

Podstawy Programowania

Zajęcia laboratoryjne 3

I rok Bioinformatyki Politechniki Poznańskiej

Sterowanie wykonywaniem programu cz.2

1 Instrukcja warunkowa *else if*

W poprzednim protokole zapoznaliśmy się z instrukcjami warunkowymi *if* oraz *else*. Jako rozwinięcie podstawowej instrukcji *if* pojawia się instrukcja *else if*, która umożliwia ustalenie większej liczby opcjonalnych warunków i scenariuszy z nimi związanych. Składnia przedstawiona została poniżej:

```
1 if(warunek1){
2     //wykonaj instrukcje jesli warunek1 jest TRUE
3 }
4 else if(warunek2){
5     //wykonaj instrukcje jesli warunek1 jest FALSE, a warunek2 jest TRUE
6 }
7 else if(warunek3){
8     //wykonaj instrukcje jesli warunek1 i warunek2 sa FALSE, a warunek3 jest TRUE
9 }
10 else{
11     //wykonaj instrukcje jesli wszystkie powyzsze warunki sa FALSE
12 }
```

Przetestuj poniższy kod zawierający instrukcje warunkowe *else if* (kod 1 dostępny w materiałach na stronie). Zmień wartość dla zmiennej *liczba* i zwróć uwagę na to jaki będzie wynik.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int liczba = 5;
6
7     if (liczba > 0) {
8         printf("liczba %d jest dodatnia", liczba);
9     }
10    else if(liczba < 0) {
11        printf("liczba %d jest ujemna", liczba);
12    }
13    else{
14        printf("liczba %d jest rowna 0", liczba);
15    }
16    return 0;
17 }
```

Przetestuj poniższy kod (kod 2 dostępny w materiałach na stronie). Zwróć uwagę, że ten kod wykorzystuje instrukcje *if* i *else*, przyjrzyj się wynikowi działania programu. Następnie zmień poniższe instrukcje na instrukcje warunkowe *else if*. Czy dostrzegasz różnicę w wynikach i rozumiesz z czego wynikają?

```
1 #include <stdio.h>
2 int main()
3 {
```

```

4   char znak = 'b';
5   int liczba = 3;
6   int inna = 5;
7
8   if(znak == 'a' || znak == 'b')
9       printf("instrukcja wykona sie jesli znak = a lub jesli znak = b\n");
10
11  if(znak == 'b' && liczba == 13)
12      printf("instrukcja wykona sie jesli znak = b i jesli liczba = 13\n");
13
14  if(znak != 'A' )
15      printf("instrukcja wykona sie jesli znak bedzie rozny od wielkiej litery A\n");
16
17  if (znak == 'a' || (liczba == 3 && inna == 7))
18      printf("TRUE\n");
19  else
20      printf("NOT TRUE\n");
21  return 0;
22 }

```

2 Instrukcja *switch...case*

Instrukcja *switch...case* pozwala na utworzenie wielu warunków i wykonania konkretnych instrukcji w zależności od wartości zmiennej. Funkcja ta umożliwia utworzenie czytelnego menu programu. Zwróć uwagę na składnię w kodzie poniżej: po słowie kluczowym *switch* w nawiasach należy podać wyrażenie sterujące, po słowie kluczowym *case* należy określić wartość zwracaną przez wyrażenie sterujące i wstawić dwukropek. Dopiero po dwukropku można wypisać instrukcje, które wykonają się dla konkretnej wartości *case*. Po zaprogramowaniu odpowiednich instrukcji należy podać słowo kluczowe *break*, które kończy wykonywanie instrukcji i przerywa działanie *switch...case'a*. Instrukcja *switch* powinna obsłużyć przypadek w którym podano wartość nieobsługiwana przez żaden blok *case*, w takiej sytuacji należy określić instrukcję z bloku *default*. Przetestuj poniższy kod (kod 3 dostępny w materiałach na stronie). Jak myślisz, co się stanie jeśli usuniesz słowo kluczowe *break* z pewnego *case'a*?

```

1  #include <stdio.h>
2  int main()
3  {
4      int opcja;
5      printf("wybierz opcje 1-3\n");
6      scanf("%d", &opcja);
7
8      switch (opcja)
9      {
10         case 1:
11             printf("wybrales opcje 1\n");
12             break;
13
14         case 2:
15             printf("wybrales opcje 2\n");
16             break;
17
18         case 3:
19             printf("wybrales opcje 3\n");
20             break;
21
22         default:
23             printf("podano inna opcje niz 1, 2 czy 3\n");
24             break;
25     }
26     return 0;
27 }

```

3 Instrukcja *break* i *continue*

W przypadku instrukcji *switch...case* już pojawiło się użycie instrukcji *break*. Jednakże ta instrukcja może także pojawiać się w pętlach i będzie przerywała wykonywanie pętli. Zatem użycie instrukcji *break* ma zastosowanie wszędzie tam, gdzie dalsze powtarzanie instrukcji w pętli nie będzie już konieczne. Przykładowe użycie *break* znajduje się poniżej (kod 4 dostępny w materiałach na stronie).

