższych słów kluczowych języka C++, które są nazwami zastrzeżonymi.

W poniższym wykładzie słowa kluczowe będą oznaczane
`pogrubioną czcionką w kolorze czerwonym`, np. `int`, `float`, `void`

Typy danych (zmiennych, obiektów)

Podział typów:

- **typy fundamentalne (podstawowe)**
- **typy pochodne –** to jakby wariacje na temat typów podstawowych

- **typy wbudowane –** tj. takie, w które język C++ jest wyposażony
- **typy zdefiniowane przez użytkownika –** tj. takie, które możesz wymyślić sobie samemu

Typ wbudowany reprezentujący znaki alfanumeryczne

```
char Znak='A';   //znaki w kodzie ASCII, np. A ma kod 65
```

Typy wbudowane reprezentujące liczby całkowite

```
short Mala=5;
int Suma_Punktow;
long Duza=10e6;
```

Typy wbudowane reprezentujące liczby zmiennoprzecinkowe

```
float Srednia=14.24;
double Masa=10.4e-8;
long double Odleglosc=15.23e+24;
```

Wszystkie powyższe typy mogą być w dwóch wariantach – ze znakiem (**signed**) lub bez znaku (**unsigned**). Wyposażenie typu w znak sprawia, że może on reprezentować liczbę ujemną i dodatnią. Typ bez znaku reprezentuje liczbę dodatnią. Przez domniemanie typ występuje ze znakiem, np. `int a;` oznacza `signed int a;`

TYPY CAŁKOWITE

Nazwa	Zakres	liczba bajtów
`char, signed char`	-128 ... 127, jeden znak	1
`unsigned char`	0 ... 255, jeden znak	1
`int, signed int`	-32768 ... 32767	2
`unsigned int`	0 ... 65535	2
`long, signed long`	-2 mld ... 2 mld	4
`unsigned long`	0 ... 4 mld	4

TYPY RZECZYWISTE

Nazwa	Zakres	Liczba znaczących cyfr	liczba bajtów
`float` (pojedyńczej precyzji)	3.4 E-38 ... 3.4 E38	6	4
`double` (podwójnej precyzji)	1.7 E-308 ... 1.7 E308	15	8
`long double` (wysokiej precyzji)	3.4 E-4932 ... 1.1 E4932	18	10

Przykład:

```
cout<<sizeof(int)<<endl;  //sprawdzenie rozmiaru typu
```

Typ wyliczeniowy `enum` – to osobny typ dla liczb całkowitych. Przydaje się, gdy w obiekcie typu całkowitego chcemy przechować nie tyle liczbę, co pewien rodzaj informacji (wpisując tam kilka wartości, ale o szczególnym dla nas znaczeniu).

Przykład:

```
enum KIERUNEK1 {gora, dol, lewo, prawo};
enum KIERUNEK2 {wschod=0, zachod, polnoc=5, poludnie};
```

Definicja zmiennych typu wyliczeniowego `KIERUNEK1` oraz `KIERUNEK2`:

```
KIERUNEK1 ruch1;
KIERUNEK2 ruch2;
```

Do zmiennych `ruch1` i `ruch2` można podstawić tylko jedną z wartości na ***liście wyliczeniowej***.

Legalne są następujące operacje:

```
ruch1=gora;          ruch2=zachod;
```

Nielegalne są następujące operacje:

```
ruch1=0;             ruch2=1;
```

Na ***liście wyliczeniowej*** `KIERUNEK2` zauważamy liczby. Mimo, iż `zachod` jest reprezentowany przez liczbę 1, to tej liczby nie można podstawić do zmiennej `ruch2` typu wyliczeniowego `KIERUNEK2`. **Można podstawić tylko co jest na liście**, czyli wartość `zachod`.

Przez domniemanie ***lista wyliczeniowa*** zaczyna się od wartości 0 i dalej co 1. Programista może to wyliczanie dowolnie określić (mogą być np. dwa elementy na liście o tej samej reprezentacji liczbowej).

