**Структурне програмування**

Методологія структурного програмування заснована на припущенні, що логічність і зрозумілість програми забезпечує надійність, полегшує модифікацію і прискорює розробку програми.

Характерними рисами структурного програмування є:

* відмова від неструктурних передач управління;
* обмежене використання глобальних змінних;
* модульність.

**Цикл з передумовою**

Існує три види циклів: *while*, for и do. ***Цикл while*** Має наступну форму:

while (e) s;

Оператор s виконується до тих пір, поки значення виразу e "істина". Значення e обчислюється перед кожним виконанням оператора s.

У попередніх лекціях ми користувалися цією формою циклу. Розглянемо ще один приклад його роботи:

/\* вгадування числа \*/

#include <stdio.h>

main( )

{

int i = 1;

char res;

printf("Задумайтесь ціле число від 1 до 100.  
Комп'ютер спробує вгадати його.\n");

printf ("Відповідайте y, якщо здогадка правильна и");

printf("\n n, якщо програма помиляється \n");

printf("Отже, ваше число %d?\n",i);

/\*Отримання відповіді \*/

while((res = getchar( )) !='y')

if(res !='\n')

/\* Пропуск символу новий рядок \*/

printf("Ну тоді воно дорінює %d\n" ,++i);

printf("Число вгадано!\n");

}

|  |  |
| --- | --- |
| ! | Це досить проста програма. Вона написана правильно і вирішує поставлене завдання, але робить це вкрай неефективно. Даний приклад показує, що правильність написання - **не єдиний критерій, за яким необхідно оцінювати програму. При цьому дуже важлива її ефективність!** |

У наших прикладах досі використовувалися умовні вирази, але взагалі кажучи, це можуть бути вирази довільного типу. В якості оператора можна використовувати простий оператор з символом "крапка з комою" в кінці або складовий оператор, укладений у фігурні дужки. Якщо вираз істинний (або в загальному випадку дорівнює одиниці), то оператор, що входить в цикл while, виконується один раз, а потім вираз перевіряється знову.

Ця послідовність дій, що складається з перевірки та виконання оператора, періодично повторюється до тих пір, поки вираз не стане хибним (або в загальному випадку рівним нулю). Кожен такий крок називається ітерація. Дана структура аналогічна структурі оператора if. Основна відмінність полягає в тому, що в операторі if перевірка умови і (можливе) виконання оператора здійснюється тільки один раз, а в циклі while ці дії проводяться, взагалі кажучи, неодноразово.

|  |  |
| --- | --- |
| ! | При побудові циклу while ви повинні включити в нього якісь конструкції, що змінюють величину перевіряється вираження так, щоб врешті-решт воно стало хибним. В іншому випадку виконання циклу ніколи не завершиться. |

Цикл while є умовним циклом, що використовує передумову, тобто умова на вході. Він називається умовним, тому що виконання оператора залежить від істинності умови, що описується за допомогою виразу. Подібний вираз задає передумова, оскільки виконання цієї умови має бути перевірено перед початком виконання тіла циклу.

|  |  |
| --- | --- |
| ! | Алгоритми і псевдокод. Покроковий процес, гарантовано приводить до бажаного результату, називається **алгоритмом**. Пропозиції, що не є пропозицією якоїсь мови програмування, хоча і дуже нагадує те, що ми пишемо на даній мові програмування, називають **псевдокодом**.  Псевдокод дуже ефективний при розробці логіки програми. Після того як логіка здасться вам правильною, ви можете звернути особливу увагу на деталі перекладу псевдокоду на реальну мову програмування. Перевага використання псевдокоду полягає в тому, що він дозволяє сконцентруватися на логіці і структурі програми, не піклуючись поки про спосіб перекладу цих ідей на мову машини. Якщо ми хочемо поліпшити програму, нам в першу чергу необхідно поліпшити алгоритм! |

Підведемо підсумки.

