Laboratorium 4- Pętle **while** i **do...while**

Pętle while oraz do…while działają podobnie do pętli for – wykonują iteracyjnie fragment kodu zawarty w ciele pętli (pomiędzy nawiasami klamrowymi), sprawdzając po każdym wykonaniu, czy warunek umieszczony w nawiasach okrągłych jest prawdziwy czy fałszywy (wartość logiczna wyrażenia przyjmuje wartości TRUE lub FALSE). Składnia obu pętli przedstawiona jest na obrazkach poniżej:

```
while (warunek) {
    // Ciało pętli
}
```

Figure 1: Składnia pętli while

```
do {
    // Ciało pętli
} while (warunek);
```

Figure 2: Składnia pętli do...while

Jaka jest różnica w działaniu pętli while oraz do...while?

1. Pętla while wykona się tylko wtedy, jeżeli warunek logiczny przyjmuje wartość TRUE od samego początku wykonania algorytmu. Jeżeli warunek ma wartość logiczną TRUE, wykonuje się działanie określone w ciele pętli, a następnie warunek sprawdzany jest ponownie. Jeżeli warunek przyjmie wartość FALSE, działanie pętli zostaje zakończone.
2. Pętla do...while ZAWSZE wykona się minimum jeden raz, ponieważ wartość logiczna warunku sprawdzana jest dopiero po pierwszym wykonaniu ciała pętli. Jeżeli warunek ma wartość TRUE, ciało pętli jest wykonywane ponownie, jeżeli warunek ma wartość FALSE, działanie pętli jest przerywane.

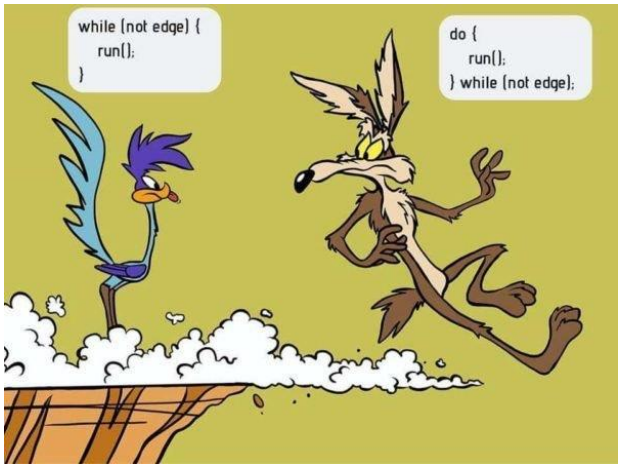

Figure 3: Ilustracja różnicy pomiędzy pętlami while oraz do...while

Przykład 1: W pierwszym przykładzie zapoznamy się z działaniem pętli while

```c
#include <stdio.h>

int main() {
    int licznik;
    /* Pytamy użytkownika o wartość początkową licznika */
    printf("Podaj wartość poczatkowa licznika: ");
    scanf("%d", &licznik);
    printf("Uwaga, zaraz sprawdze czy wejde do petli...\n");
    /* Sprawdzamy warunek PRZED wejściem do pętli */
    while (licznik < 10) {
        printf("Wartość licznika wynosi %d\n", licznik);
        licznik++; // Ręcznie zwiększamy licznik o 1
    } /* Sprawdzamy warunek licznik < 10 */
    /* Jeżeli licznik >= 10 to wychodzimy z pętli */
    /* Kończymy program po opuszczeniu pętli */
    printf("Tu koncze program.");
    return 0;
}
```

Po skompilowaniu programu, przetestuj go podając wartości początkowe licznika:

a) 1
b) 7
c) 120

Ile razy wykonała się pętla w każdym z przypadków?

Przykład 2: W drugim przykładzie sprawdzimy jak działa pętla do...while

```c
#include <stdio.h>

int main() {
    int licznik;
    /* Pytamy użytkownika o wartość początkową licznika */
    printf("Podaj wartość poczatkowa licznika: ");
    scanf("%d", &licznik);
    printf("Uwaga, zaraz wejde do petli...\n");
    /* Wchodzimy do pętli (na pewno przynajmniej raz!) */
    do {
        printf("Wartość licznika wynosi %d\n", licznik);
        licznik++; // Ręcznie zwiększamy licznik o 1
    } while (licznik < 10); /* Dopiero tu sprawdzamy warunek na wartość licznika*/
    /* Kończymy program po opuszczeniu pętli */
    printf("Tu koncze program.");
    return 0;
}
```

Po skompilowaniu programu, przetestuj go podając wartości początkowe licznika:

a) 0
b) 5
c) 20

Ile razy wykonała się pętla w każdym z przypadków?