Typy pochodne – to jakby wariacje na temat typów podstawowych. Są to takie typy, jak na przykład *tablica* czy *wskaźnik*. Typy pochodne oznacza się stosując nazwę typu, od którego pochodzą, i operator deklaracji typu pochodnego.

Oto lista operatorów do tworzenia obiektów typów pochodnych:

[]	-	*tablica* obiektów danego typu
()	-	*funkcja* zwracająca wartość danego typu
*	-	*wskaźnik* do pokazywania na obiekty danego typu
&	-	*referencja* (przezwisko) obiektu danego typu

Typy pochodne będą omówione na kolejnych wykładach. Tutaj krótko o tych typach:

- *Tablica* – to macierz lub wektor obiektów danego typu
- *Funkcja* – to podprogram, realizujący jakieś zadanie
- *Wskaźnik* – to obiekt, zawierający adres jakiegoś innego obiektu w pamięci
- *Referencja* – to jakby przezwisko (inna, druga nazwa) jakiegoś obiektu

Typ `void` – w deklaracjach typów pochodnych może się pojawić słowo **`void`** (ang. próżny, pusty). Słowo to stoi w miejscu, gdzie normalnie stawia się nazwę typu.

Przykład:

```
void *p;        //oznacza, że p jest wskaźnikiem do pokazywania na
                  obiekt nieznanego typu

void fun();     //oznacza, że funkcja nie będzie zwracać żadnej wartości
```

Typ `bool` – projekt standardu ANSI/ISO C++ dostarcza typ danych **`bool`**, którego wartościami są **fałsz** (**`false`**) lub **prawda** (**`true`**), będące lepszą alternatywą dla starego stylu używania 0 do oznaczania fałszu i wartości niezerowej do oznaczania prawdy.

Przykład:

```
bool TN=false;
cout<<TN<<endl;         //wartosc zmiennej TN wynosi 0 (false)

TN=true;
cout<<TN<<endl;         //wartosc zmiennej TN wynosi 1 (true)
```

String (stała tekstowa) – to ciąg znaków ujęty w cudzysłów. *Stringi* przechowywane są w pamięci jako ciąg liter, a na samym jego końcu dodawany jest znak o kodzie 0 (w kodzie ASCII), czyli znak NULL.

Przykład stringu: `"to jest string";`

Przykład:

```
#include <iostream>
#include <cstring>

using namespace std;

main()
{
   string imie_nazwisko;

   cout<<"Podaj swoje imie i nazwisko: ";
   cin>>imie_nazwisko;                          <---------- Jan Kowalski
   cout<<"Witaj "<<imie_nazwisko<<endl;         ----------> Jan

   char *zdanie="to jest string";
   cout<<zdanie<<endl;                          ----------> to jest string

   cin>>zdanie;                                 <---------- Ala ma kota
   cout<<zdanie<<endl;                          ----------> problem z aplikacją
}
```

Operatory

Symbol	Nazwa	Przykład	Kojarzenie
`::`	operator zasięgu	`::nazwa_globalna;`	P do L
`[]`	element tablicy	`tablica[3]=1;`	L do P
`()`	wywołanie funkcji	`dodaj(5,3);`	L do P
`()`	nawias w wyrażeniach	`a=b*(c+d);`	L do P
`++`	post(pre)inkrementacja	`i++; (++i;)`	L do P (P do L)
`--`	post(pre)dekrementacja	`i--; (--i;)`	L do P (P do L)
`+`	jednoargumentowy plus	`+23.4;`	P do L
`-`	jednoargumentowy minus	`-34.5;`	P do L
`!`	negacja	`!prawda;`	P do L
`&`	adres zmiennej	`&zmienna;`	P do L
`*`	wskaźnik do zmiennej	`*zmienna;`	P do L
`*`	mnożenie	`a=b*c;`	L do P
`/`	dzielenie	`a=b/c;`	L do P
`%`	reszta z dzielenia	`a=b%c;`	L do P
`+`	dodawanie	`a=b+c;`	L do P
`-`	odejmowanie	`a=b-c;`	L do P

