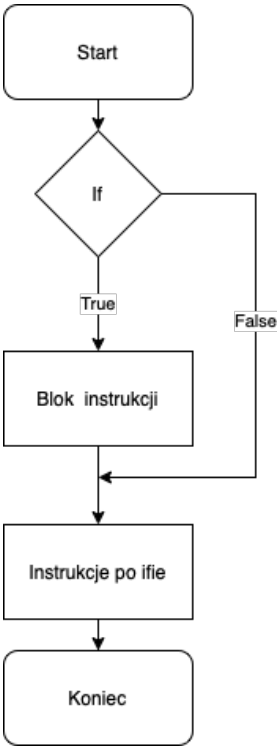
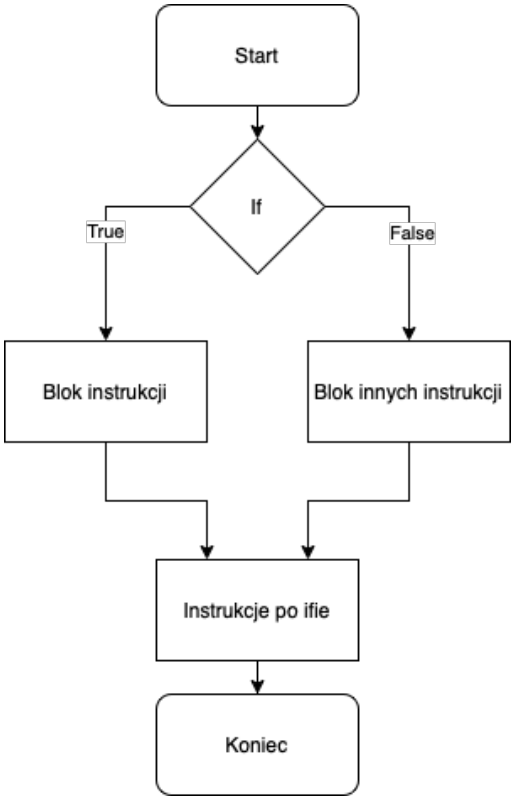
Więcej warunków, pętle i może cośtam jeszcze

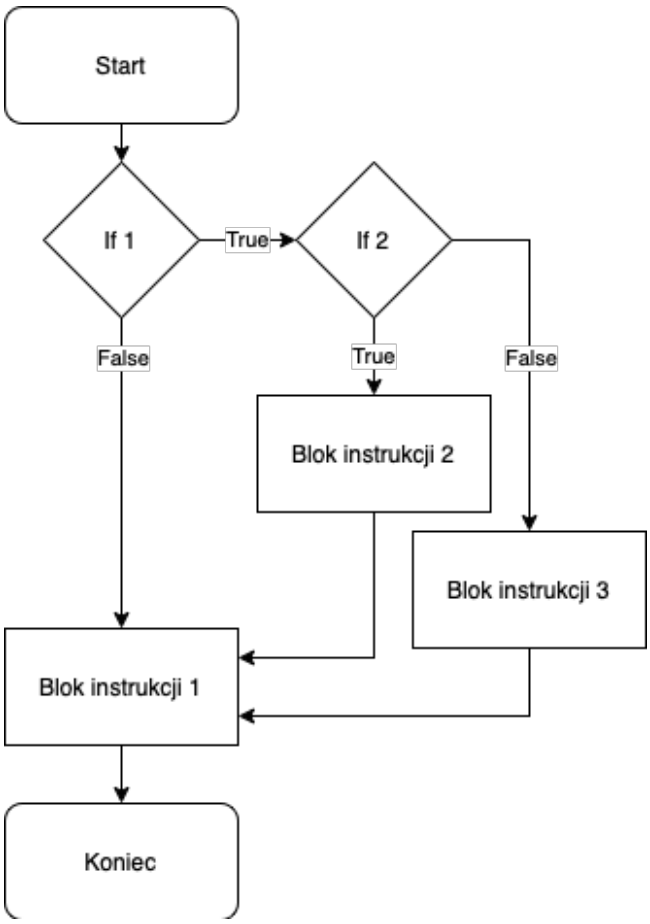
Poprzednio poznaliśmy instrukcje warunkowe, teraz sobie tę wiedzę ociupinkę rozszerzymy, zagłębimy się bowiem w tajniki instrukcji bardziej złożonych. Zastosowań jest wiele, ale do sterowania zachowaniem programu są to konstrukty w zasadzie kluczowe.



