Pętla for, switch-case, bardziej złożone programy

Kiedy dokładnie wiemy, ile razy powinniśmy wykonać jakąś operację (bądź zestaw operacji), zamiast pętli while użyjemy pętli for. Ma ona bardzo przyjemną składnię, która w sposób precyzyjny pozwoli sterować blokiem instrukcji, równocześnie pozostając bardziej czytelną, niż pętle while i do…while.

```
for (int i=0; i<10; i++){
    cout<<"przejscie petli: "<<i<<endl;
}
```

W nawiasach umieszczamy trzy, oddzielone średnikami, *statementy*.

1. Inicjalizacja pętli - `int i=0` – ta instrukcja wykona się tylko jeden raz
2. Warunek pętli - `i<10` – pętla będzie się wykonywać do momentu spełnienia warunku
3. Krok pętli - `i++` - instrukcja wykonywana po każdym przejściu pętli

Pętla for ma taką właściwość, że można w niej stworzyć zmienną, która będzie widoczna tylko wewnątrz pętli, a po jej opuszczeniu już nie. W powyższym przykładzie jest to zmienna *i* typu integer. Żeby zobrazować:

```
for (int i=0; i<4;i++){
    cout<<"i ma wartość "<<i<<endl;
}
cout<<"a po przejściu pętli i ma wartość"<<i<<endl;
```

identyfikator "i" jest niezdefiniowany C/C++(20)

Wyświetl problem Szybka poprawka... (⌘.)

Oczywiście możemy pracować na zmiennej zadeklarowanej poza pętlą i na różnych wartościach, ale przypadek na górze jest stosowany w zdecydowanej większości przypadków. Akademicki obowiązek nakazuje jednak pokazać:

```
int i=0;
for (; i<4;i++){
    cout<<"i ma wartość "<<i<<endl;
}
cout<<"a po przejściu pętli i ma wartość "<<i<<endl;
```

Wydrukuje:

```
i ma wartość 0
i ma wartość 1
i ma wartość 2
i ma wartość 3
a po przejściu pętli i ma wartość 4
```

Czasem może się zdarzyć sytuacja, że z jakiegoś powodu w trakcie wykonywania pętli okaże się, że nie będziemy już jej potrzebowali i nie musi wykonywać się do końca i przerwanie jej działania zaoszczędzi nam mnóstwo czasu. Użyjemy wtedy wyrażenia `break;`

Wyjątkowo głupi przykład:

```
for (int i=0; i<10;i++){
    cout<<"i ma wartość "<<i<<endl;
    if (i==4){
        cout<<"i ma wartość 4, koniec zabawy";
        break;
    }
}
```

Powyższy przykład wykona się tylko do momentu, w którym i osiągnie wartość 4, a potem zakończy wykonywanie pętli przejdzie do dalszej części programu

Jest jeszcze instrukcja `continue`, która zadziała w podobny sposób, ale nie do końca. W momenci natknięcia się na to wyrażenie, program natychmiast przejdzie do wykonania następnego kroku pętli.

```
for (int i=0; i<10;i++){
    if (i==4){
        continue;
    }
    cout<<"i ma wartość "<<i<<endl;
}
```

Ten przykład wydrukuje nam informację o tym, że i ma wartość 1, 2, 3, 5, 6 itd. Pominie zatem krok w którym drukuje 4.

Switch...case

`switch()`, w połączeniu ze słówkiem kluczowym `case` pozwoli nam wybrać jeden z wielu bloków kodu. W nawiasach `switch()` umieszczamy wyrażenie, a instrukcja wybierze `case` spełniający warunki wyrażenia.

Ważne jest, żeby zauważyć, że wykonany zostanie skok do odpowiedniego `case,` ale po jego wykonaniu kod zostanie wykonany normalnie, po kolei. Dlatego po każdym bloku kodu we wszystkich case'ach użyć musimy wspomnianego wcześniej `break`.
Do obsługi nieobsłużonych wariantów użyjemy słówka kluczowego `default.` To taki case, do którego będziemy skakać, jeśli żaden z wcześniej wymienionych nie zadziała.

```
int number=1;
switch (number){
    case 1:
        cout<<"wybrano 1\n";
        break;
    case 2:
        cout<<"wybrano 2\n";
        break;
    case 3:
        cout<<"wybrano 3\n";
        break;
    default:
        cout<<"nie wybrano ani 1, ani 2, ani 3\n";
}
```

Wracamy do pętli for

Poprzednio zagnieżdżaliśmy wyrażenia warunkowe, teraz przejdziemy do zagnieżdżania pętli. Koncept działania jest bardzo prosty, po prostu umieszczamy pętlę w pętli.