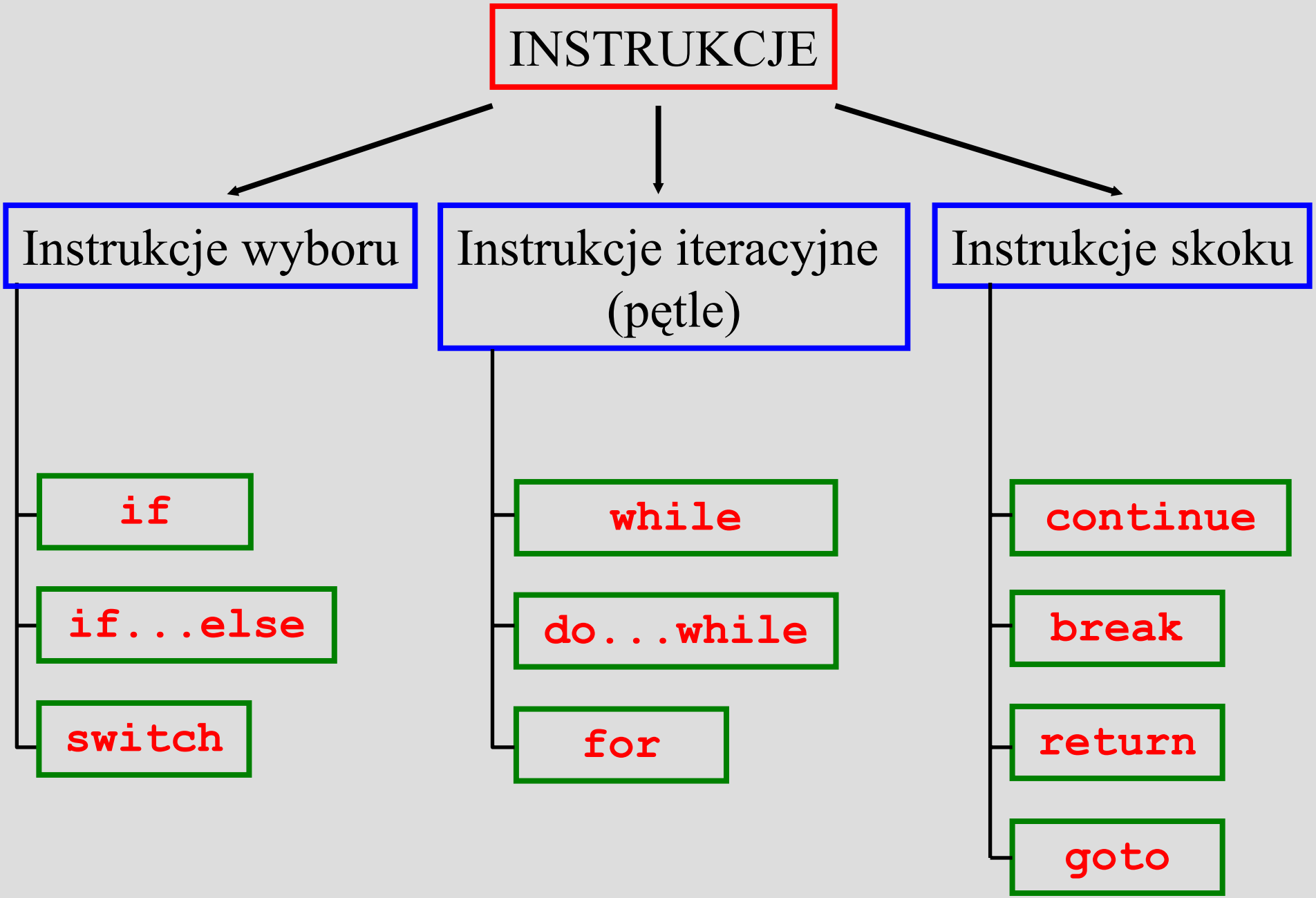
UWAGA:

Mnożenie, dzielenie, dodawanie i odejmowanie mają takie same priorytety, jak to pamiętamy ze szkoły!