Оператор while визначає операції, які циклічно виконуються до тих пір, поки вираз що перевіряється не стане хибним, або рівним нулю.. Оператор while - це цикл з передумовою. Рішення, виконати в черговий раз тіло циклу, приймається перед початком його проходження. Тому цілком можливо, що тіло циклу не буде виконано жодного разу. Оператор, який утворює тіло циклу, може бути або простим, або складеним. Форма запису:

while (выражение) оператор

Виконання оператора циклічно повторюється до тих пір, поки вираз не стане хибним, або рівним нулю.

**Цикл з лічильником**

Оператор *циклу* for

for(e1; e2; e3) s

є зручним скороченим записом для циклу while виду

e1;

while(e2) {

s;

e3;

}

*Вираз* e1 служить для задання початкових умов виконання циклу, вираз e2 забезпечує перевірку умови виходу з циклу, а вираз e3 модифікує умови, задані виразом e1. Будь-який з виразів e1, e2, e3 може бути опущено. Якщо опущено e2, то за замовчуванням замість нього підставляється значення TRUE. Наприклад, цикл for

for(;e2;) s;

з опущеними e1, e3 эквивалентен циклу

while(e2) s;

Цикл

for(;;) s;

з усіма опущеними виразами еквівалентний циклу

while(TRUE) s;

тобто еквівалентний нескінченному циклу. Такий цикл може бути перерваний тільки явним виходом з нього за допомогою операторів break, goto, return, що містяться в тілі циклу s.

Незважаючи на зовнішню схожість з ітеративними циклами for мов Паскаль і Ада або ітеративними циклами do мов Фортран або ПЛ / 1, цикл for мови Сі не є їх семантичної копією. Цикл for мови Сі має більшу спільністю, ніж цикли for і do інших мов. На відміну від цих циклів, в загальному випадку число ітерацій в циклі for мови Сі не може бути визначено до виконання цього циклу. continue

Цикл for і його альтернатива цикл while семантично майже еквівалентні, але, як вказується в роботі (Ritchie, D.M. 1980. The Programming Language - Reference Manual /. AT & T Bell Laboratories, Murray Hill, N.J. 07974), не ідентичні. Наприклад, розглянемо випадок, коли оператор s є оператором continue або складеним оператором, що містить оператор continue. Дія оператора continue полягає в переході до кінця циклу, що має різні наслідки для циклу for і його еквівалента у формі циклу while. У випадку з циклом for вираз e3 виконується до обчислення значення виразу e2, в той час як в еквівалентному циклі while вираз e3 пропускається.

Гнучкість конструкції for - наслідок способу використання виразів в специфікації циклу. До сих пір перший вираз застосовувалося для ініціалізації лічильника, друге - для завдання його граничного значення, а третє - для збільшення його поточного значення на 1. Але крім цієї можливості існує ще й багато інших можливостей його застосування, дев'ять з яких ми наведемо нижче.

1. Можна застосовувати операцію зменшення для рахунку в порядку убування замість рахунку в порядку зростання:
2. for(n=10; n>0; n--)
3. printf("%d секунд!\n",n);

printf("Пуск! \n");

1. При бажанні можна вести рахунок двійками, десятками і т.д.:
2. for (n=2; n<60;n=n+13);

printf("%d\n",n);

1. Можна вести підрахунок за допомогою символів, а не тільки чисел:
2. for(ch = 'a';ch<='z';ch++)

printf("Величина коду ASCII для %с дорівнює %d.\n",ch,ch);

При виконанні цього оператора будуть виведені на друк всі букви від а до z разом з їх кодами ASCII. Цей оператор працює, оскільки символи в пам'яті машини розміщуються в вигляді чисел, і тому в даному фрагменті ведеться рахунок з використанням цілих чисел.