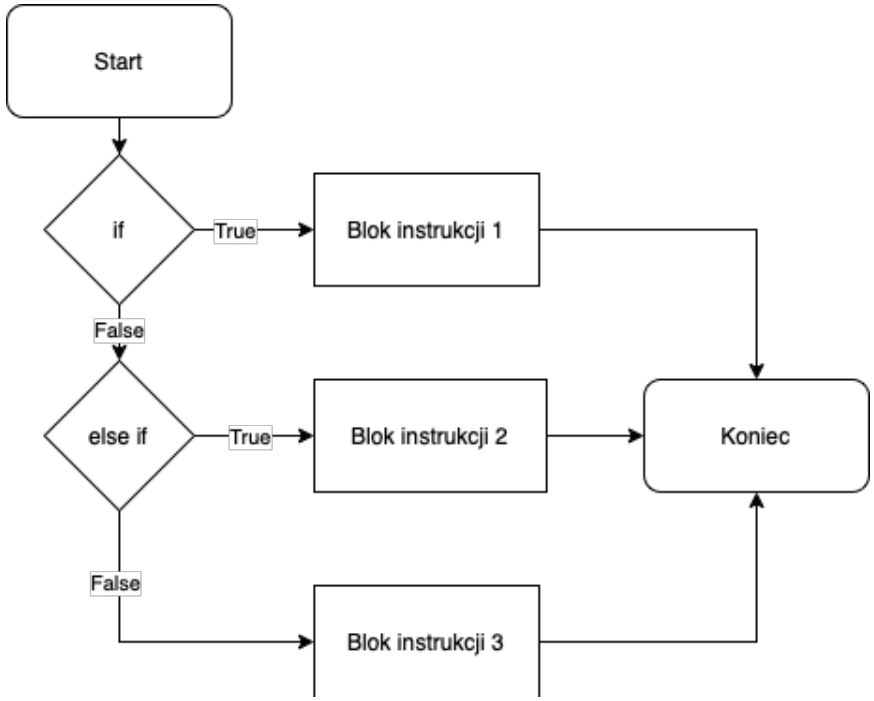

```
int main(){
    bool condition;
    if (condition){
        //this will be executed only if
condition is true
    }
    //this will always be executed
return 0;
}
```

```
int main(){
    bool condition;
    if (condition){
        //this will be executed
only if condition is true
    }

    else{
        //this will be executed
only if condition is false
    }

    //this will always be executed
return 0;
}
```






```
int main(){
    bool condition1, condition2;
    if (condition1){
        if (condition2){
            //this will be executed only if condition1 and
condition2 are true
        }
        else{
            //this will be executed if condition1 is true and
condition2 is false
        }
        //this will be executed only if condition1 is true
    }
    else{
        //this will be executed only if condition1 is false
    }
    //this will always be executed
return 0;
}
```




```
int main(){
    bool condition1, condition2;
    if (condition1){
        //this will be executed only if condition1 is true
    }
    else if(condition2){
        //this will be executed only if condition1 is false
and condition2 is true
    }
    else{
        //this will be executed only if condition1 is false
and condition2 is false
    }
    //this will always be executed
return 0;
}
```

Wykorzystując powyższe jesteśmy w stanie w bardzo elegancki sposób wysterować zachowaniem programu. Przypomnę jeszcze o `cin.fail()`, które zwraca nam boolean o błędzie (lub jego braku) przy wczytaniu z konsoli.

Zadanko na zajęcia

Napisz program, który wczyta liczbę, a następnie sprawdzi, czy liczba jest podzielna przez 3 oraz czy jest parzysta. Chciałbym zobaczyć na terminalu jeden z 5 outputów:

- Liczba podzielna przez 3
- Liczba podzielna przez 2

- Liczba podzielna przez 2 i 3
- Liczba niepodzielna przez 2 ani 3
- Error

Jeszcze jedno zadanko na zajęcia, szybsze

Napisz programik, wczytujący 3 liczby, a następnie wypisujący najmniejszą z nich

Pętle while i do-while

Pętle są wyjątkowo ważnym i potężnym narzędziem, które będziemy wykorzystywać bardzo intensywnie. Pozwolą nam one na wielokrotne wykorzystywanie instrukcji bez konieczności wielokrotnego ich umieszczania w kodzie. Pozwalają również na zaawansowane sterowanie zachowaniem programu – np. wykonywanie obliczeń (lub dowolnej innej akcji) wielokrotnie, dopóki użytkownik nie uzna, że pora skończyć.