Symbol	Nazwa	Przykład	Kojarzenie
<	mniejsze niż	`if (a<5);`	L do P
<=	mniejsze lub równe	`if (a<=5);`	L do P
>	większe niż	`if (a>5);`	L do P
>=	większe lub równe	`if (a>=5);`	L do P
==	równe	`if (a==5);`	L do P
!=	różne	`if (a!=5);`	L do P
&&	iloczyn logiczny	`if (a==5 && b>0);`	L do P
||	suma logiczna	`if (a==5 || b>0);`	L do P
=	przypisanie	`a=b=c;`	P do L
+=	przypisanie sumy	`a+=5;`	P do L
-=	przypisanie różnicy	`a-=5;`	P do L
*=	przypisanie iloczynu	`a*=5;`	P do L
/=	przypisanie ilorazu	`a/=5;`	P do L
,	przecinek	`int a,b;`	L do P

- Większość obliczeń jest przeprowadzana w instrukcjach przypisania
- Wyrażenia arytmetyczne są obliczane w porządku określonym przez zasady pierwszeństwa operatorów i współdziałania. Operatory mogą być jedno lub dwuargumentowe, i mogą wiązać od lewej do prawej lub od prawej do lewej.

Instrukcje sterujące


INSTRUKCJE
Instrukcje wyboru
Instrukcje iteracyjne (pętle)
Instrukcje skoku
if
if...else
switch
while
do...while
for
continue
break
return
goto

PRAWDA – FAŁSZ w języku C++

Wartość zero – odpowiada stanowi FAŁSZ
Wartość inna niż zero – odpowiada stanowi PRAWDA

Przykład 1:

```
int liczba=-5;

if (liczba)
   cout<<"prawda"<<endl;
else
   cout<<"nie prawda"<<endl;
```

Przykład 2:

```
int liczba=-5;

if (liczba==5)
   cout<<"prawda"<<endl;
else
   cout<<"nie prawda"<<endl;
```

Instrukcja warunkowa **if**

```
if(warunek)
    instrukcja;
```

```
if(warunek)
{
    instrukcja_1;
    instrukcja_2;
}
```

Przykład:

```
int liczba=5;

if (liczba)    //(liczba>0)
   cout<<”warunek prawdziwy”;
```

Przykład:

```
int liczba=-5;

if (liczba+5)    //(liczba==0)
{
   cout<<”warunek nieprawdziwy”;
   cout<<”warunek jest rowny zero”;
}
```

Instrukcja warunkowa **if/else**

```
if(warunek)
    instrukcja_1;
else
    instrukcja_2;
```

```
if(warunek_1)
{
    instrukcja_1;
    instrukcja_2;
}
else if(warunek_2)
    instrukcja_3;
else
{
    instrukcja_4;
    instrukcja_5;
}
```

Przykład:

```
int liczba;

if (liczba>0)
   cout<<"liczba jest dodatnia";
else if (liczba<0)
   cout<<"liczba jest ujemna";
else
   cout<<"liczba jest równa zero";
```

Struktura wielokrotnego wyboru **switch**

```
switch(wyrażenie)
{
   case wyrażenie_stałe_1:
   instrukcje;
   break; //nie jest konieczne

   case wyrażenie_stałe_2:
   instrukcje;
   break; //nie jest konieczne
   .
   .
   .

   default:
   instrukcje;
   break; //nie jest konieczne
}
```

