C - 3 - Pętle

3.1 Pętla *for*

Większość algorytmów, które będziemy omawiać na zajęciach, będzie wymagało wielokrotnego powtarzania tych samych czynności dla różnych danych wejściowych. Nazywa się to mądrze *iteracją*. Najprostszym sposobem iterowania jest pętla **for** (ang. „dla"). Przykład:

```
int i;
for(i=0 ; i<10 ; i++){
        printf("Kolejna liczba: %d\n",i);
}
```

i będzie przebiegało pętlę, więc trzeba je zadeklarować. Bardzo częstym błędem jest zapominanie o tej deklaracji.

W forze używamy ŚREDNIKÓW, ale studenci uwielbiają pisać tam przecinki :(

Kod ujęty w klamerki zwany jest ciałem pętli.

Powyższa pętla wypisuje na ekranie napisy od *„Kolejna liczba: 0"* do *„Kolejna liczba: 9"*. **For**a otaczamy klamerkami – można je pominąć, jeżeli w środku jest tylko jedna instrukcja (tak jest w powyższym przykładzie), ale nie jest to do końca bezpieczne – podobnie jak to było w **if**ie, programista może kiedyś dopisać parę linijek i zapomnieć o klamerkach – program nie będzie wówczas poprawnie działał.

Powyższego **for**a można zapisać krócej wrzucając deklarację zmiennej **i** do nagłówka **for**a:

```
for(int i=0 ; i<10 ; i++){
        printf("Kolejna liczba: %d\n",i);
}
```

Jeżeli w następnym forze chcesz skorzystać ze zmiennej i, to nie wolno drugi raz pisać for(int i ... gdyż oznaczałoby to ponowną deklarację zmiennej.

Zadanie 0. Sprawdź, że jeżeli zamiast **i++** wpiszesz **i=i+1** lub **i+=1**, to program zadziała tak samo dobrze.

Zadanie 1. W pętli od 1 do (sprawdź eksperymentalnie dokąd) wypisuj na ekranie gwiazdki, ale bez przechodzenia do nowej linii – zbadaj ile znaków szerokości ma czarne okienko.
Następnie wykonaj ten sam eksperyment w pionie, a więc przechodząc do nowej linii po każdej gwiazdce.

Nie jesteśmy skazani na zwiększanie **i** tylko o jeden. Zmienna też nie musi wcale nazywać się **i**.

Zadanie 2. W całym programie zmień nazwę zmiennej **i** na jakąś inną, np. **zm**. Zamiast **i++** wpisz np. **zm=zm+3**, zmień wyświetlany komunikat na: „Kolejna liczba podzielna przez 3 wynosi: ".

Przykład **for**a zawierającego więcej niż jedną instrukcję:

```
int sum = 0;
int sil = 1;
for(int i=1 ; i<=10 ; i++){
        sum = sum + i;
        sil = sil * i;
        }
printf("Suma od 1 do 10 wynosi: %d \n Silnia z 10 wynosi: %d",sum,sil);
```

Zadanie 3. Napisz program, w którym użytkownik poda liczbę całkowitą **n**, zaś program obliczy sumę liczb od 1 do **n** za pomocą pętli **for**. Jeżeli złośliwy użytkownik podał liczbę ujemną, to pomnóż ją razy -1.

Zadanie 4. Napisz program, w którym użytkownik poda dwie liczby całkowite: **n** oraz **k**, zaś program obliczy sumę liczb od **k** do **n** za pomocą pętli **for**. Jeżeli użytkownik podał **k** większe od **n**, to zamień je miejscami.

Zawsze staraj się obsługiwać błędy powodowane przez złośliwych użytkowników. Taka cecha programu nazywa się *idiotoodpornością*.

Zadanie 5. Napisz program, w którym użytkownik poda liczbę całkowitą **n**, zaś program obliczy silnię tej liczby.

Zadanie 6. Napisz program, w którym użytkownik poda dwie liczby całkowite: **n** oraz **k**, zaś program obliczy symbol Newtona $\binom{n}{k}$ - przydadzą się trzy osobne pętle **for** dla obliczenia **n!**, **k!** oraz **(n-k)!**. Czy wystarczą tylko dwie pętle **for**, jeżeli odpowiednio skrócisz wzór? Napisz też taką wersję.