1. Можна перевірити виконання деякої довільної умови, відмінної від умови, що накладається на число ітерацій. Розглянемо приклад в програмі що друкує таблицю кубів цілих чисел:
2. /\* табліця кубів \*/
3. main( )
4. {
5. int num;
6. for(num=1; num<=6; num++)
7. printf("%5d %5d\n", num,num\*num\*num);

}

А тепер замінимо специфікацію

for(num=1; num<=6; num++)

на

for(num=1; num<6; num++)

Це було б доцільно в разі, якби нас більше займало обмеження максимального значення діапазону кубів чисел, а не кількість ітерацій.

1. Можна зробити так, щоб значення деякої величини зростала в геометричній, а не в арифметичній прогресії, тобто замість додавання фіксованого значення на кожному кроці циклу виконувалося б множення:
2. for(x=100.0; x<150.0; x=x\*1.1)

printf("Ваш долг теперь %3.2f.\n",x);

У цьому фрагменті програми значення змінної x множиться на 1.1 на кожному кроці циклу, що збільшує її на 10%. Результат виглядає наступним чином:

Ваш долг теперь 100.00.

Ваш долг теперь 110.00.

Ваш долг теперь 121.00.

Ваш долг теперь 133.10.

Ваш долг теперь 146.41.

1. У якості третього вираження можна використовувати будь-який правильно складений вираз. Який б вираз ми не вказали, його значення буде змінюватися при кожній ітерації:
2. for (x=y=1; y<=75; y=5\*x++)

printf("%10d %10d\n",x,y);

Зверніть увагу, що в специфікації циклу перевіряється значення y, а не x. У кожному з трьох виразів, які керують роботою циклу for, можуть використовуватися будь-які змінні.

|  |  |
| --- | --- |
| ! | Хоча цей приклад і правильний, він не може служити ілюстрацією хорошого стилю програмування. Програма виглядала б набагато зрозуміліше, якби ми не змішали процес зміни змінної циклу з алгебраїчними обчисленнями. |

1. Можна навіть опустити одне або більше виразів, але при цьому не можна опустити символи "крапка з комою". тіло циклу
2. for(;;) {
3. ...

}

буде виконуватися нескінченну кількість разів, оскільки пуста умова завжди вважається дійсною.

1. Перший вираз не обов'язково має ініціювати змінну. Замість цього, наприклад, там міг би стояти оператор printf( ). Необхідно пам'ятати тільки, що перший вираз обчислюється тільки один раз перед тим, як інші частини циклу почнуть виконуватися.
2. for(printf("Запам’ятовуй числа! \n"); num != 6;)
3. scanf("%d",&num);

printf("Це як раз те, чогоя хочу!\n");

У цьому фрагменті перше повідомлення виявляється виведеним на друк один раз, а потім здійснюється прийом чисел, що вводяться до тих пір, поки не надійде число 6.

1. Параметри, що входять до виразу, що знаходяться в специфікації циклу, можна змінити при виконанні операцій в тілі циклу. Припустимо, наприклад, що у вас є цикл зі специфікацією такого вигляду:

for(n=1; n<1000; n+t)

І якщо після декількох ітерацій наша програма вирішує, що величина t занадто мала або велика, то оператор if всередині циклу може змінити значення параметра. У діалогової програми **користувач може змінити цей параметр в процесі виконання циклу.** for

**Рідведемо підсумки**. В операторі for використовуються три вирази, що керують роботою циклу. Вони розділені символом "крапка з комою". Ініціалізуючий вираз обчислюється тільки один раз до початку виконання якого-небудь з операторів циклу. Якщо вираз що перевіряється виявляється істинним, або не рівним нулю, тіло циклу виконується один раз. Потім обчислюється величина корегуємого виразу, і значення виразу що перевіряється визначається знову.

Оператор for - це цикл з передумовою. Рішення про те, чи виконати в черговий раз тіло циклу чи ні, приймається до початку його проходження. Тому може статися так, що тіло циклу не буде виконано жодного разу. Оператор, який утворює тіло циклу, може бути як простим, так і складеним.

