Maciej Sobieraj

Wykład 1

Outline

1. **Wprowadzenie do programowania**

Pierwszy program

- Określenie oczekiwań. Jaki rezultat chcemy otrzymać.
- Wyświetlenie krótkiego tekstu na ekranie. Załóżmy, że tekst będzie następujący:

  „It's me, your first program.”
- Jakie następne kroki musimy podjąć? Spróbujmy je wymienić:
  - Uruchomienie programu
  - Wyświetlenie tekstu
  - Zakończenie działania
- Ten rodzaj struktury, pół formalny opis każdego kroku programu nazywany jest **algorytmem**.

Pierwszy program

- # (hash) na początku pierwszej linki określa zawartość określaną mianem dyrektywy preprocesora (**preprocessor directive)**

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "It's me, your first program.";
        return 0;
}
```

- Oddzielna część kompilatora, której zadaniem jest wstępne wczytanie tekstu do programu i wykonanie pewnych modyfikacji.

Pierwszy program

- Zmiany, które wprowadzi preprocesor, są w pełni kontrolowane przez jego dyrektywy.
- Mamy do czynienia z dyrektywą **include**.
- Kiedy preprocesor wykonuje tę dyrektywę, zastępuje dyrektywę zawartością pliku, którego nazwa znajduje się w tej dyrektywie.

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "It's me, your first program.";
        return 0;
}
```

Pierwszy program

- Pisanie programu jest podobne do budowania konstrukcji z gotowymi blokami.
- W naszym programie użyjemy jednego takiego bloku, który zostanie wykorzystany gdy chcemy napisać coś na ekranie.
- Ten blok nazywa się **cout**, ale kompilator nic o nim nie wie.

Pierwszy program

- Zestaw wstępnych informacji, których potrzebuje kompilator, znajduje się w plikach nagłówkowych (**header files**).
- Pliki te zawierają zbiór wstępnych informacji o gotowych blokach, które mogą być używane przez program do wyświetlenia tekstu na ekranie lub do wczytania znaków z klawiatury.
- Kiedy więc nasz program zamierza coś wyswietlić, użyje bloku zwanego **cout**.

Pierwszy program

- W języku C ++ wszystkie elementy standardowej biblioteki C ++ deklarowane są w przestrzeni nazw (**namespace)** o nazwie std.
  Przestrzeń nazw jest abstrakcyjnym kontenerem lub środowiskiem stworzonym do przechowywania, logicznego grupowania unikatowych elementów (bloków).

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "It's me, your first program.";
        return 0;
}
```

Pierwszy program

- Standard języka C ++ zakłada, że spośród wielu różnych bloków, które mogą być wprowadzone do programu, zawsze musi być obecny jeden określony blok, w przeciwnym razie program nie będzie poprawny.
- Ten blok jest zawsze funkcją o tej samej nazwie: **main**.

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "It's me, your first program.";
        return 0;
}
```

Pierwszy program

- Wewnątrz głównego obiektu funkcjonalnego znajdujemy odnośnik do bloku o nazwie **cout**.
- Każda instrukcja w C ++ musi kończyć się średnikiem.
- To konkretne stwierdzenie mówi: poinstruuj podmiot o nazwie **cout**, aby wyświetlał następujący tekst na ekranie (jak wskazuje <<, który określa kierunek, w którym wysyłany jest tekst).

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "It's me, your first program.";
        return 0;
}
```

Pierwszy program

- Polecenie **return** używane jest w funkcji, powoduje zakończenie wykonywania funkcji.
- Jeśli wykonasz **return** w dowolnym miejscu wewnątrz funkcji, funkcja ta natychmiast przerywa wykonywanie.

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "It's me, your first program.";
        return 0;
}
```

Liczby

- Liczby obsługiwane przez współczesne komputery są dwojakiego rodzaju:
  - liczby całkowite (integers), pozbawione części ułamkowej,
  - liczby zmiennoprzecinkowe (floating-point numbers), które zawierają (lub są w stanie pomieścić) część ułamkową.
- Charakterystyka liczby określająca jej rodzaj, zakres i zastosowanie nazywa się typem (**type**).