Zmienna **for**owa nie musi wcale być typu **int**. Przykład:

```
for(double i=0 ; i<1 ; i=i+0.2){
        printf("Kolejny ulamek: %1.2lf\n",i);
}
```

Zadanie 7. Napisz program, w którym użytkownik poda liczbę zmiennoprzecinkową **n**, zaś program obliczy sumę liczb od 1 do **n** (skok co **0.2**) za pomocą pętli **for**. Jakiego typu będzie zmienna przechowująca sumę?

Przykład programu zamieniającego stopnie Fahrenheita na stopnie Celsjusza:

```
for (int fahr=-120 ; fahr<=300 ; fahr=fahr+20){
        printf( "%4d %6.1f\n",fahr,(5.0/9.0)*(fahr-32));
}
```

Zadanie 8. Uruchom powyższy program, zauważ, że dzięki **%4d** liczby po lewej stronie ładnie się wyrównały w pionie.

Wyniki obliczeń wyglądają o wiele lepiej otoczone tabelką – na razie będzie to tabelka złożona z **+ -** oraz **|**:

```
printf("+-----+-------+\n");
for (int fahr=-120 ; fahr<=300 ; fahr=fahr+20){
        printf("|%4d | %6.1f|\n",fahr,(5.0/9.0)*(fahr-32));
}
printf("+-----+-------+\n");
```

Zadanie 9. Znajdź w Internecie przelicznik na stopnie Kelvina, wydrukuj je jako trzecią kolumnę tabelki.

Poniższy kod drukuje tabelę kodów ASCII w czterech kolumnach oddzielonych tabulacjami:

```
for(int i=0;i<64;i++){
        printf("%4d %c\t%4d %c\t%4d %c\t%4d %c\n",i,i,64+i,64+i,128+i,128+i,192+i,192+i);
}
```

Zadanie 10. Uruchom powyższy program, zauważ, że kolumny są krzywe. Znajdź w internecie tabelę kodów ASCII i sprawdź, jakie znaki mają kod 9 i 10.
Dodaj w programie jakiegoś **if**a wewnątrz **for**a, który będzie sprawdzał, czy **i** jest równe 9 lub 10. Jeżeli jest równe, to należy wypisać podobnego **printf**a, ale „nie rozjechanego". Jeżeli **i** jest różne od 9 i 10, to należy użyć **printf**a z kodu powyżej.

Zadanie 11. Przerób poprzednie zadanie – dodaj tabelkę złożoną z **+ - |** dookoła kolumn z kodami ASCII.

Zadanie 12. Zauważ, że w wyświetlonych kodach ASCII znajdują się kody o wiele ładniejszych fragmentów tabelek, np. 203, 205, 186, 187, itd ... Użyj tych znaków w zadaniu **11** do zrobienia tabelki (zamiast **+ - |**).

Zadanie 13. Napisz program, który oblicza pierwsze 100 liczb Fibonacciego. Przydadzą się dwie zmienne pomocnicze, które będą przechowywać dwa poprzednie wyrazy ciągu.

Zadanie 14. Przerób program z poprzedniego ćwiczenia – oprócz obliczania liczb Fibonacciego sprawdzaj też, czy iloraz dwóch kolejnych liczb Fibonacciego dąży do złotego środka. Dla lepszych efektów zwiększ pętlę do 1000. Nie musisz wypisywać liczb Fibonacciego, tylko ich iloraz.

Zadanie 15. Napisz program, w którym użytkownik podaje liczbę całkowitą dodatnią **n**, zaś program znajdzie wszystkie dzielniki **n**. Ile razy wystarczy wykonywać instrukcję w pętli **for**? Następnie przerób program tak, by zliczał liczbę dzielników do jakiejś zmiennej i wyświetlił ją na końcu.

Zadanie 16. Uogólnij poprzedni program tak, by odpowiadał tylko na pytanie, czy liczba jest pierwsza, czy złożona. Ile razy wystarczy wykonywać instrukcję w pętli **for**? I jak nazywała się biblioteka, w której znajduje się funkcja obliczająca pierwiastek?

(*) W jaki sposób można użyć znaków **/** oraz ***** zamiast znaku **%** (modulo) żeby zbadać podzielność?