Przykład:

```
char odpowiedz;
int T_zlicz=0,N_zlicz=0;

cout<<"Wprowadz odpowiedz T lub N: ";
cin>>odpowiedz;

switch (odpowiedz)
{
case 'T':
case 't':
   cout<<"odpowiedziales twierdzaco";
   ++T_zlicz;
break;    //wyjscie ze switch
case 'N':
case 'n':
   cout<<"odpowiedziales przeczaco";
   ++N_zlicz;
break;    //wyjscie ze switch
default:
   cout<<"odpowiedziales ani twierdzaco,"
       <<„ ani przeczaco";
break;    //wyjscie ze switch
}
```

Struktura wielokrotnego wyboru **switch**

```
switch(wyrażenie)
{
   case wyrażenie_stałe_1:
   instrukcje;
   break; //nie jest konieczne

   case wyrażenie_stałe_2:
   instrukcje;
   break; //nie jest konieczne
   .
   .
   .

   default:
   instrukcje;
   break; //nie jest konieczne
}
```

Przykład:

```
int liczba;
int zlicz01=0,zlicz23=0;

cout<<"Wprowadz liczbe: ";
cin>>liczba;

switch (liczba)
{
case 0:
case 1:
   cout<<"wprowadziles 0 lub 1";
   zlicz01;
break;    //wyjscie ze switch
case 2:
case 3:
   cout<<"wprowadziles 2 lub 3";
   ++N_zlicz;
break;    //wyjscie ze switch
default:
   cout<<"wprowadziles inna liczbe";
break;    //wyjscie ze switch
}
```

Instrukcja **while**

```
while(wyrazenie)
    instrukcja;
```

```
while(wyrazenie)
{
    instrukcja_1;
    instrukcja_2;
}
```

Przykład:

```
int liczba=5;

while (liczba)    //(liczba>0)
{
    cout<<"*";
    liczba=liczba-1;    //lub liczba--;
}
```

Instrukcja **do/while**

```
do
    instrukcja;
while (wyrazenie);
```

```
do
{
    instrukcja_1;
    instrukcja_2;
} while (wyrazenie);
```

Przykład:

```
int liczba=0;

do
{
    cout<<"*";
    liczba++;
} while (liczba<5);
```

Pętla **for**

```
for(instrukcja_inicjalizujaca; wyrazenie_warunkowe; krok)
{
   instrukcja_1;
   instrukcja_2;
}
```

Przykład:

```
int liczba=0;

for (int i=1; i<=4 ; i++)
   liczba=liczba+1;

cout<<"liczba: "<<liczba<<endl;
```

4

Instrukcje `continue`, `break`, `return` oraz `goto`

Instrukcja **`continue`**, gdy jest wykonywana w jednej ze struktur powtórzenia, powoduje pominięcie realizacji pozostałych wyrażeń w ciele tej pętli i kontynuuje następną iterację pętli (pętla nie zostaje przerwana).

Instrukcja **`break`**, gdy jest wykonywana w jednej ze struktur powtórzenia, powoduje bezpośrednie i natychmiastowe wyjście z tej pętli. W przypadku zagnieżdżonych pętli, intrukcja break powoduje przerwanie tylko tej pętli, w której bezpośrednio tkwi (przerwanie z wyjściem tylko o jeden poziom wyżej).

Instrukcja **`return`** powoduje zwrot wartości, stojącej po prawej stronie instrukcji, przez funkcję. Wartością tą może być dowolne wyrażenie lub wywołanie innej lub tej samej (tzw. wywołanie rekurencyjne) funkcji.

Instrukcja **`goto`** powoduje przeniesienie wykonywania programu do miejsca, gdzie jest dana etykieta, której nazwa stoi po prawej stronie instrukcji. Etykieta jest to nazwa, po której następuje dwukropek. Używanie instrukcji **`goto`** należy do „złego stylu programowania”.

Typowy błąd programisty

Zastosowanie liczb zmiennoprzecinkowych w sposób, który zakłada że są one reprezentowane precyzyjnie może prowadzić do niewłaściwych wyników. Liczby zmiennoprzecinkowe przez większość komputerów są przedstawiane tylko w przybliżeniu.