Zmienne

- Do przechowywania liczb lub wyników operacji arytmetycznych używamy specjalnych "kontenerów" zwanych zmiennymi.
- Każda zmienna ma :
  - nazwę
  - typ
  - wartość

Zmienne

- Jeśli chcesz nadać nazwę zmiennej, musisz przestrzegać kilku ścisłych zasad:
  - nazwa zmiennej może składać się z wielkich i małych liter alfabetu łacińskiego, cyfr i znaku _ (podkreślenie),
  - nazwa zmiennej musi zaczynać się od litery,
  - znak podkreślenia to litera (dziwne, ale prawdziwe),
  - wielkie i małe litery traktowane są jako różne (trochę inaczej niż w świecie rzeczywistym - Alice i ALICE to te same imiona, ale są to dwie różne nazwy zmiennych, a co za tym idzie dwie różne zmienne).

Zmienne

- **Typ** jest atrybutem, który jednoznacznie określa, które wartości mogą być przechowywane wewnątrz zmiennej.
- Spotkaliśmy już typy całkowite (**int**) i zmiennoprzecinkowe (**float**).
- **Wartość** zmiennej jest tym, co do niej włożyliśmy.
- Możemy wprowadzić tylko wartość zgodną z typem zmiennej.
- Zmienna tworzona jest w wyniku **deklaracji**.
- **Deklaracja** jest strukturą składniową, która wiąże nazwę podaną przez programistę ze specjalnym **typem** oferowanym przez język C ++.

Zmienne

- **Deklaracja** zmiennej typu int o nazwie Counter.

```
int Counter;
```

- **Operator przypisania (assignment operator)** ma prostą składnię i jednoznaczną interpretację.

```
Counter = 1;
```

- Kolejny przykład:

```
Result = 100 + 200;
```

Zmienne

- W języku C ++ znak = nie oznacza, że **jest równy**, ale **przypisuje wartość**.
- Wartość zmiennej x jest zwiększana o 1, nie ma to nic wspólnego z porównywaniem zmiennej z jakąkolwiek wartością.

```
x = x + 1;
```

Zmienne

- Lista słów, które odgrywają szczególną rolę w każdym programie napisanym w języku C ++

and	do	new	switch
and_eq	double	not	template
asm	dynamic_cast	not_eq	this
auto	else	operator	throw
bitand	enum	or	true
bitor	explicit	or_eq	try
bool	export	private	typedef
break	extern	protected	typeid
case	false	public	typename
catch	float	register	union
char	for	reinterpret_cast	unsigned
class	friend	return	using
compl	goto	short	virtual
const	if	signed	void
const_cast	inline	sizeof	volatile
continue	int	static	wchar_t
default	long mutable	static_cast	while
delete	namespace	struct	xor
			xor_eq

Komentarze

- Komentarz wierszowy (line comment) odrzuca wszystko, od miejsca, w którym znaleziono parę znaków ukośnych (//) do końca tej samej linii.

```
int Counter; // counts the number of sheep in the meadow
```

- W języku C ++ komentarz blokowy jest tekstem rozpoczynającym się od pary następujących znaków:

```
/* Counter variable counts the number of sheep in the meadow */
int Counter;
```

Liczby zmiennoprzecinkowe

- **Uwaga:** separator dziesiętny jest niezbędny do rozpoznawania liczb zmiennoprzecinkowych w C ++. Spójrz na te dwie liczby:
  - 4
  - 4.0
- Możesz myśleć, że są dokładnie takie same, ale kompilator C ++ widzi je zupełnie inaczej:
  - 4 jest typu int.
  - 4.0 jest typu float.

Liczby zmiennoprzecinkowe

- Bardzo ważna różnica między tymi dwoma typami danych

```
int i;
float x;

i = 10 / 4;
x = 10.0 / 4.0;
```

- Po zmianie z **int** na **float**, wartość zmiennej **f** wynosi 100.0, ponieważ wartość typu **int** (100) jest automatycznie konwertowana na zmienną (100.0).

```
int i;
float f;

i = 100;
f = i;
```

Operatory

- **Operator przypisania** znak =
- * **operator mnożenia**

```
int i,j,k;
float x,y,z;

i = 10;
j = 12;
k = i * j;
x = 1.25;
y = 0.5;
z = x * y;
```

