Лабораторна робота №9
Алгоритми циклічної структури

Мета роботи: отримання практичних навичок в побудови алгоритмів циклічної структури засобами мови *C*.

Зміст:

Короткі теоретичні відомості

Оператори циклів

При програмуванні часто виникає необхідність повторювати однотипні дії кілька разів. Для реалізації цього завдання використовують оператори циклів. У мові *C* цикли організовуються за допомогою операторів **`for`**, **`while`** та **`do-while`**.

Оператор циклу `for`

Оператор циклу **`for`,** зазвичай, використовується в тих випадках, коли відомо скільки разів треба повторити певну дію.

Тіло оператора **`for`** виконується нуль чи більше разів до тих пір, поки умова (необов'язкова) не стане хибною. Іншими словами, цикл повторюється, поки умова істинна. Необов'язкові вирази можуть використовуватися в операторі **`for`** для ініціалізації та зміни величин в процесі виконання оператора.

Синтаксис:

```
for (вираз-ініціалізації; вираз-умова; вираз-повторення)
   оператор
```

`Вираз-ініціалізації` , зазвичай, (але не обов'язково!) використовується для встановлення початкового значення змінних, що керують циклом.

`Вираз-умова` визначає умову, за якої тіло циклу буде виконуватися.

`Вираз-повторення` визначає як модифікується зміна чи змінні після кожного виконання тіла циклу.

Виконання циклу відбувається наступним чином:

1. Обчислюється, якщо присутній, *`вираз-ініціалізації`*. Так задаються початкові умови циклу. Обмеження на тип виразу ініціалізації не накладаються.
2. Обчислюється, якщо присутній, *`вираз-умова`*. Цей вираз має бути арифметичного типу. Він обчислюється перед виконанням кожної ітерації. Можливі три варіанти:
    a. Якщо значення *`вираз-умова`* істинне (ненульове), виконується *`оператор`*, потім обчислюється, якщо присутній, *`вираз-повторення`*. Цей вираз обчислюється після кожної ітерації.

Обмеження на тип цього виразу не накладаються. Потім процес повторюється з обчислення *виразу-умови*.

b. Якщо *вираз-умова* опущено, його значення вважається істинним і виконання триває подібно попередньому випадку. Виконання оператора **for** з опущеним *виразом-умови* переривається лише за допомогою операторів **break** або **return** у тілі циклу або переходом за межі циклу за допомогою оператора **goto**.

c. Якщо значення *вираз-умова* хибне (нульове) виконання оператора **for** припиняється і керування передається наступному оператору в програмі. Якщо значення *вираз-умова* хибне від початку, тіло циклу не виконається жодного разу.

Виконання оператора **for** також переривається, якщо оператори **break**, **goto** або **return** виконуються в тілі циклу. Оператор **continue** в тілі циклу приводить до переходу до виконання умови повторення (частина складеного оператора, що залишилась, не виконується). При виконанні оператора **break** в тілі циклу вираз повторення не обчислюється.

Оператор

```
for ( ; ; )
   оператор
```

є типовим способом створення нескінченного циклу. Його виконання може бути перерване тільки явним виходом з допомогою операторів **break**, **goto**, **return.**

Приклад 1

Як правило, цикл використовується для повторення деякого оператора задане число разів. Наступний код може бути використаний для обчислення факторіала числа 10:

```
for (a = 1, i = 1; i <= 10; i++)
   a *= i;
```

Спочатку змінним *a* та *i* присвоюється значення 1. Після цього обчислюється значення *виразу-умови* *i <= 10*. Оскільки воно істинне, то виконується тіло циклу, тобто обчислюється *a *= i;*, а потім обчислюється *вираз-повторення* *i++*, тобто значення змінної *i* збільшується на одиницю. Після цього всі операції повторюються починаючи з перевірки *виразу-умови*. Очевидно, що тіло циклу повториться 10 разів зі значеннями змінної *i*, рівними 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. На останній ітерації значення змінної *i* досягне 11 і виконання циклу буде перервано.

Приклад 2

Можна застосовувати операцію зменшення для відліку у порядку зменшення замість відліку в порядку зростання. Наприклад, наступний код

```
for (n = 10; n > 0; n--)
   printf("%d ", n);
```

друкує на екрані

```
10 9 8 7 6 5 4 3 2 1
```

Приклад 3

За допомогою циклу можна маніпулювати не тільки цілочисельними змінними, а й дійсними. Крім того, можна зробити так, щоб значення деякої величини змінювалося не лінійно, а обчислювалося за довільною формулою:

```
float x;

```

```
for (x = 10.0; x < 15.0; x *= 1.3)
   printf("x = %f\n", x);
```

У цьому фрагменті програми значення змінної *x* множиться на 1.3 на кожному кроці циклу до тих пір, поки її значення не перевищує 15. При виконання програми на екран буде виведено

```
x = 10.000000
x = 13.000000
```

Приклад 4

Вираз-ініціалізації не обов'язково має ініціалізувати змінну. Замість цього, наприклад, там міг би стояти оператор *printf()*. Необхідно пам'ятати лише, що вираз ініціалізації обчислюється тільки один раз перед тим, як інші частини циклу почнуть виконуватися. Наприклад, наступний фрагмент програми

```
int num = 1;

for(printf("Begin loop: \n"); num <= 6; num++)
   printf("%d ", num);
```

друкує на екрані

```
Begin loop:
1 2 3 4 5 6
```

Оператор циклу з передумовою `while`

Оператор **`while`** дозволяє повторювати *`оператор`* доти, поки заданий *`вираз`* істинний. Коли значення виразу стає хибним, виконання циклу припиняється.

