1 Podstawy c++ w pigułce.

1.1 Struktura dokumentu.

Kod programu c++ jest zwykłym tekstem napisanym w dowolnym edytorze. Plikowi takiemu nadaje się zwykle rozszerzenie .cpp i kompiluje za pomocą kompilatora, aby utworzyć działającą aplikację. Dla ułatwienia wielu programistów wykorzystuje zintegrowane środowiska programistyczne, które zawierają niezbędne narzędzia do tworzenia programu m.in. edytor i kompilator. Takim środowiskiem jest np. *Dev c++*.
Poniżej przedstawiamy standardowy szablon programu c++:

```
#include<...>
⋮
#include<...>

using namespace std;

main()
{
   kod programu

system("pause");
return 0;
}
```

Omówimy teraz krótko poszczególne elementy tego szablonu.
Grupa wierszy `#include<...>` to grupa tzw. dyrektyw preprocesora. Są to wiersze zawierające specjalne instrukcje dla kompilatora lub przydatne funkcje np. `#include<cmath>` zawiera wiele przydatnych funkcji matematycznych m.in. *sqrt()*, *pow()*, itd. My na początku, oprócz powyższej dyrektywy, będziemy wykorzystywać następującą: `#include<iostream>`. Pozwala ona m.in. wyświetlać tekst na ekranie monitora.
Wiersz `using namespace std;` jest poleceniem, które nakazuje użycia standardowej przestrzeni nazw `std`. Pojęcie przestrzeni nazw służy do określenia, które zmienne i funkcje można użyć w danym miejscu. Można tworzyć własne przestrzenie nazw, jednak my będziemy korzystać ze standardowej.
Następnie definiujemy funkcję o nazwie `main()`, wewnątrz której tworzymy kod programu. Na końcu tej funkcji umieszczamy wiersze: `system("pause")` - polecenie to zatrzymuje wykonanie programu do momentu naciśnięcia jakiegoś klawisza (pozwala to zobaczyć efekt pracy programisty); oraz wiersz `return 0` - polecenie to powoduje, że funkcja `main()` zwraca wartość 0, co oznacza, że program skompilował się pomyślnie.

Zasady stosowane w pisaniu kodu programu.

1. W języku c++ rozróżnia się małe i wielkie litery (polecenia c++ piszemy małymi literami).
2. Każdą instrukcję w kodzie źródłowym należy kończyć średnikiem.
3. W kodzie programu warto używać komentarzy dla poprawienia czytelności kodu szczególnie w obszernych programach. Komentarz jednowierszowy umieszczamy po znaku //, zaś wielowierszowy między znakami /*komentarz*/.

1.2 Deklarowanie zmiennych i typy danych.

Zmienne najogólniej mówiąc służą do przechowywania danych, które potem można zmienić w każdej chwili według potrzeby. Istnieje kilka typów zmiennych:

- char - typ znakowy
- int - liczba całkowita
- short - liczby całkowite krótkie
- long - liczby całkowite długie
- float - liczba zmiennoprzecinkowa (rzeczywista)
- double - liczby zmiennoprzecinkowe podwójnej precyzji
- long double - liczby zmiennoprzecinkowe podwójnej precyzji długie

Wielkość i zakres liczbowy (oprócz typu char) jest uzależniona od systemu operacyjnego.

Typ	**Wielkość w bajtach**	**Zakres liczbowy**
int	2 lub 4	liczby całkowite o takim samym zakresie jak *short* albo *long*
short	2	małe liczby całkowite z zakresu -32 768 do 32 767 ze znakiem albo od 0 do 65 535 bez znaku
long	4	duże liczby całkowite z zakresu -2 147 483 648 do 2 147 483 647 albo od 0 do 4 294 967 295

float	4	liczby zmiennoprzecinkowe
double	8	liczby zmiennoprzecinkowe podwójnej precyzji
long double	10	liczby zmiennoprzecinkowe rozszerzonej podwójnej precyzji

Aby użyć zmiennej należy najpierw ją zadeklarować, czyli podać jej typ np. chcąc zadeklarować zmienną o nazwie *cena* typu całkowitego użyjemy składni `int cena;`. Dobrą praktyką jest deklarowanie zmiennych na początku funkcji `main()`. Ich nazwy powinny opisywać przeznaczenie zmiennej w programie. Nazwa zmiennej może zawierać jedynie litery, cyfry, oraz znak podkreślenia, nie może zawierać spacji i nie może zaczynać się od cyfry. W nazwach rozróżniana jest wielkość liter i nie mogą one pokrywać się ze słowami kluczowymi języka c++ np. *return*, *main*, itp.