Operatory

- / operator dzielenia

```
int i,j,k;
float x,y,z;

i = 10; j = 5;
k = i / j;
x = 1.0; y = 2.0;
z = x / y;
```

- Dzielenie przez zero

```
float x;

x = 1.0 / 0.0;
```

```
float x,y;

x = 0.0;
y = 1.0 / x;
```

Operatory

- **Operator dodawania** + (plus)

```
int i,j,k;
float x,y,z;

i = 100; j = 2;
k = i + j;
x = 1.0; y = 0.02;
z = x + y;
```

- **Operator odejmowania** – (minus)

```
int i,j,k;
float x,y,z;

i = 100; j = 200;
k = i - j;
x = 1.0; y = 1.0;
z = x - y;
```

Operatory

- **Operator modulo** jest dość osobliwy, ponieważ nie ma odpowiednika wśród tradycyjnych operatorów arytmetycznych.
- Jego graficzna reprezentacja w języku C++ to % (procent). Jest operatorem binarnym (wykonuje operację modulo)

```
int i,j,k;

i = 13;
j = 5;
k = i % j;
```

Operatory

- Operatory w kolejności od najwyższego do najniższego priorytetu.

+ -	unary
* / %	
+ -	binary

Operatory

- Oba operatory (* i %) mają ten sam priorytet.

2 * 3 % 5

- Podwyrażenia w nawiasach są zawsze obliczane jako pierwsze

```
int i,j,k,l;
i = 100;
j = 25;
k = 13;
l = (5 * ((j % k) + i) / (2 * k)) / 2;
```

Operatory

- Operator użyty do zwiększenia wartości zmiennej o jeden
  - **int** SheepCounter;
  - SheepCounter = 0;
  - SheepCounter++;
- Zmniejszenie wartości o jeden
  - DaysUntilHoliday--;

Operatory

- Operacja: ++Variable --Variable
  - Efekt: Zwiększ / zmniejsz zmienną o 1 i użyj jej wartości już zwiększonej / zredukowanej.
- Operacja: Variable++ Variable--
  - Efekt: Użyj wartości oryginalnej (niezmienionej), a następnie zwiększ lub zmniejsz wartość zmiennej o 1.

Variable++	post-increment operator
++Variable	pre-increment operator
Variable--	post-decrement operator
--Variable	pre-decrement operator

Operatory

- Wynik?

```
int i,j;

i = 4;
j = 2 * i++;
i = 2 * --j;
```

Operatory

- Operatory skrótów

```
i *= 2;
SheepCounter += 10;
```

```
SheepCounter = SheepCounter + 10;
i = i * 2;
```

Typ char

- char, skrót od słowa “character”.

```
char Character;
```

- Komputery przechowują znaki jako liczby.

Character	Dec	Hex		Character	Dec	Hex
@	64	40		`	96	60
A	65	41		a	97	61
B	66	42		b	98	62
C	67	43		c	99	63
D	68	44		d	100	64

Typ char

- Ujęte w pojedyncze cudzysłowy (apostrofy)

```
Character = 'A';
```

- Przypisz nieujemną liczbę całkowitą, która jest kodem wymaganego znaku

```
Character = 65;
```

Typ char

- Język C ++ wykorzystuje specjalną konwencję, która obejmuje także inne postacie, nie tylko apostrofy.
- Znak \ (zwany odwrotnym ukośnikiem) działa jako tak zwany znak **escape**, ponieważ używając znaku \ odchodzimy od normalnego znaczenia znaku następującego po \

```
Character = '\';
```

Typ char

- **\n** – oznacza przejście do nowej linii i czasami nazywany jest LF (Line Feed)

\n

- \r – oznacza powrót do początku linii i czasami nazywany jest CR (Carriage Return)

\r

Typ char

- **\a** (alarm) jest reliktem przeszłości, gdy często używano telegrafów do komunikowania się z komputerami; wysłanie znaku włącza dzwonek, stąd znak jest oficjalnie nazywany BEL (jako dzwonek);

```
\a
```

Typ char

- W języku C ++ istnieje założenie, które może wydawać się zaskakujące: typ **char** jest traktowany jako szczególny rodzaj typu **int**.

```
char Char;

Char = 'A';
Char += 32;
Char -= ' ';
```

```
Char = 'A' + 32;
Char = 'A' + ' ' ;
Char = 65 + ' ';
Char = 97 - ' ';
Char = 'a' - 32;
Char = 'a' - ' ';
```

Pytanie: czy **x** jest równy **y**?