W pętli **for** nie trzeba koniecznie zwiększać wartości jakiejś zmiennej. Przykład:

```
for(int i=9 ; i>=0 ; i--){
        printf("Kolejna liczba, tym razem malejaco: %d\n",i);
}
```

Zadanie 17. Zauważ, że tutaj **i** zaczyna się od 9 i kończy się na zerze. Częstym błędem jest pisanie w środku **i<=0** – sprawdź, że program nie zadziała. Częstym błędem jest pisanie **i++** w przypadku, gdy chcemy zmniejszać zmienną – sprawdź, że program się zawiesi (logiczne – skoro zwiększamy wartość zmiennej, to nie dojdziemy do zera – chyba że w końcu przekroczymy zakres i popadniemy w liczby ujemne).

Czasem przez pomyłkę zdarzy się napisać **for**a, który nigdy się nie wykona, np:

```
for(int i=0 ; i>0 ; i--){
        printf("Kolejna liczba, tym razem malejaco: %d\n",i);
}
```

Nawiasem mówiąc, operację ++ nazywamy *„inkrementacją"*, zaś operację - - *„dekrementacją"*.

Zadanie 18. Napisz jakiegoś **for**a, który się nie wykona i który zawiera inkrementację zmiennej **for**a.

Zadanie 19. Wykonaj ponownie zadania: **2**, **9** oraz **15** korzystając z dekrementacji zmiennej **for**a.

Podobnie jak w przypadku **if**, częstym błędem jest pisanie średnika zaraz po **for**ze, np.:

```
for(int i=9 ; i>=0 ; i--);
        printf("Kolejna liczba, tym razem malejaco: %d\n",i);
```

Po polsku: **dla** i **równego** od 9 do 0 nic nie rób.

Co ciekawsze, ten błąd nie zdarza się studentom piszącym poprawnie klamerki :)

3.2 *x++* oraz *++x*

Wykonaj eksperyment – najpierw wpisz w programie pętlę

```
for(int i=0;i<10;i++){
        printf("%d",i);
}
```

A następnie:

```
for(int i=0;i<10;++i){
        printf("%d",i);
}
```

Jaka jest różnica w ich działaniu? Operacja **++i** powoduje, że zmienna **i** jest zwiększana jeszcze przed wykonaniem instrukcji w ciele pętli **for**, właśnie dlatego najpierw widzimy jedynkę, a nie zero.

++i jest przykładem inkrementacji prefiksowej, zaś **i++** - postfiksowej.

Zadanie 20. Wykonaj ponownie zadania: **4**, **5**, **6** oraz **13** korzystając z inkrementacji prefiksowej.

Zadanie 21. Wpisz w programie pętle korzystające z dekrementacji prefiksowej i postfiksowej, zauważ, że wynik ich działania też się różni.

3.3 *for* w nieco zagmatwanej formie

Podobnie jak inne elementy języka C, **for**a można skomplikować. Sprawdź, jak zadziała coś takiego:

```
int x=0;
for( ; x<10; x++){
        printf("A kuku po raz %d\n",x);
}
```

lub coś takiego:

```
int x=0;
for( ; x<10; ){
        printf("A kuku po raz %d\n",x);
        x++;
}
```

For może nawet nie mieć zmiennej, która maleje lub rośnie. Przykład:

```
char c='n';
for( ; c!='t'; ){
        printf("No nacisnij w koncu 't' !!! ");
        scanf("%c",&c);
}
```

Bardzo dobry przykład konfliktu nazw: pierwsze **c** oznacza, że chcemy pobrać wartość **char**ową; drugie **c** jest nazwą zmiennej.

Uruchom powyższy kod, najprawdopodobniej polecenie naciśnięcia litery 't' będzie pokazywać się dwukrotnie (oprócz pierwszego razu). Dzieje się tak dlatego, że naciskając literę naciskasz też enter – a więc dwa przyciski. Aby to poprawić zaincluduj w odpowiednim miejscu bibliotekę **conio.h.** Zaraz po **scanf**ie wpisz **getchar();** i przetestuj działanie programu. Funkcja **getchar** łapie znak ze standardowego wyjścia i od razu go stamtad usuwa. Różni się ona od funkcji **getch** poznanej we wprowadzeniu do języka C, ponieważ **getch** wstrzymuje działanie programu aż do naciśnięcia jakiegoś klawisza, a **getchar** nie.