1.3 Operatory arytmetyczne i relacyjne w c++.

Na danych liczbowych możemy dokonywać wielu operacji za pomocą operatorów arytmetycznych i relacyjnych. Oto najważniejsze operatory arytmetyczne.

Operator	**Wyrażenie**	**Wynik**	**Opis**
+	6+3	9	dodawanie
-	8-5	3	odejmowanie
*	6*3	18	mnożenie
/	6/4	1	dzielenie całkowite
%	6%5	1	dzielenie modulo
/	9/2	4.5	dzielenie zmiennoprzecinkowe

Wynik z dzielenia dwóch liczb całkowitych jest liczbą całkowitą. Wynik z dzielenia modulo jest resztą z dzielenia dwóch liczb całkowitych.

Teraz przedstawiamy najważniejsze operatory relacyjne.

Operator	**Przykład**	**Opis**

==	$x == y$	sprawdź czy x jest równe y
! =	$x! = y$	sprawdż czy x jest różne od y
<	$x < y$	sprawdź czy x jest mniejsze od y
>	$x > y$	sprawdź czy x jest większe od y
<=	$x <= y$	sprawdź czy x jest mniejsze lub równe y
>=	$x >= y$	sprawdź czy x jest większe lub równe y
&&	$x < 0 \&\& y < 0$	"i" logiczne
||	$x < 0 \|\| y < 0$	"lub" logiczne
++	x++	zwiększ x o 1 (inkrementacja)
−−	x−−	zmniejsz x o 1 (dekrementacja)
+=	x+=4	zwiększ x o 4
-=	x-=3	zmniejsz x o 3
*=	x*=2	zwiększ x 2 razy
/=	x/=6	zmniejsz x 6 razy

1.4 Instrukcja sterująca `if else`.

Podczas pisania programu często musimy dokonywać wyborów np. sprawdzać czy dana liczba jest dodatnia, czy ujemna. Do tego celu wykorzystujemy instrukcję `if else`. Postać tej instrukcji jest następujący:

```
if (warunek)
{
   instrukcja1;
   instrukcja2;
   instrukcja3;
}
   else
{
```

instrukcja4;
instrukcja5;
instrukcja6;
}.

Sposób działanie powyższej instrukcji jest prosty: jeśli warunek w nawiasie jest prawdziwy, wówczas zostaną wykonane instrukcje 1, 2, 3. W przeciwnym razie wykonają się instrukcje 4, 5, 6.

1.5 Instrukcja sterująca `switch`

Instrukcja `switch` jest odmianą instrukcji `if else`. Oto jej postać:

```
switch (liczba całkowita)
{
   case liczba 1:

   instrukcje;
   break;
   ⋮
   case liczba n:

   instrukcje;
   break;

   default:

   instrukcje;
   break;
   }.
```

Instrukcja ta pobiera zmienną całkowitą i porównuje jej wartość z kolejnymi możliwymi wartościami *liczba n* wyszczególnionymi w kolejnych instrukcjach `case`. Gdy te wartości będą równe wówczas wykonane zostaną instrukcje w odpowiednim wyrażeniu `case`, a następnie wykona się instrukcja `break`, która spowoduje opuszczenie ciała instrukcji `switch`. Na końcu pojawia się opcjonalny przypadek `default` (odpowiednik `else`), w którym instrukcje zostaną wykonane wtedy, gdy nie zajdzie żaden z poprzednich przypadków.