Синтаксис:

```
while ( вираз )
   оператор
```

`Вираз` повинен бути арифметичного типу. Виконання відбувається наступним чином:

1. Обчислюється *`вираз`*.
2. Якщо *`вираз`* від початку помилковий, тіло циклу взагалі не виконується і керування передається наступному оператору в програмі. Якщо *`вираз`* істинний (ненульовий), виконується тіло циклу (*`оператор`*) і процес повторюється, починаючи з кроку 1.

Виконання оператора **`while`** також переривається, якщо оператори **`break`**, **`goto`** або **`return`** виконуються в тілі циклу. Використовуйте оператор **`continue`** щоб перейти до наступного ітерації без виходу з циклу.

Тілом циклу може бути будь-який оператор, у тому числі порожній або складений.

Оператор циклу

```
for (вираз1; вираз2; вираз3)
   тіло ;
```

може бути замінений еквівалентним оператором **`while`**:

```
вираз1;
while (вираз2)
```

```
{
   тіло;
   вираз3;
}
```

Як і при виконанні оператора **`for`**, в операторі **`while`** перевірка умови відбувається ***до*** виконання тіла циклу. Якщо умова помилкова від початку, тіло циклу не виконується жодного разу.

Приклад

```
while ( i >= 0 )
{
   string1[i] = string2[i];
   i--;
}
```

У цьому прикладі символи з рядка *`string2`* копіюються в рядок *`string1`*. Якщо *`i`* більше або дорівнює нулю, символ *`string2[i]`* присвоюється *`string1[i]`* та *`i`* зменшується на 1. Коли *`i`* досягає 0, виконання оператора **`while`** припиняється.

Оператор циклу з післяумовою `do-while`

Оператор **`do-while`** виконує оператор або складений оператор доти, поки задана умова істинна. Виконання циклу припиняється, коли умова стає хибною.

Синтаксис

```
do оператор while ( вираз ) ;
```

Вираз в циклі **do-while** обчислюється ***після*** виконання тіла циклу. Таким чином, тіло циклу ***завжди виконується хоча б один раз***.

Вираз повинен бути арифметичного типу. Виконання відбувається наступним чином:

1. Виконується тіло циклу (*оператор*).
2. Потім обчислюється значення *виразу*. Якщо *вираз* хибний (нульовий), виконання оператора **do-while** переривається і керування передається наступному оператору програми. Якщо *вираз* істинний (ненульовий), процес повторюється, починаючи з кроку 1.

Виконання оператора **do-while** також переривається, якщо оператори **break**, **goto** або **return** виконуються в тілі циклу.

Приклад

```
do
{
    y = f( x );
    x--;
} while ( x > 0 );
```

У цьому прикладі оператори *y = f(x);* та *x--;* виконуються незалежно від початкового значення *x*. Потім обчислюється значення виразу *x>0*. Якщо *x* більше 0, тіло циклу виконується ще раз і повторно обчислюється значення виразу *x>0*. Виконання тіла циклу повторюється до тих пір, поки *x* більше 0. Виконання циклу припиняється, коли *x* стає нулем або негативним. Тіло циклу завжди виконується хоча б один раз.

Оператори **for**, **while** та **do-while** можуть бути вкладеними.

Оператор ***break***

Оператор **break** забезпечує припинення виконання самого внутрішнього з операторів які його осягають: **switch**, **for**, **while**, **do-while**. Після виконання оператора **break** керування передається оператору, наступному за перерваним.

Оператор ***continue***

Починає нову ітерацію циклу. При виконанні оператора **continue** поточна ітерація переривається і управління передається в початок тіла найближчого з циклів **do**, **for** або **while**. При цьому оператори тіла циклу, що знаходяться нижче **continue**, на поточній ітерації не виконуються.

Приклад

```
while ( i-- > 0 )
{
    x = f( i );
    if ( x == 1 )
        continue;
    y += x * x;
}
```

У цьому прикладі тіло циклу виконується, поки *i* більше 0. Спочатку обчислюється значення функції *f(i)* і присвоюється змінній *x*; потім, якщо *x* дорівнює 1, виконується оператор **continue**. Залишок тіла циклу ігнорується, а виконання продовжується з початку перевіркою виразу *i-- > 0*. Таким чином, до суми *y* не потрапляє значення *x*, рівне 1.

Оператор **continue**, як і оператор **break**, перериває самий внутрішній з циклів.