Zadanie 22. Sprawdź, że jeżeli zamienisz w tym przykładzie **getchar** na **getch**, to efekty nie będą zbyt dobre.

Można też ten fragment napisać w inny sposób:

```
char c='n';
for( ; c!='t'; ){
        printf("No nacisnij w koncu t !!! ");
        c = getchar();
        getchar();
}
```

Zauważ, że **getchar** może występować jako samotna instrukcja, ale może zwracać wartość do jakiejś zmiennej!

Drugie **getchar** służy tylko po to, by zlapać entera. Nie zapisujemy nigdzie tego entera, no bo po co ...

3.4 Zagnieżdżanie pętli *for*

Pętle **for** można zagnieżdżać – sprawdź, co robi poniższy kod:

```
for(int i=0;i<5;i++){
        for(int j=0;j<13;j++){
                printf("X");
        }
        printf("\n");
}
```

Zadanie 23. Przerób powyższy program tak, by wypisywał tabliczkę mnożenia do stu w 10 wierszach i kolumnach – **i** oraz **j** mają się zmieniać od **0** do **9**. Wystarczy, jeżeli na każdy iloczyn przeznaczysz 3 miejsca (czyli **%3d**).

Najlepszym ćwiczeniem dla zagnieżdżania pętli jest rysowanie wzorków. Przykład:

```
for(int i=1;i<=10;i++){
        for(int j=1;j<=10;j++){
                if(i<j) printf("0");
                else if(i==j) printf("+");
                else printf("1");
        }
        printf("\n");
}
```

Spróbuj przerobić oba przykłady tak, by i oraz j zmieniały się od 0 do 9.

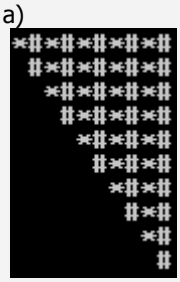
Warto nabrać nawyku liczenia od zera – tablice będą się zaczynały od zera, nie od jedynki.

Inny przykład:

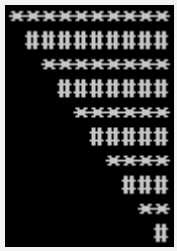
```
for(int i=1;i<=10;i++){
        for(int j=1;j<=10;j++){
                if(i<=j) printf("%c",64+i);
                else printf(" ");
        }
        printf("\n");
}
```

Zadanie 24. Napisz programy rysujące poniższe wzorki:

a)



b)



c)

d)

e)

f)

Zadanie 25. Skorzystaj z zadania **15**, by wyznaczyć dzielniki wszystkich liczb od 1 do **n** – użytkownik podaje liczbę **n** z klawiatury.

Zadanie 26. Skorzystaj z zadania **6**, by wyznaczyć wszystkie współczynniki wielomianu $(x+1)^n$, gdzie **n** jest liczbą podaną przez użytkownika.

Zadanie 27*. Przerób poprzednie zadanie – za pomocą trzech zagnieżdżonych pętli wypisz pierwsze **m** pięter trójkąta Pascala, gdzie **m** jest liczbą podaną przez użytkownika.

3.5 Instrukcje *break* oraz *continue*

Instrukcja **break** służy do wyskoczenia z pętli – przydaje się, gdy stwierdzamy że obliczyliśmy/wyznaczyliśmy już daną wartość i nie chce nam się czekać, aż pętla skończy działanie. Przykład:

```
char odp='n';
for(;;){
        printf("Jestem pętlą nieskończoną, czy chcesz już wyskoczyć?\n");
        scanf("%c",&odp);
        if(odp=='t') break;
}
```

Tak wygląda nieskończony for.

Zadanie 28. Popraw powyższy fragment programu – zawartość **printf**a znowu drukuje się podwójnie.

Instrukcja **continue** jest o wiele rzadziej używana. Po jej wywołaniu w ciele pętli program ignoruje dalsze instrukcje w pętli i wskakuje znowu do pierwszej instrukcji pętli. Załóżmy, że chcemy sprawdzać pierwszość liczb od 3 do 100 – nie ma sensu sprawdzać liczb parzystych, napisalibyśmy więc coś takiego:

```
for(int i=3;i<100;i++){
        if(i%2==0){
                continue;      // po tej instrukcji i zwiększa się o jeden i znowu wchodzimy do pętli
        }
        else{
                .... (coś tam) ...   // tu sprawdzamy pierwszość liczby
        }
}
```

Zadanie 29. Napisz program rozwiązujący problem Collatza dla dużej liczby **n** podanej z klawiatury.