Typowy błąd programisty

Ponieważ wartości zmiennoprzecinkowe mogą być przybliżone, kontrolowanie pętli licznikowych za pomocą tych wartości może dać nieprecyzyjne wartości licznika i niedokładne testy dla zakończenia. Kontroluj pętle licznikowe wartościami całkowitymi.

Typowy błąd programisty

Próba użycia operatora inkrementacji lub dekrementacji na wyrażeniu innym niż prosta nazwa zmiennej, np. napisanie `++(x+1)` zamiast `++x`, jest błędem składni.

Typowy błąd programisty

Kiedy zmienna kontrolowana pętli **`for`** jest wstępnie definiowana w sekcji inicjalizacji nagłówka tej pętli, zmienna ta może być użyta tylko w ciele pętli **`for`**, gdyż tylko tutaj znana jest jej nazwa. Użycie tej zmiennej po ciele pętli **`for`** jest błędem składni.

Typowy błąd programisty

Dostarczenie identycznych etykiet przypadków w strukturze **`switch`** jest błędem składni.

Typowy błąd programisty

Użycie operatora == do przypisania, a operatora = do porównania jest błędem logicznym.

Dobry styl programisty

Jeżeli to możliwe, ograniczaj rozmiar nagłówka instrukcji sterujących do pojedynczej linii.

Dobry styl programisty

Nie porównuj wartości zmiennoprzecinkowych pod względem równości czy nierówności. Sprawdź raczej, czy bezwzględna wartość ich różnicy jest mniejsza od określonej małej wartości.

Dobry styl programisty

W sekcji inicjalizacji i inkrementacji pętli `for` wstawiaj tylko wyrażenia dot. liczników.

Wskazówka dotycząca wydajności

Unikaj umieszczania wewnątrz pętli wyrażeń, których wartości nie zmieniają się. Obliczenie takie wytwarza za każdym razem taki sam wynik w pętli, co jest marnotrawstwem. Wiele dzisiejszych kompilatorów umieści takie wyrażenia na zewnątrz pętli w generowanym kodzie języka maszynowego.

Wskazówka dotycząca wydajności

Niektóre kompilatory mogą znacznie szybciej kompilować programy, w których są użyte „skrócone” wersje operatorów przypisania, zwłaszcza gdy są one umieszczone w pętlach.

Wskazówka dotycząca wydajności

Mimo, iż wiele kompilatorów posiada cechy optymalizujące, które udoskonalają kod, jaki tworzysz, staraj się napisać dobry kod od początku.

Podsumowanie

PODSUMOWANIE 1:

- Języki maszynowe są maszynowo zależne i składają się głównie z łańcuchów cyfr (ostatecznie sprowadzonych do 0 i 1).
- Języki asemblera tłumaczą programy na język maszynowy i zostały uformowane przez angielsko podobne skróty.
- Języki wysokiego poziomu zawierają słowa angielskie i konwencjonalne zapisy matematyczne.
- Kompilatory tłumaczą programy w językach wysokiego poziomu na język maszynowy.
- Komputer pod kontrolą procesora wykonuje program po jednej instrukcji w danej chwili.
- Wielozadaniowość umożliwia programistom określenie równoległych działań.
- Programowanie strukturalne polega na pisaniu programów, przez właściwe łączenie struktur sterujących.
- Programowanie obiektowe polega na pisaniu programów poprzez modelowanie obiektów świata rzeczywistego, ich atrybutów i zachowań, a także komunikację pomiędzy obiektami przez wiadomości.
- Wszystkie systemy w C++ składają się z samego języka oraz bibliotek standardowych i niestandardowych. Funkcje biblioteczne nie są częścią samego języka C++; przeprowadzają wiele popularnych operacji, np. matematycznych.
- Dyrektywy preprocesora zazwyczaj wskazują pliki do dołączenia w pliku, który będzie kompilowany, i specjalne symbole do zastąpienia w tekście programu.
- Możliwe jest pisanie programów w C i C++, które są przenośne na większość komputerów. Jest wiele niezgodności między różnymi implementacjami C++ i różnymi komputerami, co czyni przenośność trudnym do osiągnięcia celem.
- C++ umożliwia programowanie zarówno strukturalne, jak i obiektowe, czemu zawdzięcza swoją popularność.
- Obiekty to elementy oprogramowania do ponownego użycia, tworzone są z „planów” zwanych klasami.