- Tutaj mamy innego programistę, który osobno liczy czarne i białe owce i może zasnąć tylko wtedy, gdy jest dokładnie dwa razy więcej czarnych owiec niż białych

```
BlackSheepCounter == 2 * WhiteSheepCounter
```

Pytanie: czy **x** jest różny od **y**?

- Aby zadać to pytanie, używamy **!=** (exclamation equal).

```
DaysUntilTheEndOfTheWorld != 0
```

Pytanie: czy **x** jest większe od **y**?

- Można zadać to pytanie używając operatora **>** (greater).

```
BlackSheep > WhiteSheep
```

Pytanie: czy **x** jest większe bądź równe **y**?

- Operator “większe” operator ma inny specjalny, nieostry wariant, ale jest oznaczony inaczej niż w klasycznej notacji arytmetycznej: **>=** (greater equal).

CentigradesOutside >= 0.0

Pytanie: czy **x** jest mniejsze bądź równe **y**?

- Operatory, które używamy w tym przypadku to operator **<** (mniejszy) oraz operator nieostry **<=** (mniejszy równy).

```
CurrentVelocity < 110
CurrentVelocity <= 110
```

Jak możemy użyć odpowiedź, którą otrzymaliśmy?

- Co możemy zrobić z odpowiedzią, którą dał nam komputer? Mamy co najmniej dwie opcje: po pierwsze, możemy ją zapamiętać (zapisać w zmiennej) i wykorzystać później. Jak to zrobimy?

```
int Answer, Value1, Value2;

Answer = Value1 >= Value2;
```

Instrukcje warunkowe

- Mechanizm umożliwiający nam zrobienie czegoś, jeśli spełniony jest warunek.
- Język C ++ oferuje nam specjalne instrukcje. Ze względu na swój charakter i zastosowanie, nazywa się to **instrukcją warunkową**.

```
if(true_or_not) do_this_if_true;
```

```
if(TheWeatherIsGood) GoForAWalk();
HaveLunch();
```

Instrukcje warunkowe

- Mechanizm umożliwiający nam zrobienie czegoś, jeśli spełniony jest warunek.
- Język C ++ oferuje nam specjalne instrukcje. Ze względu na swój charakter i zastosowanie, nazywa się to **instrukcją warunkową**.

```
if(SheepCounter >= 120) SleepAndDream();
```

Instrukcje warunkowe

- Kiedy musimy wykonać warunkowo więcej niż jedną instrukcję, musimy użyć nawiasów klamrowych **{** i **}**, które tworzą strukturę zwaną **strukturą złożoną** lub (znacznie prościej) **blokiem**. Kompilator traktuje blok jako pojedynczą instrukcję.

```
if(SheepCounter >= 120) {MakeABed(); TakeAShower();
SleepAndDream(); } FeedTheSheepdogs();
```

Instrukcje warunkowe

- Kiedy musimy wykonać warunkowo więcej niż jedną instrukcję, musimy użyć nawiasów klamrowych **{** i **}**, które tworzą strukturę zwaną **strukturą złożoną** lub (znacznie prościej) **blokiem**. Kompilator traktuje blok jako pojedynczą instrukcję.

```
if(SheepCounter >= 120){
        MakeABed();
        TakeAShower();
        SleepAndDream();
}
FeedTheSheepdogs();
```

Wyjście

- **cout** jest jednym ze strumieni i jest gotowy do pracy bez specjalnych przygotowań - potrzebuje tylko nazwy pliku nagłówkowego.
- Zarówno operator **<<** jak i strumień **cout** odpowiadają za dwa ważne działania:
  - **przekształcenie** wewnętrznej (maszynowej) reprezentacji wartości całkowitej w formę akceptowalną dla ludzi
    przeniesienie skonwertowanej postaci na urządzenie wyjściowe, np. konsola

```
int herd_size = 110;
cout << herd_size;
```