1.6 Pętla `for`

Oprócz instrukcji warunkowych w tworzeniu aplikacji wykorzystujemy pętle, których zadaniem jest wykonanie ustalonych instrukcji programu określoną liczbę razy. Jedną

z takich pętli jest pętla `for`. Poniżej przedstawiamy jej postać:

```
for(wyrażenie początkowe; warunek; wyrażenie końcowe)
   {
   instrukcje;
   }
```

Najpierw wykonywane jest `wyrażenie początkowe`, a następnie sprawdzany jest `warunek`. Jeśli jest prawdziwy wówczas wykonywane są `instrukcje`, po czym zostaje wykonane `wyrażenie końcowe`. Następnie znowu sprawdzany jest `warunek` i cała procedura jest wykonywana od nowa (`wyrażenie początkowe` wykonywane jest tylko jeden raz) aż do momentu, gdy `wyrażenie końcowe` stanie się fałszywe.

1.7 Pętle `while` i `do while`.

Kolejnymi przykładami pętli są pętle `while` i `do while`. Oto ich postać:

```
while(warunek)
   {
   instrukcje;
   }
```

oraz dla pętli `do while`

```
do
   {
   instrukcje;
   }
while(warunek)
```

W przypadku pętli `while` najpierw sprawdzany jest `warunek` i jeśli jest prawdziwy, wtedy zostają wykonane `instrukcje`; jeśli w`warunek` jest fałszywy następuje wyjście z pętli i wykonują się dalsze instrukcje programu.

W przypadku pętli `do while` zostają wykonane `instrukcje` dopóki `warunek` jest prawdziwy; gdy `warunek` jest fałszywy następuje wyjście z pętli i wykonują się dalsze instrukcje programu.
Różnica między tymi pętlami polega jedynie na tym, że w przypadku pętli `do while` `instrukcje` wykonają się przynajmniej jeden raz (gdyż `warunek` sprawdzany jest na końcu), w przeciwieństwie do pętli `while`, gdzie `instrukcje` mogą się nie wykonać ani razu (`warunek` sprawdzany jest na początku).

1.8 Tablice

Do tej pory poznaliśmy zmienne różnych typów, w których mogliśmy przechowywać tylko pojedyncze wartości. Zaletą tablic jest możliwość przechowywania wielu wartości w ramach jednej zmiennej.

Podobnie jak w przypadku innych zmiennych tablica musi mieć określony typ; nie może natomiast przechowywać wartości różnych typów. Aby utworzyć tablicę musimy podać jej typ, nazwę oraz liczbę elementów. Oto przykładowa deklaracja tablicy:

```
int mojaTablica[10];
```

Możemy też jednocześnie zadeklarować tablicę i przypisać do niej elementy:

```
int mojaTablica[4]={4, 6, 8, 3};
```

Aby odwołać się do wybranego elementu tablicy podajemy jej nazwę i indeks, który jest umieszczony w nawiasach kwadratowych. Należy przy tym pamiętać, że pierwszy element jest indeksowany przez 0 a nie przez 1. Poniższy zapis informuje nas, że pierwszym elementem tablicy o nazwie *mojaTablica* jest liczba 13.5:

```
mojaTablica[0]=13.5;
```

Praca z pojedynczymi elementami tablicy należy do rzadkości. Zazwyczaj w tworzeniu programu wykorzystujemy wszystkie jej elementy. Stąd też często wykorzystuje się pętle `for` w pracy z tablicami. Symbolicznie fakt ten możemy zapisać następująco (zakładamy, że tablica składa się z 10 elementów):

```
for(int i=0; i<10; i++)
{
instrukcje dotyczące tablicy;
}
```

Powyżej przedstawiliśmy tablice jednowymiarowe. podobne zasady dotyczą tablic dwuwymiarowych. Oto niektóre z nich:
Przykładowa deklaracja:

```
float macierz[2][3];
```

Przykładowa deklaracja z przypisaniem wartości:

```
int macierz[2][3]={2, 3, 4, 5, -2, 4};
```

Wykorzystanie wszystkich elementów tablicy w programie (zakładamy, że tablica ma 2 wiersze i trzy kolumny):

```
for(int i=0; i<2; i++)
 {
  for(int j=0; j<3; j++)
      instrukcje dotyczące tablicy
          }
```