Przypomnienie: Weźmy dowolną liczbę naturalną c_0 (większą od 0). Jeśli jest ona parzysta, to za c_1 przyjmijmy $c_0/2$, w przeciwnym wypadku niech $c_1=3c_0+1$ co formalnie można zapisać jako:

$$c_{n+1} = \begin{cases} \frac{1}{2}c_n & \text{dla } c_n \text{ parzysta} \\ 3c_n + 1 & \text{dla } c_n \text{ nieparzysta} \end{cases}$$

Hipoteza Collatza stwierdza, że niezależnie od jakiej liczby wystartujemy, w końcu dojdziemy do liczby 1.

Skorzystaj z nieskończonej pętli **for(;;)**. Ponieważ w ogólnym przypadku nie wiadomo, czy pętla się zatrzyma, pytaj co 20 kroków, czy już przerwać (musisz jakoś zliczać te kroki). Jeżeli użytkownik się zgodzi, to wyskocz z pętli za pomocą instrukcji **break**.

Jeżeli mamy zagnieżdżoną pętlę, to **break** wychodzi tylko z tej pętli, w której został bezpośrednio wywołany. Sprawdź, że działanie poniższego programu nie zakończy się po użyciu instrukcji **break**:

```
for(int i=1;i<=10;i++){
        for(int j=1;j<=10;j++){
                printf("%3d ",i*j);
                if(i*j==32) break;
        }
        printf("\n");
}
```

Aby w ogóle zakończyć działanie programu po dotarciu do liczby 32, można napisać coś takiego:

```
bool chceWyskoczyc=false;
for(int i=1;i<=10;i++){
        for(int j=1;j<=10;j++){
                printf("%3d ",i*j);
                if(i*j==32){
                        chceWyskoczyc = true;
                        break;
                }
        }
        if(chceWyskoczyc==true){
                break;
        }
        printf("\n");
}
```

Zmienna pomocnicza do wyskakiwania.

Sprawdź, że jeżeli wpiszesz tę linijkę po breaku, to program nie będzie działał poprawnie.

Dopiero tu wyskakujemy z zewnętrznej pętli.

Zadanie 30. Napisz program robiący cokolwiek, lecz korzystający z trzech zagnieżdżonych pętli **for**. W danym momencie wyskocz z najbardziej wewnętrznej pętli korzystając z powyższego sposobu.

3.6 Pętla *while*

Pętla **while** (ang. „tak długo jak …") ma następującą składnię:

```
while([zachodzi warunek]){
        [instrukcje]
}
```

Przykład dla problemu Collatza:

```
int n = 1999;
while(n!=1){
        printf("%d ",n);
        if(n%2==0) n=n/2;
        else n = 3*n+1;
}
printf("%d ",n);  // <- bez tego nie wydrukowałaby się końcowa jedynka
```

Warunek w **while**u jest sprawdzany przed wykonaniem instrukcji w ciele pętli. Przy odpowiednio źle dobranych danych może się więc zdarzyć, że **while** nie wykona się ani razu.

Zadanie 31. W powyższym przykładzie dobierz wartość początkową **n** w taki sposób, by **while** nie wykonał się ani razu.

While jest bardzo wygodny do wyświetlania menu. Przykład:

```
char opcja=' ';
while(opcja!='e'){
        printf("e - koniec\n");
        printf("l - równanie liniowe\n");
        printf("k - równanie kwadratowe\n");
        printf("Co wybierasz? ");
        opcja=getchar();
        getchar();
        if(opcja =='l')
                printf("wybrałeś równanie liniowe\n");
        else if(opcja=='k')
                printf("wybrałeś równanie kwadratowe\n");
        else if(opcja!='e')
                printf("nieznana opcja\n");
        else
                printf("do widzenia :)\n");
}
```

Tu w apostrofach jest spacja. Sprawdź, czy da się przypisać do zmiennej typu char puste apostrofy.