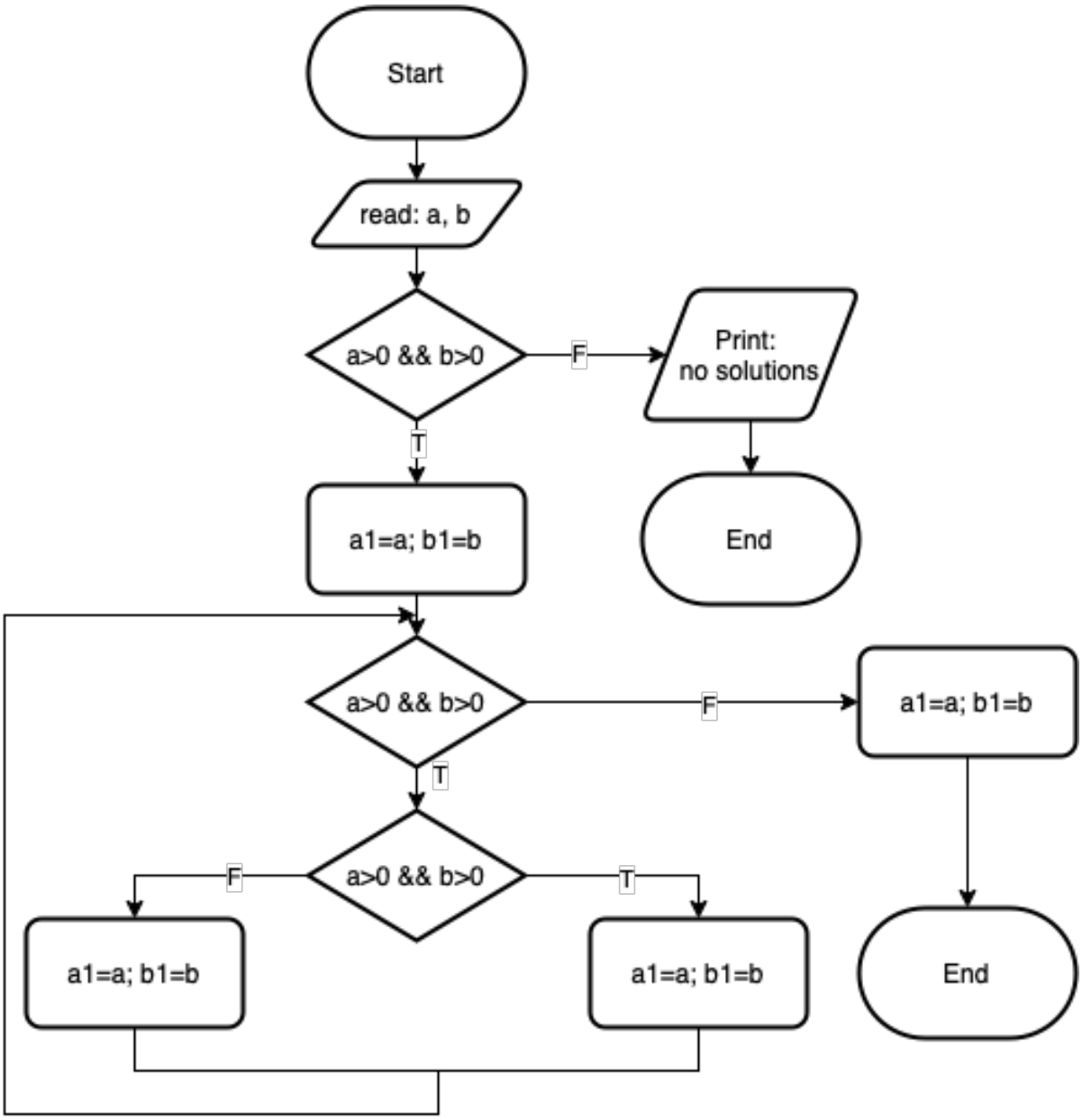
```
for (int i =0; i<5; i++){
    for (int j=0; j<3; j++){
        cout<<"i: "<<i<<", j: "<<j<<endl;
    }
}
```

Pętla wewnętrzna wykona się z każdym nowym obejściem pętli zewnętrznej, czyli w powyższym przypadku dla każdego *i* wykonamy pętlę zagnieżdżoną trzykrotnie. Łącznie 15 razy, w każdym wypadku *j* będzie startować od zera. Elegancko.

Zestaw zadań

Podczas zajęć przećwiczymy wspólnie implementację algorytmu z pętlą while oraz zaimplementujemy 1-2 wybrane przez was zadania z pętli for. Pozostałe traktujemy jako zadania do samodzielnego wykonania

1. Zaimplementuj poniższy algorytm obliczający największy wspólny dzielnik



2. Napisz program, rysujący pionową kreskę ze znaków # o wysokości wczytanej z klawiatury
3. Napisz program rysujący pusty prostokąt ze znaków # o wymiarach 7x5
4. Napisz program rysujący równoramienny trójkąt prostokątny ze znaków # o wysokości wczytanej z klawiatury
5. Napisz program rysujący literkę A z literek A
6. Napisz program wypisujący pełną tabliczkę mnożenia o rozmiarze podanym z klawiatury (max 9x9) – tego na zajęciach nie zrobimy, możemy omówić