Wyjście

- Możesz także połączyć więcej niż jeden operator **<<** w jednym ciągu **cout**, a każdy z wypisywanych elementów może być innego typu.

```
int herd_size = 123;
cout << "Sheep counted so far: " << herd_size;

int square_side = 12;
cout << "The square perimeter is: " << 4 * square_side;
```

Wyjście

- Jeśli chcesz, aby wartość typu **int** była przedstawiana jako stała liczba heksadecymalna, musisz użyć tak zwanego **manipulatora**.
- Manipulator zaprojektowany do przełączania strumienia w tryb szesnastkowy nazywany jest **hex**.

```
int byte = 255;
cout << "Byte in hex: " << hex << byte;

int byte = 255;
cout << hex << byte;
cout << byte << dec << byte;
```

Wyjście

- Manipulator **oct** przełącza strumień na tryb ósemkowy.

```
int byte = 255;
    cout << oct << byte;
```

Wyjście

- Trzy manipulatory, które wcześniej pokazywaliśmy, są tylko jedną z metod (prawdopodobnie najprostszych) dostępu do właściwości basefield. Ten sam efekt można osiągnąć za pomocą manipulatora **setbase**, który bezpośrednio instruuje strumień o wartości bazowej, której powinien użyć podczas konwersji.
- Wymaga pliku nagłówkowego o nazwie **iomanip**

```
#include <iostream>
#include <iomanip>

using namespace std;
int main(void)
{
	int byte = 255;
	cout << setbase(16) << byte;
	return 0;
}
```

Wyjście

- Ogólnie, strumienie wyjściowe (w tym **cout**) są w stanie **rozpoznać rodzaj wypisywanej wartości** i odpowiednio działać, tzn. będą używać odpowiedniej formy prezentacji danych dla wartości **char** i **float**.

```
char Char = 'X', Minus = '-';
float Float = 2.5;
cout << Char << Minus << Float;
```

Wyjście

- **cout** jest w stanie rozpoznać **rzeczywisty typ swojego elementu**, nawet jeśli jest to efekt konwersji.
- Kod ASCII znaku X to 88

```
char Char = 'X';
 int Int = Char;
 cout<<Char<<" "<<(int)Char<<" "<<Int<<" "<<(char)Int;
```

Wyjście

- Czasami możemy chcieć (a czasami musimy) **złamać** linię przesyłaną na ekran.

```
cout << "1\n2" << endl << "3\n";
```

Wyjście

- Strumienie wyjściowe próbują wyprowadzać wartości zmiennoprzecinkowe w formie, która jest bardziej zwarta i decyzja jest podejmowana dla każdej wydrukowanej wartości zmiennoprzecinkowej.
- Na przykład poniższy fragment:
  - float x = 2.5, y = 0.0000000025;
  - cout << x << endl << y << endl;

- Na ekranie wyświetli następujące dane wyjściowe:
  - 2.5
  - 2.5e-009

Wyjście

```
float x = 2.5, y = 0.0000000025;
cout << fixed << x << " " << y << endl;
cout << scientific << x << " " << y << endl;
```

- Program wyświetli następujący tekst:
  - 2.500000 0.000000
  - 2.500000e+000 2.500000e-009

Wejście

- Najprostszym sposobem jest mentalna zmiana kierunku transferu i potwierdzenie tego dla wprowadzania danych:
  - Używamy strumienia **cin** zamiast **cout**
  - Używamy operatora **>>** zamiast **<<.**
- Nawiasem mówiąc, operator >> jest często nazywany **operatorem ekstrakcji**.
- Strumień **cin**, wraz z operatorem ekstrakcji, jest odpowiedzialny za :
  - przenoszenie czytelnej dla człowieka postaci danych z urządzenia wejściowego, np. konsola
  - przekształcenie danych w wewnętrzną (maszynową) reprezentację wprowadzanej wartości.

