**Практична робота №6. Розгляд прикладів програм лінійної, розгалуженої та циклічної структури, побудова блок-схем**

**Мета:** навчитися складати алгоритми програм лінійної, розгалуженої та циклічної структури**.**

**Завдання.**

1. Ознайомтеся з теоретичною частиною.
2. Намалюйте блок-схему до задачі розглянутій в теоретичній частині (рівняння *ax*+*b* =0) для різних пар a,b.
3. Зробіть постановку задачі з визначеннямвхідних та вихідних даних, розробіть блок-схему для наступних двох розрахунків та напишіть програму.

 якщо 

Запис  означає, х може приймати значення з діапазону [-1;2]. тобто х ≥ -1 та х ≤2. Запис  означає, що х змінюється з кроком 0.2.

 якщо    

1. Надайте відповідь на запитання: чи можна об’єднати ці розрахунки в одному циклі?

***Блок - схеми можна розробити в будь-якому наявному у вас редакторі, зокрема у Word, використовуючи графічні примітиви, які надаються в меню Вставка→Фігури. В крайньому випадку намалюйте в зошиті, сфотографуйте і вставте в файл з виконанням завдання як малюнок.***

Результати у вигляді текстового файлу надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**АР<Номер групи><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**. Наприклад, **АРPTBD-2104P**buts.doc.

**Тему в заголовку листа записати**

**АР <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище >**

**Строк виконання 22.03.2024**

**ТЕОРЕТИЧНА ЧАСТИНА**

**Проектування програми**.

Найчастіше алгоритм створюють, поступово уточнюючи поняття, пов'язані із задачею, і необхідні дії. Тоді кажуть, що розробку ведуть **згори донизу**.

Напишемо програму, що розв'язує рівняння *ax*+*b* =0.

**Уточнення постановки задачі**. Визначимо вхідні й вихідні дані програми. *Вхід*: коефіцієнти рівняння – два дійсних числа *a* та *b*. *Вихід*: кількість розв'язків; якщо розв'язок один – то саме цей розв'язок.

**Математичний аналіз задачі**. За умови *a* ≠0 рівняння має один розв'язок - *b*/*a*.

Тому потрібна перевірка, що вхідні дані коректні, тобто, що *a* ≠0.

**Проектування програми**.

У загальному вигляді алгоритм такий:

1. Отримати вхідні дані.

2. Обробити вхідні дані.

3. Вивести результат обробки.

Уточнимо кожен із кроків алгоритму.

"Отримати вхідні дані".

1.1. Вивести запрошення на введення даних.

1.2. Увести коефіцієнти рівняння в дійсні змінні **a** та **b**.

"Обробити вхідні дані". На основі аналізу задачі, якщо *a* ≠ 0, то кількість розв'язків дорівнює 1, а розв'язком є -*b*/*a*. Розв'язок запам'ятаємо в дійсній змінній **x**.

"Вивести результат обробки".

3.1. Вивести рівняння, уведене користувачем.

3.2. За допомогою значень змінної **x** вивести розв'язок, якщо він є, інакше вивести повідомлення про відсутність рішення.

Нарешті, можна кодувати.

**//програма, що розв'язує рівняння ax+b=0**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**double a=0, b=0; //** коефіцієнти рівняння

**double x; // розв'язок рівняння**

**// отримати вхідні дані**

**cout<<"Enter coefficients a and b of " << "equation ax+b=0 (2 reals)\n";**

**cin>>a>>b;**

**// Підготовити виведення результату**

**cout<<"Equation "<<a<<"x+"<< b <<"=0 has ";**

**// обробити введені дані**

**if (a!=0)**

**{x=(-b)/a;**

**// повідомити результат**

**cout <<" solution " << x<<"\n";**

**}**

**else cout<<" no solution\n"; // повідомити результат**

**system("pause");**

**return 0;**

**}**

У цій програмі кожен фрагмент коду задає певні дії для отримання необхідного результату, тобто має своє призначення, або свій **обов'язок**. На перший погляд, програму можна зробити коротшою: якби обчислювати й відразу виводити розв'язок. Однак тоді код обробки даних був би *перевантажений обов'язками*, тобто відповідав за кілька різних функцій (тут – обчислення й виведення на екран).