### Операція "кома" збільшує гнучкість використання циклу for, дозволяючи включити в його специфікацію кілька ініціюючих або коригувальних виразів.

**Цикл з постумовою**

Цикл

do оператор while(e);

виконується до тих пір, поки вираз e має значення "істина". На відміну від циклу while, в якому перевірка умови закінчення циклу робиться до виконання тіла циклу, в циклі do така перевірка має місце після виконання тіла циклу. Отже, тіло циклу do буде виконано хоча б один раз, навіть якщо вираз e має значення "брехня" c самого початку. Цикл do аналогічний циклу repeat в мові *Паскаль*, відмінний від нього тільки тим , що цикл repeat **виконується до тих пір, поки деяка умова виходу з циклу не стає істинною, а цикл do виконується весь час, поки деяка умова залишається істинним!**

|  |  |
| --- | --- |
| ! | Цикл do while – це конструкція з умовою на виході.  Використовувати цикл do while найкраще в тих випадках, коли повинна бути виконана принаймні одна ітерація. Наприклад, ми могли б застосувати цикл do while в нашій програмі вгадування числа. На псевдокоді алгоритм роботи програми можна тоді записати в такий спосіб:  do {  висуньте припущення  отримаєте відповідь виду y,n  } while(Відповідь не співпадає з y); |

|  |  |
| --- | --- |
| ! | Ви повинні уникати використання циклу do while, структура якого аналогічна представленій нижче операції: запитайте користувача, чи хоче він продовжувати  do  оператор  while (відповідь буде так)  В даному випадку, після того як користувач відповість "ні", "оператор" буде виконаний, оскільки перевірка здійснюється занадто пізно. |

Приклад:

do

scanf("%d", &number);

while(number!=50);

Підведемо підсумки.

Оператор do while визначає дії, які циклічно виконуються до тих пір, поки вираз що перевіряється не стане хибним, або рівним нулю. Оператор do while - це цикл з постумовою. Рішення про те виконувати чи ні в черговий раз тіло циклу, приймається після його проходження. Тому тіло циклу буде виконано принаймні один раз. Оператор, який утворює тіло циклу, може бути як простим, так і складеним.

**Особливості роботи з мовою СІ** . Циклом якого виду найкраще скористатися? По-перше, вирішите, чи потрібен вам цикл з передумовою або ж з постумовою. Найчастіше вам потрібен буде цикл з передумовою. За оцінками Керніган і Рітчі, в середньому цикли з умовою поста становлять лише 5% загального числа використовуваних циклів. Існує кілька причин, за якими програмісти воліють користуватися циклами з передумовою. У їх числі один загальний принцип, згідно з яким краще подивитися, куди ви стрибаєте, до стрибка, а не після. Другим моментом є те, що програму легше читати, якщо умова яка перевіряється знаходиться на початку циклу. І нарешті, у багатьох випадках важливо, щоб тіло циклу ігнорувалося повністю, якщо умова спочатку не виконується.

*Вкладені цикли. Вкладеним називається цикл, що знаходиться всередині іншого циклу.*

Приклад:

/\* Просте число - це таке число, яке  
ділиться без остачі тільки на 1 і саме на себе.  
Першими простими числами будуть 2, 3, 5, 7 і 11. \*/

/\*прості числа\*/

main()

{

int number, divisor, limit;

int count=0;

printf("Вкажіть, будь ласка, верхню межа  
для пошуку простих чисел. \n");

printf("Bерхня межа повиненна бути 2  
або більше.\n");

scanf("%d",&limit);

while (limit<2) {

/\* друга спроба, якщо помилка при введенні \*/

printf("Bи були неуважні!  
спробуйте ще раз \n");

scanf("%d",&limit);

}

printf("Зараз надрукуємо прості числа!\n");

for (number=2;number<=limit; number++)