Przykład 3:

W trzecim przykładzie napiszemy prosty program, który oblicza największy wspólny dzielnik dwóch liczb całkowitych NWD(a,b). Chcąc obliczyć NWD(55,15) wykonujemy następujące kroki:

1. Wyznaczamy resztę z dzielenia 55 przez 15: 55=15*3 + 10
2. Dzielną zastępujemy dzielnikiem a dzielnik zastępujemy otrzymaną resztą i ponawiamy krok 1: 15=10*1 + 5
3. Zapętlamy krok 1 oraz 2 aż do momentu, gdy otrzymana reszta wyniesie 0.

   10=5*2 + 0

4. Ostatnia niezerowa reszta (w naszym przypadku 5) jest NWD(55,15)

Poniżej zamieszczony został kod realizujący powyższy algorytm z wykorzystaniem pętli while:

```c
/* Algorytm Euklidesa */

#include <stdio.h>

int main() {
    int a, b, reszta;
    /* Pytamy użytkownika o dane wejściowe */
    printf("Podaj pierwsza liczbe: ");
    scanf("%d", &a);
    printf("Podaj druga liczbe: ");
    scanf("%d", &b);
    /* Sprawdzamy, czy użytkownik nie podał zera */
    if (a == 0 || b == 0) {
        printf("Podaj liczby niezerowe!");
        /* Jeżeli użytkownik podał zero, kończymy program -
        wychodzimy z funkcji main() poleceniem return 1; */
        return 1;
    }
    /* WYKONUJEMY ALGORYTM EUKLIDESA! */
    while(b != 0) {
        reszta = a % b;
        a = b;
        b = reszta;
    }
    /* Wyświetlamy wynik */
    printf("NWD podanych liczb wynosi %d", a);

    return 0;
}
```

UWAGI:

1. Pętle while oraz do...while można przerywać poleceniem break, podobnie jak pętle for.
2. W przeciwieństwie do pętli for, gdzie inkrementacja zmiennej kontrolnej odbywała się automatycznie zgodnie ze składnią for(int i = 0; i < 10; i++), w pętli while i do...while sami musimy zadbać o aktualizację wartości zmiennej znajdującej się w warunku, jak w przykładzie 1:

```
while (licznik < 10) {
    printf("Wartość licznika wynosi %d\n", licznik);
    licznik++; // Ręcznie zwiększamy licznik o 1
} /* Sprawdzamy warunek licznik < 10 */
```

ZADANIA DO SAMODZIELNEGO WYKONANIA:

Zadanie 1: (2p)

Jeżeli liczba a jest kwadratem liczby naturalnej n, to a = n^2 = 1 + 3 + 5 + ... + (2n-1). Zatem, odejmując od a kolejno liczby nieparzyste, zaczynając od 1, w pewnym momencie otrzymamy wartość 0, np.:

16 – 1 = 15

15 – 3 = 12

12 – 5 = 7

7 – 7 = 0

Ponieważ 16 = 4^2. W przypadku liczby nie będącej kwadratem liczby naturalnej, ominiemy zero i otrzymamy liczbę ujemną.

Napisać program, który zapyta użytkownika o liczbę całkowitą dodatnią, a następnie sprawdzi, czy podana liczba jest kwadratem liczby naturalnej (jeśli tak, wyświetli tę liczbę). Wykorzystać do tego celu pętlę while lub do...while (która z pętli będzie lepiej pasowała do tego zadania?)

Proszę sprawdzić, czy użytkownik podał poprawne dane wejściowe, skomentować kod oraz zadbać o ‘coding style’.

Zadanie 2: (3p)

Poprosić użytkownika o podanie liczby naturalnej. Obliczyć odwrotność podanej liczby (jakiego typu zmiennej należy użyć?), a następnie sprawdzić, dla jakiej liczby naturalnej n, (1/2)^n będzie mniejsze od otrzymanej odwrotności, zgodnie z następującym przykładem:

1. Użytkownik podaje a=10
2. Odwrotność wynosi b = 1/a = 1/10
3. (1/2)^1 = 1/2 > 1/10
4. (1/2)^2 = 1/4 > 1/10
5. (1/2)^3 = 1/8 > 1/10
6. (1/2)^4 = 1/16 < 1/10, zatem odpowiedź wynosi n=4 – wyświetlamy na ekranie.

Proszę użyć pętli while lub do...while. Proszę dodatkowo zadbać o to, by pętla nie wykonała się więcej niż 20 razy (po 20 iteracjach przerwać działanie pętli i wyświetlić komunikat, że n>20). Jaką liczbę musi podać użytkownik, aby pętla przerwała działanie z powodu przekroczenia limitu 20 iteracji?

Sprawdzić, czy użytkownik podał poprawny parametr wejściowy, dodać komentarze do kodu oraz zadbać o ‘coding style’.