Якщо кожен фрагмент коду має своє, персональне призначення, то це, по-перше, робить загальну структуру програми прозорішою і, по-друге, полегшує модифікацію окремих частин програми. У наведеному прикладі можна забажати змінити вихідне текстове повідомлення, і це не вплине на алгоритм обчислення результату. Отже, відокремлення обробки від виведення результатів цілком обґрунтоване.

Ускладнимо розглянуту задачу. Нехай розв’язок потрібно знайти не для однієї пари чисел *a* та *b*, а для якоїсь заздалегідь невідомої кількості їх комбінацій. В цьому випадку значення *a* та *b* потрібно вводити в циклі та запитувати можливість припинення. Для цього програма може бути модифікована таким чином.

**//програма, що розв'язує рівняння ax+b=0 для різних пар a,b**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**double a=0, b=0; // коефіцієнти рівняння**

**double x; // розв'язок рівняння**

**int IsContinue=1; // Ознака продовження**

**while (IsContinue == 1)**

**{**

**// отримати вхідні дані**

**cout<<"Enter coefficients a and b of " << "equation ax+b=0 (2 reals)\n";**

**cin>>a>>b;**

**// Підготовити виведення результату**

**cout<<"Equation "<<a<<"x+"<< b <<"=0 has ";**

**// обробити введені дані**

**if (a!=0)**

**{x=(-b)/a;**

**// повідомити результат**

**cout <<" solution " << x<<"\n";**

**}**

**else cout<<" no solution\n"; // повідомити результат**

**cout <<" Enter 1 for continue \n ";**

**cin>> IsContinue;**

**}**

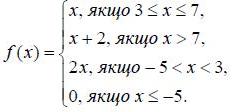
**system("pause");**

**return 0;**

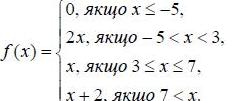
**}**

**Використання умовних операторів**

**Розглянемо приклад**. Написати фрагмент коду, що за дійсним *x* обчислює значення *f*(*x*) і присвоює його дійсній змінній *y*.



Перепишемо формулу обчислення *f*(*x*) у еквівалентному вигляді.



Якщо код писати безпосередньо за формулою, то такий фрагмент коду:

**if (x<=-5) y=0;**

**if (-5<x && x<3) y=2\*x;**

**if (3<=x && x<=7) y=x;**

**if (7<x) y=x+2;**

Цей код є правильним, але *неоптимальним* за кількістю виконуваних операцій. Якщо значенням змінної **x** є **-9.0**, то обчислюються всі чотири умови, хоча з погляду математики зрозуміло, що за істинності умови **x<=-5** решта умов хибні. Отже, модифікуємо фрагмент коду.

**if (x<=-5) y=0;**

**else if (-5<x && x<3) y=2\*x;**

**else if (3<=x && x<=7) y=x;**

**else if (7<x) y=x+2;**

Тепер за значення **-9.0** обчислюється тільки перша умова. Нехай значенням змінної **x** є **0.1**. Тоді вираз **x<=-5** є хибним, і обчислюється вираз **-5<x&&x<3**. Однак **x<=-5** хибний, тому **-5<x** є істинним! Отже, значенням виразу -**5<x&&x<3** є значення **x<3**. Міркуючи так само далі, отримуємо ще один варіант.

**if (x<=-5) y=0;**

**else if (x<3) y=2\*x; // тут значення -5<x істинне**

**else if (x<=7) y=x; // тут значення 3<=x істинне**

**else y=x+2; // тут значення 7<x істинне**

Зауважимо: наступний фрагмент коду для нашої задачі є *помилковим*.

**if (x<=-5) y=0;**

**if (x<3) y=2\*x;**

**if (x<=7) y=x;**

**else y=x+2;**

Якщо значенням **x** є **1.0**, то умова **x<3** істинна, тому спочатку виконується присвоювання **y=2\*x**. Проте потім перевіряється умова **x<=7**, виявляється істинною, і виконується **y=x**, що, вочевидь, є помилковим.