PODSUMOWANIE 2:

- Komentarze jednoliniowe rozpoczynają się znakami `//`. Komentarze ciągnące się przez wiele linii ograniczają tekst znakami `/* ... */`. Komentarze typu `/* ... */` nie mogą być w sobie zagnieżdżane.

- Linia `#include <iostream>` włącza do programu zawartość pliku nagłówkowego strumienia wejścia/wyjścia. Plik ten jest niezbędny do kompilacji programów, które używają `cin` i `cout` oraz operatorów << i >>.

- Programy w C++ rozpoczynają wykonywanie od funkcji głównej `main()`.

- Wszystkie zmienne w C++ muszą być zadeklarowane przed ich użyciem.

- Nazwa zmiennej jest dowolnym dozwolonym identyfikatorem, który jest serią znaków, składającą się z liter, cyfr i znaków podkreślenia (_), która nie rozpoczyna się cyfrą.

- Każda zmienna przechowywana w pamięci komputera ma nazwę, typ i rozmiar.

- Zawsze, gdy wartość zostaje umieszczona w komórce pamięci, zastępuje (nadpisuje) poprzednią, która jest niszczona.

- C++ rozróżnia wielkość znaków (małe/wielkie litery), a więc `a5` i `A5` to różne zmienne.

- Każda instrukcja w języku C++ MUSI kończyć się średnikiem;

- Język C++ jest językiem o tzw. wolnym formacie, tzn. kod programu może się znaleźć w każdym miejscu linii, lub może być nawet rozpisany na wiele linii. Poza nielicznymi sytuacjami, w dowolnym miejscu instrukcji można przejść do nowej linii i tam kontynuować pisanie.

- Większość obliczeń jest przeprowadzana w instrukcjach przypisania.

- Wyrażenia arytmetyczne są obliczane w porządku określonym przez zasady pierwszeństwa operatorów i współdziałania. Operatory mogą być jedno lub dwuargumentowe, i mogą wiązać od lewej do prawej lub od prawej do lewej.

PODSUMOWANIE 3:

- Pseudokod to nieformalny i naturalny język zapisu algorytmów, pomagający rozwijać programy.
- Algorytm to procedura rozwiązania problemu za pomocą odpowiednich działań w określonym porządku.
- Sterowanie programem to określanie porządku, w jakim wyrażenia (instrukcje) mają być wykonane.
- Deklaracja informuje kompilator, że dana nazwa reprezentuje obiekt jakiegoś typu.
- Definicja, jest jednocześnie deklaracją, a dodatkowo rezerwuje (przydziela) miejsce w pamięci dla zmiennej.
- Definicja jest przy okazji deklaracją, ale nie odwrotnie. Deklarować można obiekt w tekście programu wielokrotnie, ale definiować go (powoływać do życia) tylko raz.
- Inicjalizacja to nadanie obiektowi wartości w momencie jego narodzin (podczas jego definicji).
  Inicjalizować obiekt można TYLKO JEDEN RAZ.
- Przypisanie to podstawienie wartości do obiektu w jakimkolwiek późniejszym momencie.
  Przypisywać wartość do obiektu można WIELE RAZY.
- Pętle to instrukcje, którą są wykonywane wielokrotnie, aż jakiś warunek wyjścia zostanie spełniony.