Z **while** można też wychodzić za pomocą **break**a, np.:

```
char opcja=' ';
while(true){
        printf("e - koniec\n");
        printf("l - równanie liniowe\n");
        printf("k - równanie kwadratowe\n");
        printf("Co wybierasz? ");
        opcja=getchar();
        getchar();
        if(opcja =='l')
                printf("wybrałeś równanie liniowe\n");
        else if(opcja=='k')
                printf("wybrałeś równanie kwadratowe\n");
        else if(opcja!='e')
                printf("nieznana opcja\n");
        else{
                printf("do widzenia :)\n");
                break;
        }
}
```

while(true), czyli tak długo jak prawda jest prawdą, a więc w nieskończoność. Lub dopóki zasilacz w komputerze się nie spali :)

While można zagnieżdżać w sobie, np.:

```
char opcja=' ';
while(true){
        printf("e - koniec\n");
        printf("r - równanie\n");
        printf("Co wybierasz? ");
        opcja=getchar();
        getchar();
        if(opcja =='r'){
                char opcjaWewn=' ';
                while(true){
                        printf("  Masz do wyboru rownania:\n");
                        printf("  k - kwadratowe\n");
                        printf("  l - liniowe\n");
                        printf("  j - jakies inne\n");
                        printf("  w - chce wrocic do zewnetrznego menu !!!\n");
```

```
                        printf("  Co wybierasz? ");
                        opcjaWewn = getchar();
                        getchar();
                        if(opcjaWewn!= 'w')
                                printf("Wybrales rownanie, nie wazne jakie.\n");
                        else
                                break;
                }
        }
        else{
                printf("do widzenia :)\n");
                break;
        }
}
```

Tak jak to było w przypadku **for**a, **break** wyskakuje tylko z tej pętli, w której zostało wywołane, a nie ze wszystkich pętli na raz.

Oczywiście, można zagnieżdżać **while**a w **for**ze i **for**a w **while**u.

Zadanie 32. Przerób jeden ze wzorków w zadaniu **24** tak, by raz **while** otaczał **for**a, a raz **for** otaczał **while**a.

Zadanie 33. Napisz program wyświetlający menu, w którym użytkownik może wybrać, czy chce:
a - policzyć coś z silnią
b - policzyć coś z sumowaniem
Jeżeli wybrano **a**, to zapytaj dodatkowo, czy ma to być:
s – zwykła silnia (w takim wypadku pobierz jedną liczbę)
n – współczynnik Newtona (w takim wypadku pobierz dwie liczby)
Po dokonaniu wyboru i pobraniu danych oblicz żądaną wartość korzystając z poprzednich zadań.
Jeżeli wybrano **b**, to poproś tylko o jedną liczbę i oblicz sumę liczb całkowitych od 1 do tej liczby.

Poniższy kod oblicza podłogę z **n/2** nie korzystając z dzielenia zaokrąglanego w dół ani z funkcji z biblioteki **math.h**:

```
int n=19;
int podl=-1;
printf("podloga z (%d przez 2) ",n);
while(podl < n){
        podl++;
        n--;
}
printf("wynosi %d\n",podl);
```

Zadanie 34. Nie korzystając z dzielenia napisz program, który dla liczby **n** danej z klawiatury oblicza $\lceil n/2 \rceil$ oraz $\lfloor n/3 \rfloor$. Nie wolno też korzystać z funkcji obliczających podłogę i sufit w bibliotece **math.h**.

Poniższy kod przelicza liczby dziesiętne na binarne:

```
int n=19;
while(n>0){
        printf("%3d | %d\n",n,n%2);
        n=n/2;
}
printf("Spisz wynik od dołu :)\n");
```

Zadanie 35. Napisz program, w którym użytkownik podaje z klawiatury liczbę **n** oraz podstawę systemu, na który chce zamienić liczbę **n**. Program ma za zadanie zamienić liczbę **n** na podany system.

Zadanie 36. Zmień powyższy program tak, by przedstawiał liczbę **n** jako odpowiednią sumę. Przykładowo, dla **n=19** i systemu binarnego byłoby to: **19 = 1*1 + 1*2 + 1*16**. Ale już w systemie trójkowym będziemy mieli: **19 = 1*1 + 2*9**.