Дуже часто інструкції розгалуження є частиною інших розгалужень, тому їх записують "східцями", зсуваючи вкладену інструкцію праворуч, наприклад, таким чином:

**if (*умова1*)**

**if (*умова2*)**

***інструкція1***

**else**

***інструкція2***

**else ...**

Інколи виникають довгі ланцюги розгалужень, в яких за словами **else** йдуть наступні розгалуження з **if** на початку. Краще записувати їх у такому вигляді:

**if (*умова*)**

***інструкція***

**else if (*умова*)**

***інструкція***

**else if (*умова*)**

**...**

**З попередньої лекції зірваної тривогою**

Інструкція циклу **for** або **for**-інструкція, має загальний вигляд

**for (*початкова дія*; *умова*; *перехідна дія*)**

***основна дія***

Слово **for** зарезервоване, дужки та два знаки; усередині дужок є обов'язковими. Початкова дія, умова й перехідна дія є *виразами* (кожен із них може бути порожнім), основна дія – *інструкцією*.

Оператор циклу **for** реалізується таким чином:

* виконується початкова дія - вираз ініціювання (виконання цієї нотації може бути здійснено до оператора **for**);
* обчислюється вираз-умова;
* якщо умовний вираз приймає значення «істина» — виконуються оператори циклу;
* обчислюється вираз ітерації;
* знову перевіряється умова;
* як тільки умова прийме значення **0** («хиба, неправда»), керування передається оператору, що розташований за оператором циклу **for**.

Оскільки в операторі **for** перевірка виразу-умови відбувається перед циклом, то у випадку помилкової умови цикл може жодного разу не виконуватися.

Оператор **for** може використовувати декілька змінних, що керують циклом, а будь-які вирази можуть бути відсутніми, наприклад:

**int n, у;**

**for (int k = 0, n = 20;k <= n; k++, n--)**

**y = k \* n;**

або

**int і=0;**

**for (; і < 4; i++)**

Перший фрагмент має два вирази ініціювання і два вирази ітерації. Спочатку відбувається присвоювання значень змінним **k = 0 і** **n = 20**, далі здійснюється порівняння **k <= n** і, якщо ця умова має значення «істина», то буде виконуватися тіло циклу, а потім вираз **k++ і** **n–**, якщо ж умова не виконується, то цикл припиняє свою роботу.

C++ дозволяє поєднати ці дві дії в одному виразі – за допомогою операції послідовного обчислення. Операція зі знаком "**,**" позначає послідовне обчислення виразів, записаних через кому (в прикладі це **к = 0, n = 20;)**. Ця послідовність виразів розглядається як один вираз; його значенням є значення останнього виразу. Операція послідовного обчислення дозволяє на місці одного виразу записати кілька.

Операторам циклів с параметром **for** потрібно віддати перевагу при організації циклів з лічильниками.

В циклі **for** можна використовувати інструкції **break** та **continue.** Інструкція **break** у тілі циклу **for** завершує його виконання, а інструкція **continue** завершує виконання лише тіла циклу; відразу після неї виконується перехідна дія.

**Приклад.** Дуже часто інструкція циклу **for** зустрічається у вигляді

**for (k=0; k<n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 0, 1, 2, …, *n*-1 або у вигляді

**for (k=1; k<=n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 1, 2, …, *n*, або у вигляді

**for (k=n; k>0; --k) *інструкція***

й задає виконання *інструкції* за значень *k* = *n*, *n*-1, …, 2, 1. Змінну **k** у цих ситуаціях інколи називають **лічильником циклу**.

**Переривання break та продовження циклу continue**

Виконання інструкції **break** всередині циклу будь-якого різновиду перериває й завершує цикл; далі виконуються дії, наступні за цим циклом. Якщо **break** записано в інструкції циклу, вкладеній в іншу інструкцію циклу, то виконання **break** завершує вкладений цикл, а зовнішній цикл продовжується.

Інструкція **continue** всередині циклу задає перехід на кінець тіла циклу. В інструкціях циклу з перед- і післяумовою після **continue** обчислюється умова продовження циклу.

**Приклад.** За допомогою клавіатури вводиться послідовність дійсних чисел. Потрібно підрахувати суму її додатних елементів, а за появи 0 видати накопичену суму й завершити роботу.