/\*зовнішній цикл\*/

{

for(divisor=2; number%divisor!=0; divisor++)

;

if(divisor==number)

{

printf("%5d",number);

if(++count%10==0)

printf("\n");/\* новий рядок починається  
через кожні 10 простих чисел \*/

}

}

printf("Pa6oта закінчена!\n");

}

### У зовнішньому циклі кожне число, починаючи з 2 і закінчуючи величиною limit, послідовно береться для перевірки. Зазначена перевірка здійснюється у внутрішньому циклі. У count зберігається лічильник одержуваних простих чисел. При друку кожне одинадцяте просте число друкується з нового рядка.

**Інші керуючі оператори**

*Оператори, що визначають циклічні обчислення, які ми розглянули, і умовні оператори* if, if-else і switch, є найважливішими засобами управління виконанням програми на мові Сі. Вони повинні використовуватися для реалізації загальної структури програми. Три оператора, що розглядаються нижче, зазвичай розглядаються рідше, оскільки занадто часте їх використання погіршує читаність програми, збільшує ймовірність помилок і ускладнює її модифікацію. Ніклаус Вірт дав визначення структурного програмування, як програмування без goto.

#### Оператор break

Оператор break викристовується для виходу з операторів while, do, for, switch, що безпосередньо його містить. Управління передається на оператор, наступний за оператором, з якого здійснено вихід. Оператор break має форму

break;

Приклад:

while((ch=getchar()) != EOF)

/\* читаєтся символ ch=getchar(). Якщо він не співпадає з EOF, виконується тіло оператора while \*/

{

if(ch=='\n')

break;

putchar(ch);

}

Робота циклу повністю припиняється, як тільки при введенні зустрічається символ "новий рядок".

#### Оператор continue

Оператор continue служить для пропуска решти виконуваної ітерації циклу, що його містить. Якщо умовами циклу допускається нова ітерація, то вона виконується, в іншому випадку цикл завершується. Оператор continue має наступну форму:

continue;

Пример:

while((ch=getchar()) != EOF)

/\* читается символ ch=getchar(). Якщо він не співпадає з EOF, виконується тіло оператора while \*/

{

if(ch=='\n')

continue;

putchar(ch);

}

В версії з оператором continue просто пропускаються символи "новий рядок", а вихід з циклу відбувається, тільки коли читається ознака EOF.

#### Оператор goto

Оператор goto призначений для безумовної передачі керування до оператора з вказаною міткою. Він має наступну форму:

goto метка;

Керниган і Рітчі вважають оператор goto "Надзвичайно поганим" засобом і пропонують застосовувати його як можна рідше або не застосовувати зовсім. Наведемо приклад запису оператора:

goto part1;

Щоб цей оператор виконувався правильно, необхідна наявність іншого оператора, що має мітку part1. У цьому випадку запис оператора починається з мітки, за якою слідує двокрапка:

part1: printf("точка переходу\n");

|  |  |
| --- | --- |
| ! | Уникайте використовувати goto! З тих пір як Дейкстра опублікував свою знамениту працю "Шкідливість оператора GOTO", фахівці билися над методами структурного програмування. Повний опис структурного програмування виходить за рамки цього курсу лекцій, проте відзначимо, що потрібно дуже обережно використовувати оператори: goto, break, continue, return. |

Якщо без операторів goto, break, continue, return ніяк не обійтись , то при використанні goto переходьте вперед по коду, а не назад.

Оператор break краще не використовувати для передчасного виходу з циклу, його корисно використовувати всередині оператора switch.

Оператор continue небажано використовувати для модифікації логіки циклів.

Чому небажано використовувати функції з багатьма операторами return. Один із принципів структурного програмування полягає в тому, що програма повинна мати одну точку входу і одну точку виходу. Функції з багатьма операторами return більш складні для читання, ніж ті, які мають лише один оператор return в кінці тіла функції.