```
1 #include <stdio.h>
2 int main()
3 {
4     for(int i = 0; i < 10; i++)
5     {
6         if (i == 5)
7             break; //przerwij działanie petli jesli i == 5
8
9         printf("%d\n", i);
10    }
11    return 0;
12 }
```

Kolejna instrukcja, *continue*, podobnie jak *break* jest związana z pętlami. Jej pojawienie się w ciele pętli powoduje przerwanie wykonywania się bieżącej iteracji pętli (instrukcji następujących po tym słowie kluczowym) i powoduje przejście do kolejnej iteracji. Użycie instrukcji *continue* ma zastosowanie wszędzie tam, gdzie nie ma konieczności wykonania instrukcji pojawiających się po słowie kluczowym *continue*, ale są umieszczone w bloku pętli. Z tą różnicą, że przerywane jest działanie danej iteracji, a nie całej pętli jak w przypadku słowa kluczowego *break*. Przykładowe użycie *continue* znajduje się poniżej (kod 5 dostępny w materiałach na stronie).

```
1 #include <stdio.h>
2 int main()
3 {
4     for(int i = -5; i <= 5; i++)
5     {
6         if (i == 0)
7             continue; //przerwij wykonywanie dalszych instrukcji
8                       //i przejdź do kolejnej iteracji petli
9
10        printf("%d\n", i);
11    }
12    return 0;
13 }
```

4 Zadania

Zad 1. Napisz program, który poprosi użytkownika o wprowadzenie z klawiatury dwóch wartości dla zmiennych całkowitych a i b . Następnie program powinien sprawdzić czy wprowadzone liczby są sobie równe, jeśli tak jest to kod wyświetla odpowiedni komunikat. Jeśli a i b nie są sobie równe to program powinien sprawdzić czy $a < b$. Jeśli $a < b$ jest to kod wyświetla odpowiedni komunikat. W innym wypadku program powinien wyświetlić, że $a > b$.

Zad 2. Wykorzystaj kod z poprzedniego zadania i zastanów się jak zmienić kod, aby zamiast instrukcji *else if* wykorzystać zagnieżdżone instrukcje *if* i *else*.

Zad 3. Napisz program, który określi czy wprowadzony przez użytkownika współczynnik r wskazuje na brak, słabą, średnią, silną czy bardzo silną korelację dodatnią, zgodnie z niżej opisanymi założeniami:

- $0.7 \leq r \leq 1$ bardzo silna korelacja dodatnia
- $r \geq 0.5$ silna korelacja dodatnia
- $r \geq 0.3$ umiarkowana korelacja dodatnia
- $r \geq 0.2$ słaba korelacja dodatnia
- $r \geq 0$ brak korelacji

Zad 4. Napisz program, który będzie prostym kalkulatorem implementującym działanie dodawania, odejmowania, dzielenia. Program prosi użytkownika o wprowadzenie dwóch liczb zmiennoprzecinkowych oraz o wprowadzenie znaku dla pożądanego działania ('+' lub '-' lub '*' lub '/'). Ponadto program powinien zawierać zabezpieczenie dla dzielenia przez 0; odpowiedni warunek przy którym pojawi się komunikat "nie można dzielić przez 0!".

Zad 5. Przetestuj poniższy kod (kod 6 dostępny w materiałach na stronie) i spróbuj napisać własną pętlę nieskończoną *while()*. Zwróć uwagę, że pętla *while* wymaga podania warunku, niech ten warunek będzie zawsze prawdziwy.

```

1 #include <stdio.h>
2 int main()
3 {
4     for(int i = 0; ;i++)
5     {
6         printf("%d ", i);
7     }
8     return 0;
9 }
```

Warto zapoznać się z typem logicznym *bool*, który można wykorzystać by warunek nieskończonej pętli *while* był zawsze prawdziwy (kod 7 dostępny w materiałach na stronie):

```

1 #include <stdio.h>
2 #include <stdbool.h>
3 int main()
4 {
5     bool zmienna_logiczna = true;
6
7     if(zmienna_logiczna == true)
8         printf("PRAWDA");
9     else
10        printf("FALSZ");
11
12    return 0;
13 }
```

Zad 6. Wykorzystaj prosty kalkulator napisany w Zad 4. i zmień go tak aby wykonywał się w nieskończonej pętli. Niech użytkownik zdecyduje kiedy zakończyć działanie programu. Program poza zapytaniem jakie działanie wykonać, powinien także zapytać czy zakończyć swoje działanie. Aby zakończyć działanie programu wykorzystaj funkcję *return*.

Zad 7. Napisz program, który zawiera pętlę nieskończoną, która wyświetla kolejne liczby całkowite z tym, że każdą podzielną przez 2 pomija odpowiednią instrukcją. Ponadto, napisz warunek przerwania pętli, jeśli iterator będzie równy 200.

Zad 8. Napisz program, który sprawdzi czy wprowadzona przez użytkownika naturalna dodatnia liczba n , posiada co najmniej jeden dzielnik z przedziału $[0, \sqrt{n}]$. Oznacza to, że jeśli program znajdzie pierwszy dzielnik z tego przedziału to wyświetla odpowiedni komunikat i przerywa działanie pętli. Aby wykonać pierwiastkowanie wykorzystaj gotową funkcję $\text{sqrt}()$, z której można skorzystać po dołączeniu biblioteki math.h .