Wejście

- Użytkownik wprowadza wartość z klawiatury, a program zapisuje ją w określonej zmiennej (*MaxSheep*).

```
cin >> MaxSheep;
```

Wejście

```
#include <iostream>

using namespace std;

int main(void)
{
	int value,square;

	cout << "Give me a number and I will square it!\n";
	cin >> value;
	square = value * value;
	cout << "You've given " << value << endl;
	cout << "The squared value is " << square << endl;
	return 0;
}
```

Wejście

```
#include <iostream>
#include <cmath>

using namespace std;

int main(void) {
	float value,squareroot;

	cout << "Give me a number and I will find its square root:" << endl;
	cin >> value;
	if(value >= 0.0) {
		squareroot = sqrtf(value);
		cout << "You have given: " << value << endl;
		cout << "The square root is: " << squareroot << endl;
	}
	return 0;
}
```

Outline

1. **Kontrola przepływu i typy danych**

Instrukcje warunkowe

- Język "C ++" pozwala nam wyrazić alternatywne plany.
- Tak więc wykonanieinstrukcji **if-else** wygląda następująco :
  - **jeśli** warunek jest "prawdziwy" (jego wartość nie jest równa zero), wykonywana jest instrukcja *perform_if_condition_true* i kończy się instrukcja warunkowa;
  - **jeśli** warunek jest "fałsz" (jest równy zeru), wykonywana jest *perform_if_condition_false*, i kończy się instrukcja warunkowa

```
if (true_or_false_condition)
    perform_if_condition_true;
else
    perform_if_condition_false;
```

```
if(TheWeatherIsGood)
    GoForAWalk();
else
    GoToATheatre();
HaveLunch();
```

Instrukcje warunkowe

- Podobnie jak inne, prostsze instrukcje, które już napotkaliśmy, zarówno **if**, jak i **else**, mogą zawierać tylko **jedną instrukcję**.
- Jeśli chcesz dodać więcej niż jedną instrukcję, musisz użyć bloku

```
if(TheWeatherIsGood) {
        GoForAWalk();
        HaveFun();
}
else {
        GoToATheatre();
        EnjoyTheMovie();
}
HaveLunch();
```

Instrukcje warunkowe

- Pomyśl o tym, kiedy instrukcja umieszczona po **if** jest kolejną instrukcją **if**.
- Pamiętaj, że każdy **else** odnosi się do najbliższej poprzedniej instrukcji **if,** która nie pasuje do żadnego innego **else**

```
if(TheWeatherIsGood)
    if(NiceRestaurantFound)
        HaveLunch();
    else
        EatASandwich();
else
    if(TicketsAvailable)
        GoToATheatre();
    else
        GoShopping();
```

Instrukcje warunkowe

- Struktura złożona z kilku instrukcji **if**, nazywa się kaskadą.

```
if(TheWeatherIsGood)
   GoForAWalk();
else if(TicketsAvailable)
   GoToATheatre();
else if(TableAvailable)
   GoForALunch();
else
   PlayChessAtHome();
```

Nie tylko **int** jest **int**

- Aby określić nasze wymagania dotyczące pamięci, możemy użyć dodatkowych słów kluczowych (modyfikatorów):
  - **long** – używany do zadeklarowania, że potrzebujemy szerszego zakresu **int** niż standardowy;
  - **short** – używany do zadeklarowania, że potrzebujemy węższego zakresu **int** niż standardowy;
  - **unsigned** – używane do deklarowania, że zmienna będzie używana tylko dla liczb nieujemnych;

Nie tylko **int** jest **int**

- Zmienna **Counter** użyje mniejszej liczby bitów niż standardowy **int** (np. może mieć długość 16 bitów - w tym przypadku zakres zmiennej zostanie zawężony do zakresu [-32768 do 32767]).
- Słowo **int** może zostać pominięte, ponieważ wszystkie **deklaracje bez nazwy typu są uważane za domyślnie określające int**

```
short int Counter;
```

Nie tylko **int** jest **int**

- Zmienna **Ants** użyje więcej bitów niż standardowy **int** (na przykład 64 bity, więc może być używana do przechowywania liczb z zakresu od [-9223372036854775808 do 9223372036854775807]

```
long int Ants;
```

Nie tylko **int** jest **int**

- Jeśli uznamy, że zmienna nigdy nie będzie wartością ujemną, możemy użyć modyfikatora *unsigned*

```
unsigned int Positive;
```

- Możemy również łączyć ze sobą niektóre modyfikatory

```
unsigned long int HugeNumber;
```