Wskazówka: Przy tego typu zadaniach często wynik jest postaci: **19 = 1*1 + 1*2 + 1*16 +** . Aby było bardziej elegancko, można tuż po wyjściu z pętli wydrukować znaczek **\b** (czyli **backspace**), który zmaże ostatniego plusa. Ewentualnie można sprawdzać **if**em, czy to już ostatni obieg pętli – jeżeli ostatni, to nie należy pisać plusa.

Zadanie 37. Napisz program, który dla liczby podanej z klawiatury (np. 1234) wyświetli napis postaci:
4 + 10*(3 + 10*(2+10*1)).

Wskazówka – warto zliczać otwierane nawiasy, by móc je potem zamknąć odpowiednią liczbę razy.

Zadanie 38. Napisz program, który w pętli **while** będzie prosił użytkownika o liczby zmiennoprzecinkowe i sumował je aż do naciśnięcia zera.

Zadanie 39. Wykonaj od nowa zadania: **13**, **15** oraz **26** korzystając tylko i wyłącznie z pętli **while**.

Zadanie 40. Napisz program robiący coś sensownego w pętli **while** i korzystający z dekrementacji zmiennej.

3.7 Pętla do-while

Pętla **do-while** jest bardzo podobna do pętli **while**:

```
do{
        [instrukcje]
}while([zachodzi warunek]);
```

WSZYSCY zapominają o tym średniku :(

Przykład problemu Collatza napisanego w inny sposób:

```
int n = 1999;
do{
        if(n>1){
                printf("%d ",n);
                if(n%2==0) n=n/2;
                else n = 3*n+1;
        }
}while(n!=1);
printf("%d ",n);
```

Zauważ, że w przeciwieństwie do pętli **while**, pętla **do-while** przynajmniej raz wykona instrukcje zawarte w ciele pętli. **breaki** i zagnieżdżenia działają dokładnie tak samo, jak w pętlach **for** i **while**.

Zadanie 41. Wykonaj od nowa zadania: **5**, **7**, **16** oraz **24a** korzystając wyłącznie z pętli **do-while**.

3.8 Najczęstsze błędy

Tradycyjnie: klamerki, średniki, cudzysłowy, apostrofy.
Również tradycyjnie, pojedyńcze **=** w **if**ach.

Pisanie przecinków zamiast średników w nagłówku **for**a.
Zapominanie o średniku po warunku na końcu pętli **do-while**.
Brak umiejętności odróżnienia x++ od ++x.
Napisanie jakiejś instrukcji bezpośrednio po **breaku** i dziwienie się, że instrukcja się nie wykonuje.

Przy zagnieżdżonych pętlach: brak umiejętności oszacowania *„co i ile razy ma robić pętla”*, co widać zwłaszcza w zadaniach typu **wzorki**.

Najbardziej irytujący błąd:
Gdy studenci mają tylko coś obliczyć, to zawsze to wypisują, bo jest to dla nich jednoznaczne z obliczeniem. Wbrew pozorom, można coś obliczyć i tego nie wyświetlać !!!

3.9 Quiz

1. Ile errorów pojawi się, gdy w nagłówku **for**a wpiszesz przecinek zamiast pierwszego średnika? ____.
   Jakie to będą błędy?
   _____
   _____
   _____
   _____
2. Ile errorów pojawi się, gdy w nagłówku **for**a wpiszesz przecinek zamiast drugiego średnika? ____.
3. Jakim znaczkiem możemy wydrukować **backspace** na ekranie? ___ Co się stanie ze znaczkiem napisanym przed **backspace**m? _____.
4. Jaki znak ma kod ASCII równy 9? _____. A jak to jest dla 10? _____.

5. Podaj oryginalny i pomysłowy powód (a więc inny niż **menu**) zastosowania pętli nieskończonej:
   _____
   _____
   _____

6. Czym się różni **getch** od **getchar**?
   _____
   _____

7. Podaj przykład, w którym chętniej zastosowałbyś **getch** zamiast **getchar**:
   _____
   _____

8. Wypisz trzy najczęstsze błędy, które popełniasz podczas pisania programów:
   _____
   _____
   _____

9. Wiedząc, że **x**, **y** oraz **z** wynoszą **9**, orzeknij, jaka będzie wartość **y** po wykonaniu instrukcji:

a) **y += x++ + --z;** _____
b) **y *= ++x * z--;** _____
c) **y = x++ * --z;** _____
d) **++y += x++ + --z;** _____