Запрограмуємо цикл, в якому вводиться й обробляється послідовність чисел. Уведене число зберігаємо в змінній **x**, а суму додатних елементів – у змінній **sum**. Якщо під час уведення трапилася помилка, то подальші дії з уведення не виконуються, а змінна **x** зберігає своє останнє значення. Тому умовою продовження циклу буде саме відсутність помилок (інакше можна отримати цикл, який ніколи не завершиться!). Цю умову задає значення виразу введення **cin>>x**, перетворене до логічного типу.

**#include <iostream>**

**using namespace std;**

**int main()**

**{ double x;**

**double sum=0;**

**cout<<"Enter reals:\n";**

**while (cin>>x){**

**if (x==0.) break; //виходимо з циклу**

**if (x<0.) continue; //пропускаємо від'ємні**

**sum+=x;**

**}**

**cout << "sum=" << sum << endl;**

**system("pause");**

**return 0;**

**}**

Використання інструкції **continue** в цій програмі є дуже штучним. Ще одним недоліком є те, що в кінці не повідомляється, чи були помилки під час уведення. Інструкції програми виконуються *в порядку їх запису в програмі*. Про таку програму кажуть, що вона **структурована**. Інструкції **break** і **continue** *порушують* цей порядок обчислень, заплутуючи текст програми. Тому, користуючись ними, програміст повинен ретельно відслідковувати точку програми, якою продовжуються обчислення. Інколи ці інструкції дійсно скорочують запис розгалужень у циклі, проте в більшості випадків ті ж самі дії *можна описати без них*. Тому краще не зловживати **break** і **continue**.

**ДЛЯ ДОВІДКИ**

**Математичні функції (**заголовний файл **math.h)**

| **Прототип функції** | **Ім’я** | **Призначення** |
| --- | --- | --- |
| double sin (double \_х); | **sin (x)** | синус x (в радіанах) — **sin x** |
| double cos (double \_x); | **cos (x)** | косинус x (в радіанах) — **cos х** |
| double tan (double \_x); | **tan (x)** | тангенс х (в радіанах) — **tg х** |
| double asin (double \_x); | **asin (x)** | арксинус х — **arcsin х** |
| double acos (double \_x); | **acos (x)** | арккосинус х — **arcos х** |
| double atan (double \_x); | **atan (x)** | арктангенс х — **arctg х** |
| double atan2 (double \_y, double\_x); | **atan2 (y,x)** | арктангенс у/х — **arctg (у/х)** |
| double sinh (double \_x); | **sinh (x)** | синус гіперболічний х — **sh х** |
| double cosh (double \_x); | **cosh (x)** | косинус гіперболічний х — **ch х** |
| double tanh (double \_x); | **tanh (x)** | тангенс гіперболічний х — **th х** |
| double log (double \_x); | **log (x)** | натуральний логарифм х — **ln х** |
| double log10 (double \_x); | **log10 (x)** | десятковий логарифм х — **log х** |
| double exp (double \_x); | **exp (x)** | піднесення е до степеня х — **ех** |
| double pow (double \_x, double\_y); | **pow (x,y)** | піднесення х до степеня у — **ху** |
| double pow 10 (int \_p) | **pow10 (p)** | повертає **10р** |
| double sqrt (double \_х); | **sqrt (x)** | корінь iз x, x > 0 |
| double hypot (double\_x, double\_y); | **hypot (x,y)** | корінь із (х2+у2) |
| double fabs (double \_\_x); | **fabs (x)** | абсолютне значення х — |х| типу **double** |
| int abs (int \_x); | **abs (x)** | абсолютне значення х — |х| типу **int** |
| long labs (long \_x); | **labs (x)** | абсолютне значення х — |х| типу **long** |
| double fmod (double \_\_x, double\_y); | **fmod (x,y)** | залишок від ділення х на у |
| double ceil (double \_\_x); | **ceil (x)** | округлення до більшого |
| double floor (double \_x); | **floor (x)** | повертає найближче ціле, не більше за х |
| double modf (double \_x, double); | **modf(x,&p)** | виділяє цілу й дробову частинні числа |
| double atof(const char\* \_s); | **atof (s)** | перетворює рядок символів у число з плаваючою крапкою |