Nie tylko **int** jest **int**

- Modyfikatory *short* i *long* nie mogą być używane z **float**, ale istnieje typ o nazwie **double**.
- Dane przechowywane w zmiennej zmiennoprzecinkowej mają **skończoną precyzję** - innymi słowy, tylko pewna liczba cyfr jest dokładnie zapisana w zmiennej

```
11111111111111111.11111111111111111111
111111113185165312 0.000000
```

- Można powiedzieć, że zmienna zapisuje (tylko) 8 cyfr. Jest to oczekiwana dokładność 32-bitowych zmiennych typu **float**. Używanie typu **double** (zwykle 64-bitowe) gwarantuje, że zmienna zapisze więcej cyfr - około 15 do 17.

Zmienne float

- Jeśli bardzo mała wartość zmiennoprzecinkowa zostanie dodana do bardzo dużej, może skończyć się to niespodzianką.

11111111000.0 + 0.00011111111

- Jeśli dodamy te dwie zmienne, prawdopodobnie dostaniemy
  - 11111110656.000000

Pamięci George'a Boole'a

- Zmienne tego typu mogą przechowywać tylko dwie różne wartości : **true** i **false**.

```
bool developer_is_hungry = false;
```

Prosty program

```
/* finding the larger of two numbers */
#include <iostream>
using namespace std;

int main(void) {
   /* the two numbers */
   int number1,number2;

   /* we will save the larger number here */
   int max;

   /* read two numbers */
   cin >> number1;
   cin >> number2;

   /* we temporarily assume that the former number is the larger one */
   /* we will check it soon */
   max = number1;

   /* we check if the assumption was false */
   if(number2 > max)
      max = number2;

    /* we print the result */
   cout << "The larger number is " << max << endl;

   /* we finish the program successfully */
   return 0;

}
```

Prosty program

```
/* finding the largest of three numbers */
#include <iostream>
using namespace std;

int main(void) {
   /* the three numbers */
   int number1,number2, number3;

   /* we will save the larger number here */
   int max;

   /* read three numbers */
   cin >> number1;
   cin >> number2;
   cin >> number3;

   /* we temporarily assume that the former number is the larger one */
   /* we will check it soon */
   max = number1;

   /* we check if the second value is the largest */
   if(number2 > max)
   max = number2;

   /* we check if the third value is the largest */
   if(number3 > max)
   max = number3;

   /* we print the result */
   cout << "The largest number is " << max << endl;

   /* we finish the program successfully */
   return 0;

}
```

Prosty program

- Zignorujmy teraz język "C ++" i spróbujmy analizować problem, nie myśląc o programowaniu. Innymi słowy, spróbujmy napisać **algorytm**, a kiedy będziemy z niego zadowoleni, spróbujemy go zaimplementować.
- Zamierzamy użyć jakiejś notacji, która w ogóle nie jest językiem programowania, ale jest sformalizowana, zwięzła i czytelna. Nazywamy to **pseudo-kodem**.

```
1. max = -999999999;
2. read number
3. if(number == -1) print max next stop;
4. if(number > max) max = number
5. go to 2
```

Pętla “while”

- Wykonywanie pewnej części kodu więcej niż jeden raz nazywa się **pętlą**.

```
while my hands are dirty
    I am washing my hands;
```

- ***while*** powtarza wykonywanie tak długo, dopóki warunek jest „true".

```
while(conditional_expression)
    statement;
```

Pętla “while”

- jeśli chcesz aby pętla **while** wykonała więcej niż jedną instrukcję, musisz (jak w przypadku **if**) użyć bloku

```
while(conditional_expression) {
        statement_1;
        statement_2;
        :
        :
        statement_n;
    }
```

```
while(1) {
        cout << "I am stuck inside a loop" << endl;
}
```

Pętla “while”

```
#include <iostream>
using namespace std;
int main(void) {
    /* temporary storage for the incoming numbers */
    int number;
    /* get the first value */
    cin >> number;
    /* we will store the currently greatest number here */
    int max = number;
    /* if the number is not equal to -1 we will continue */
    while(number != -1) {
        /* is the number greater than max? */
        if(number > max)
          /* yes – update max */
          max = number;
        /* get next number */
        cin >> number;
    }
     /* print the largest number */
    cout << "The largest number is " << max << endl;
     /* finish the program successfully */
    return 0;
}
```