Pętla while w swojej składni jest poniekąd podobna do instrukcji warunkowej. Przyjmuje ona bowiem jeden argument typu boolean, ale w odróżnieniu od instrukcji warunkowej, wykonywać się ona będzie wielokrotnie, tak długo, aż argument nie stanie się false. Dla przykładu:

```
while(true){
        cout<<"ratunku, utknąłem w nieskończonej
pętli!"<<endl;
    }
```

Zaspamuje nam terminal drukując tekst wiele razy na sekundę, a potem nie będziemy wiedzieli jak z tego wyjść. Ctrl+C, w razie czego.
Powyższa pętelka jest tzw. Pętlą nieskończoną. Wykonywać się będzie tak długo, jak działać będzie program. Czyli dopóki go nie ubijemy, to zablokowaliśmy terminal.

```
int main(){
    int przejscie = 1;
    while(przejscie<100){
        cout<<"przejscie petli nr "<<przejscie<<endl;
        przejscie++;
    }
return 0;
}
```

Powyższy programik jest już (odrobinę)bardziej sensowny, wypisze on nam bowiem, w którym przejściu pętli się znajdujemy i wykonywać się będzie tak długo, jak długo zmienna przejście będzie < 100. Pojawia się tu też dość ważna instrukcja, mianowicie inkrementacja, konkretniej postinkrementacja. Jest to dokładnie to samo, co

```
przejscie += 1;
przejscie = przejscie + 1;
```

Wypadku zastosowania postinkrementacji wartość zmiennej zwiększana jest o 1, po wykorzystaniu tejże zmiennej. Czyli najpierw odczytujemy, a potem inkrementujemy. Czyli w zasadzie w powyższym przykładzie ciało pętli możemy zmienić na pojedynczą linię

```
cout<<"przejscie petli nr "<<przejscie++<<endl;
```

Warto w tym miejscu wspomnieć jeszcze o preinkrementacji. W tym wypadku zmienna jest najpierw zwiększana, a dopiero potem odczytywana. Czyli w wypadku umieszczenia w naszym przykładzie:

```
cout<<"przejscie petli nr "<<++przejscie<<endl;
```

W połączeniu z tym, że zmienna w momencie wejścia do pętli ma już wartość 1 spowoduje, że program już w pierwszym obejściu zwróci nam „przejscie petli nr 2”, ale za to skończy się nie na 99, a na „przejscie petli nr 100”.

Pętla do… while jest szczególnym przypadkiem pętli while, w którym najpierw wykonywana jest instrukcja, a dopiero potem następuje weryfikacja, czy warunek w instrukcji while() jest spełniony. Co za tym idzie, taka pętla zawsze będzie wykonana co najmniej raz.

```
do{
        cout<<"to się wydrukuje tylko raz, bo while ma podany
false"<<endl;
    } while (false);
```

Zadania na punkty

1. Napisz program, który na początku przyjmie od użytkownika liczbę operacji, które ma wykonać. Następnie przez pierwszą połowę swojego działania będzie prosił użytkownika o podanie liczby i dodawał je kolejno do siebie, a w drugiej połowie swojego działania będzie wczytane liczby od ogólnego wyniku odejmował. Na końcu działania powinien zwrócić ostateczny wynik.
   Na przykład, dla 5 operacji i podanych kolejno: 5,4,2,2,1 wynik powinien wynieść 2.

2. Napisz kalkulator, który będzie na początku swojego działania przyjmował od użytkownika znak. Jeśli ten znak będzie symbolem operacji arytmetycznej, to program powinien poprosić o podanie dwóch liczb (`cin>>a>>b;` też zadziała), a następnie zwróci wynik obliczeń i wróci o proszenia o podanie znaku. Jeśli użytkownik poda znak q, to program powinien się zakończyć, jeśli będzie to jakikolwiek inny znak (nie symbol operacji i nie „q”), to program powinien zwrócić informację o błędzie i powrócić do początku
   TO ZADANIE BĘDZIE ŁATWIEJSZE PO NASTĘPNYCH ZAJĘCIACH, ALE MOŻNA PRÓBOWAĆ JUŻ TERAZ.