Зауваження по стилю

При обчисленнях за допомогою циклів, у тому числі при обчисленні сум рядів, важливо реалізувати обчислення раціональним чином. Найчастіше наступний член ряду відрізняється від попереднього на деяку величину, що просто обчислюється. У цьому випадку доцільно зберігати поточне значення члена ряду для подальших обчислень.

Наприклад, якщо потрібно обчислити значення 2^n, де $n = 0, 1, 2, 3, \ldots$, то перед початком виконання циклу можна завести змінну $pow\_2 = 1$. І тепер якщо в тілі циклу на кожній ітерації виконувати операцію $pow\_2\ *= 2$, то в цій змінній буде зберігатися значення 2^n.

Обчислення абсолютної та відносної похибки

Абсолютною похибкою обчислення певного значення $\tilde{x}$ називають різницю між цим числом і його точним значенням x^*:

$$\Delta x = \left|\tilde{x} - x^*\right|.$$

Відносною похибкою називають величину

$$\delta x = \frac{\Delta x}{\left|x^*\right|}.$$

Позаяк на практиці точне значення не відоме (інакше не було б потреби обчислювати наближене), то вдаються до простіших, хоч і неточних оцінок. Наприклад, для збіжних рядів абсолютною похибкою можна вважати абсолютне значення поточного члена, бо саме на його величину буде уточнено значення суми:

$$\Delta S = \left|a_n\right|.$$

Подібним чином відносну похибку обчислення суми ряду на *n*-му кроці можна увести як

$$\delta S_n = \frac{\left|a_n\right|}{\left|S_n\right|}.$$

Робоче завдання

1. Обчислити суму ряду

$$S_1 = \sum_{n=1}^{k} a_n$$

за його першим k членами. Вивести на екран значення S_1.

2. Обчислити суму ряду

$$S_2 = \sum_{n=1}^{N(\varepsilon)} a_n$$

так, щоб похибка обчислення суми не перевищувала заданої відносної похибки ε. Для цього можна перевіряти співвідношення абсолютних величин суми ряду та поточного члена.

Вивести на екран результати розрахунків, які включають:

- ✓ Номер ітерації;
- ✓ Величину поточного члена ряду a_n;
- ✓ Величину накопиченої суми ряду S_1;
- ✓ досягнуту на поточній ітерації похибку.

Номер варіанта	k	X	ε	a_n
1	7	$\frac{\pi}{8}$	10^{-4}	$(-1)^n \frac{1}{2^n + nx}$
2	14	$\frac{\pi}{16}$	10^{-6}	$(-1)^n \frac{n+x}{n^2 + 2^n}$
3	8	2.05	10^{-3}	$\frac{x^n}{n!}$
4	5	$\frac{\pi}{2}$	10^{-4}	$(-1)^n \frac{2^n}{nx + 5^n}$
5	9	1.5	10^{-2}	$(-1)^n \frac{n+1}{(nx)^2 + 1}$

Номер варіанта	k	X	ε	a_n
6	7	0.1	10^{-5}	$\frac{(-1)^n \sqrt[n]{nx}}{(2n+1)!}$
7	40	-0.5	10^{-3}	$n^2 e^{-\sqrt{3n-x}}$
8	6	$\frac{\pi}{16}$	10^{-7}	$\frac{(-1)^{n-1} x^{2n-1}}{(2n+1)!}$
9	8	1.25	10^{-6}	$\frac{n^{\ln n}}{(x+\ln n)^n}$
10	9	0.1	10^{-7}	$\frac{n^{\sqrt{n}} - x}{n!}$
11	5	0.2	10^{-6}	$\frac{x^n}{n\sqrt{n}}$
12	14	4.5	10^{-5}	$(x-5)^n n$
13	6	0.25	10^{-7}	$\frac{(-1)^{n-1} x^n}{n}$
14	9	0.5	10^{-5}	$\frac{(-1)^{n-1} x^{2n}}{(2n)!}$
15	12	$\frac{\pi}{8}$	10^{-5}	$(-1)^n \frac{2^{n+1} x}{2^{2n}+1}$
16	7	0.1	10^{-5}	$\frac{(-1)^{n-1} x^{n-1}}{2n^n - 1}$
17	5	1	10^{-6}	$\frac{x}{2^n} + \frac{x}{3^n}$
18	6	0.5	10^{-6}	$\frac{(3x)^n n!}{(3n)!}$
19	3	1.25	10^{-5}	$\frac{xn}{\ln(n!+1) + n^n}$
20	4	0.25	10^{-6}	$\frac{(-1)^n (n-x)^2}{n!(2n+1)!}$
21	9	$\frac{\pi}{16}$	10^{-7}	$\frac{(-1)^n \sqrt{nx}}{(n+1)!(2n+1)!}$

Контрольні питання

1. За допомогою яких операторів можна організувати цикли в мові С?
2. Які особливості має оператор циклу **`for`**?
3. Які особливості має оператор циклу **`while`**?
4. Які особливості має оператор циклу **`do-while`**?
5. Для чого використовується оператор **`break`**?
6. Для чого використовується оператор **`continue`**?