Pętla “while”

```
int main(void) {
    /* we will count the numbers here */
    int Evens = 0, Odds = 0;

    /* we will store the incoming numbers here */
    int Number;

    /* read first number */
    cin >> Number;

    /* 0 terminates execution */
    while(Number != 0) {
        /* check if the number is odd */
        if(Number % 2 == 1)
            /* increase „odd” counter */
            Odds++;
        else
            /* increase „even” counter */
            Evens++;
        /* read next number */
        cin >> Number;
    }
    /* print results */
    cout << "Even numbers: " << Evens << endl;
    cout << "Odd numbers: " << Odds << endl;
    return 0;

}
```

Pętla “while”

```
if(Number % 2 == 1) ...
if(Number % 2) ...
```

Pętla “while”

```
int main(void) {
     int counter = 5;

    while(counter != 0) {
          cout << "I am an awesome program" << endl;
          counter--;
    }
    return 0;

}
```

```
int main(void) {
    int counter = 5;

    while(counter) {
          cout << "I am an awesome program" << endl;
          counter--;
    }
    return 0;

}
```

```
int main(void) {
     int counter = 5;

     while(counter--)
          cout << "I am an awesome program" << endl;
     return 0;

}
```

Pętla „do … while”

- Pętla **do while** :
  - warunek jest sprawdzany na końcu,
  - pętla jest wykonywana co najmniej raz, nawet jeśli warunek nie jest spełniony.

```
do
        statement;
while(condition);

do {
      statement_1;
      statement_2;
      :
      :
      statement_n;
} while(condition);
```

Pętla „do … while”

```
#include <iostream>

using namespace std;

int main(void) {
        int number;
        int max = -100000;
        int counter = 0;
        do {
                cin >> number;
                if(number != -1)
                        counter++;
                if(number > max)
                        max = number;
        } while (number != -1);
        if(counter)
           cout << "The largest number is " << max << endl;
        else
           cout << "Are you kidding? You haven't entered any number!" << endl;
        return 0;
}
```

Pętla “for”

- ważniejsze jest **policzenie „przebiegów" pętli** niż sprawdzenie warunków.
- Możemy wyróżnić trzy niezależne elementy :
  - Ustawienie wartości początkowej licznika
  - Sprawdzenie warunku końca
  - Modyfikacja licznika

```
int i;

i = 0;
while (i < 100) {
      /* the body goes here */

      i++;

}
```

```
for(i = 0; i < 100; i++) {
      /* the body goes here */
}
```

Pętla “for”

```
#include <iostream>

using namespace std;

int main(void) {
   int pow = 1;

   for(int exp = 0; exp < 16; exp++) {
      cout << "2 to the power of " << exp << " is " << pow << endl;
      pow *= 2;
   }
   return 0;
}
```

Intrukcje **break** oraz **continue**

- ***break*** – natychmiastowe wyjście z pętli oraz bezwarunkowe zakończenie wykonywanych operacjiw pętli; program przechodzi do wykonania pierwszej instrukcji po ciele pętli

```
#include <iostream>

using namespace std;

int main(void) {
      int number;
      int max = -100000;
      int counter = 0;
      for( ; ; ){
         cin >> number;
         if(number == -1)
               break;
         counter++;
         if(number > max)
               max = number;
      }
      if(counter)
           cout << "The largest number is " << max << endl;
      else
            cout << "Are you kidding? You haven't entered any number!" << endl;
       return 0;
}
```

Intrukcje **break** oraz **continue**

- ***continue*** – zachowuje się tak jakby program wykonał ostatnią instrukcję w pętli; osiągnięto koniec ciała pętli i ponownie sprawdzany jest warunek końca

```
#include <iostream>

using namespace std;

int main(void) {
int number;
int max = -100000;
int counter = 0;

do {
      cin >> number;
      if(number == -1)
            continue;
      counter++;
      if(number > max)
            max = number;
} while (number != -1);
if(counter)
      cout << "The largest number is " << max << endl;
else
      cout << "Are you kidding? You haven't entered any number!" << endl;
return 0;
}
```