Arytmetyka, ify i strumienie, błędy i ich zrozumienie

Czyli coś policzymy, a potem to popsujemy

Na poprzednich zajęciach poza uruchamianiem programu poznaliśmy też podstawowe obliczenia arytmetyczne. Teraz pójdziemy o krok dalej i zmontujemy dziś kilka programów, które będą coś nam liczyły. Napisałbym co, ale w momencie pisania tego akapitu jeszcze nie wymyśliłem.

Zaczynając od przypomnienia:

```
#include <iostream>
using namespace std;
int main(){

int a = 6, b = 7;
cout<<"a+b="<<a+b<<endl;
cout<<"a*b="<<a*b<<endl;
cout<<"a-b="<<a-b<<endl;
cout<<"a/b="<<a/b<<endl;

return(0);
}
```

Taki programik wydrukuje nam na terminalu:

```
a+b=13
a*b=42
a-b=-1
a/b=0
```

Szybkie pytania:

- Dlaczego wynikiem dzielenia jest 0, a nie prawidłowe 0.857143?
- Dlaczego, inaczej niż poprzednio, nie używam przedrostka `std::` przed `cout` i `endl`?

Strumień wejściowy

Na razie operujemy tylko na strumieniu wyjściowym, czyli program zwraca nam na terminal jakieś informacje. Chcielibyśmy jednak czasem jakieś dane do niego wprowadzić – jak to zatem zrobić? Nic trudnego, należy skorzystać ze strumienia wejściowego. Standardowe wejście jest w naszym wypadku dokładnie takie samo, jak standardowe wyjście, więc nadal będziemy wprowadzali dane na terminalu. Skorzystamy ze strumienia `std::cin` oraz operatora `>>`.

```
float a = 6, b;
cout<<"Podaj liczbę b: \n";
cin>>b;
cout<<"a+b="<<a+b<<endl;
```

W ten sposób rozpoczynamy jakotaką komunikację z programem – możemy już podawać argumenty. Trzeba się jednak liczyć z tym, że nie tylko cywilizowani ludzie będą korzystać z tego co napiszemy – może zdarzyć się sytuacja, w której ktoś nam będzie chciał coś popsuć.

Szybkie pytanie

- Co się stanie, jak poproszeni o liczbę wpiszemy literę albo inny znak?

Zawsze warto się zabezpieczać – są różne sposoby, ale jednym z nich może być zastosowanie metody `cin.fail()`

```
float a = 6, b;
cout<<"Podaj liczbę b: \n";
cin>>b;
cout<<cin.fail()<<endl;
cin>>a;
cout<<"a+b="<<a+b<<endl;
```

Wprowadź odpowiednie zmiany do kodu.

- Co się stanie jak wpiszemy literę poproszeni o liczbę b?
- Co się stanie jak zrobimy to poproszeni o liczbę a?
- Co się stanie z wcześniej zadeklarowanym a, kiedy poproszeni podamy nową wartość?

Z technicznego punktu widzenia, w momencie pojawienia się błędu odczytu przy użyciu cin, np. w momencie, w którym oczekiwany jest float, a pojawi się char, ustawiana jest flaga błędu, którą sprawdzić możemy właśnie poprzez wywołanie metody `cin.fail()`. Powoduje ona, że wszystkie pozostałe próby wczytania przy użyciu `cin` są ignorowane. Usunąć tę flagę możemy przy użyciu `cin.clear()` – jednak po usunięciu flagi strumień nie jest czyszczony – posprzątać go możemy np. takim wywołaniem `cin.ignore(1000, \n)`

Operatory porównawcze i logiczne, instrukcje warunkowe

Jest jeden typ danych, o którym wcześniej nie powiedziałem, jest to *boolean*, potocznie *bool*. Jest to typ danych, który przyjąć może dwie wartości – true lub false – alternatywnie, odpowiednio 1 lub 0. Jest to typ logiczny, używać będziemy go w – niespodzianka – operacjach logicznych. Omówimy pokrótce trzy podstawowe typy, AND, OR i NOT

AND - &&			**OR - ||**			**NOT - !**	
A	B	Wynik	A	B	Wynik	A	Wynik
0	0	0	0	0	0	0	1
1	0	0	1	0	1	1	0
0	1	0	0	1	1		
1	1	1	1	1	1		

```
#include <iostream>
using namespace std;
int main(){

bool a = true, b = false, c = a&&b;
cout<<c<<endl;
return(0);
}
```

Polecam przetestować na tym kodzie chociaż kawałek zamieszczonej wyżej tabelki.

Są jeszcze operatory porównawcze, które posłużą nam - kolejna niespodzianka - do porównań.

- A > B – …większe od…
- A < B – …mniejsze od…
- A >= B – …większe lub równe od…
- A <= B – …mniejsze lub równe od…
- A == B – …równe…
- A != B - …nierówne…

Wydaje się dość prostym do zrozumienia konceptem. Takie porównania również zwracają nam boolean, czyli true albo false. Czyli na przykład:

```
int a = 7, b =8;

cout<<(a>b)<<endl;
```

wydrukuje nam 0, czyli fałsz, bo 7 nie jest większe od 8. Co zatem wydrukuje się po wykonaniu poniższego kodu?

```
int a = 7, b =8;
cout<<(a>b||b<10)<<endl;
```

Do czego to nam może być potrzebne? Między innymi do instrukcji warunkowych – czyli „ifów” – posiadają one następującą składnię:

```
if (wyrażenie_logiczne) {
//blok instrukcji
}
```

instrukcje znajdujące się wewnątrz bloku instrukcji wykonane zostaną dopiero wtedy, kiedy wyrażenie logiczne podane w tej instrukcji będzie spełnione, czyli true.

```
bool a = true, b = false, c = a&&b;

if (c){
    cout<<"Wyrazenie spelnione!";
}
```

- Co pojawi się na terminalu, gdy wykonamy powyższy program?
- Jak możemy zmienić tę sytuację? Jest kilka prawidłowych odpowiedzi

Warto zauważyć, że wyrażenie logiczne w instrukcji warunkowej może być bardziej złożone:

```
if (a&&b){
    cout<<"Wyrazenie spelnione!";
}
```

Powrócimy na chwilę do poprzedniego przykładu, ale zmodyfikujemy go odrobinę:

```
float a = 6, b;
cout<<"Podaj liczbę b: \n";
cin>>b;
cout<<cin.fail()<<endl;
cin>>a;

if (cin.fail()){
    cout<<"coś skaszaniłeś, byczqqq"<<endl;
}
else{
    cout<<"a+b="<<a+b<<endl;
}
```

Przeanalizujmy program. Co się stanie, gdy podamy liczbę? A co, gdy wpiszemy literę?

Pojawia się tu słówko kluczowe else. Jest ono dość specyficzne, ponieważ jest ściśle powiązane ze znajdującym się powyżej ifem. Blok instrukcji podany instrukcji wykona się w sytuacji, w której warunek znajdującego się bezpośrednio wcześniej ifa nie zostanie spełniony.

Szybkie zadanie na zajęcia

Napisz programik, który poprosi użytkownika o podanie wieku, a następnie określi, czy jest pełnoletni i zwróci taką informację.

Zadanie domowe

Napisz program, który przyjmie od użytkownika dwie liczby i sprawdzi, czy:

- Ich suma jest większa od 100
- Ich różnica jest większa od 0
- Ich iloczyn jest większy lub równy 100
- Ich iloraz nie jest równy 2

O każdym z tych wyników program powinien nas poinformować. Powinien też być (chociaż trochę) zabezpieczony przed błędami (np. podaniem wpisaniem litery).