Ćwiczenia powtórzeniowe

1. Napisz cztery różne instrukcje C++ dodające 1 do zmiennej całkowitej x.
   ODP: `x=x+1; x+=1; ++x; x++;`

2. Napisz instrukcje C++ wykonujące:
   a) przypisanie zmiennej `z` sumy `x` i `y` oraz zwiększenie wartości x po wykonaniu obliczenia
   b) zmniejszenie zmiennej `x` o 1, a następnie odjęcie jej od zmiennej `wynik`
   c) obliczenie reszty z dzielenia `q` przez `dzielnik` i przypisanie wyniku do `q` (na 2 różne sposoby)
   ODPa: `z=x++ +y;` **ODPb**: `wynik-=--x;` **ODPc**: `q%=dzielnik;` lub `q=q%dzielnik;`

3. Określ wartość każdej ze zmiennych po wykonaniu obliczeń, przyjmując, że na początku wykonania każdej instrukcji wszystkie zmienne mają wartość całkowitą 5:
   a) `iloczyn*=x++;` **ODP**: iloczyn=25, x=6
   b) `iloraz/=++x;` **ODP**: iloraz=0, x=6

4. Określ, czy poniższe zdania są prawdziwe, czy fałszywe:
   a) przypadek domyślny **`default`** w strukturze wyboru **`switch`** jest wymagany (**FAŁSZ**)
   b) instrukcja **`break`** jest konieczna w domyślnym przypadku struktury wybory **`switch`** (**FAŁSZ**)
   c) wyrażenie z operatorem `||` jest prawdziwe jeśli jeden lub oba operandy są prawdziwe (**PRAWDA**)
   d) wyrażenie `(x>y && a<b)` jest prawdziwe jeśli `x>y` lub `a<b` jest prawdziwe (**FAŁSZ**)

5. Napisz kod do zsumowania nieparzystych liczb całkowitych od 1 do 99 wykorzystując pętlę **`for`**:

```
int suma=0;
for(int licznik=1;licznik<=99;licznik+=2)
   suma+=licznik;          //to samo co suma=suma+licznik
```

Ćwiczenia sprawdzające

1. Napisz program, który wprowadza trzy liczby całkowite z klawiatury i wyświetla sumę, średnią, iloczyn, najmniejszą i największą z nich.

2. Napisz program, który odczytuje promień koła i wyświetla jego średnicę, obwód i powierzchnię.

3. Napisz program, który czyta liczbę całkowitą oraz określa i wyświetla, czy jest ona parzysta czy nie.

4. Napisz program, który wprowadza pięciocyfrową liczbę, dzieli ją na pojedyncze cyfry i wyświetla je, oddzielone jedna od drugiej przez trzy spacje (użyj dzielenia całkowitego i dzielenia modulo).

5. Napisz program, który oblicza kwadraty i sześciany liczb od 0 do 10 i używa znaku tabulacji `\t` do wyświetlenia wyników w postaci tabelarycznej (3 kolumny: liczba, kwadrat, sześcian).

6. Napisz program, który pobiera liczbę przejechanych kilometrów i zatankowanych litrów i oblicza średnie zużycie paliwa w litrach na 100 kilometrów.

7. Napisz program odczytujący pięciocyfrową liczbę całkowitą i określający, czy jest ona palindromem (użyj dzielenia całkowitego i dzielenia modulo do podziału liczby na poszczególne cyfry). Palindrom to liczba (lub fraza), którą można odczytać tak samo od początku, jak i od końca.

8. Napisz program odczytujący trzy niezerowe liczby całkowite i wyświetlający informację, czy mogą one być długościami boków trójkąta prostokątnego.

9. Napisz program obliczający i wyświetlający iloczyn liczb nieparzystych od 1 do 15.

10. Napisz program obliczający silnię liczb od 1 do 10 i wyświetlający wyniki w formie tabeli.

Następny wykład

Wykład nr 2

Temat: Tablice statyczne, funkcje, klasy pamięci, reguły zasięgu, obszary nazw.