### Умови

#### 4.1.1. Операції порівняння

Двомісні операції **порівняння** мають знаки **==, !=, >,** **<, >=, <=** ("дорівнює", "не дорівнює", "більше", "менше", "більше або дорівнює", "менше або дорівнює"). Результатом порівняння є **false** ("хибність") або **true** ("істина") є значення логічного типу **bool**. Нагадаємо, що при перетворенні логічних значень до типу цілих **false** стає **0**, а **true – 1**. При зворотному перетворенні ненульові значення дають **true**, а значення 0 (нуль) – **false**.

#### Приклади

1. Вирази **1==2** та **'A'=='a'** мають значення **false**, вирази **1!=2** та **1>=1** – значення **true**. Значенням виразу **sizeof(b\*b-**

**4\*a\*c)>0** є **true**.

1. Значенням виразу **b\*b-4\*a\*c>0** за **a==1**, **b==5**, **c==4** є **true**, за **a==1**, **b==2**, **c==1** та **a==1**, **b==1**, **c==1** – **false**. 

Порівнювати одне з одним можна не лише числові, але й символьні та логічні значення. При цьому символьні й логічні значення перетворюються до цілих.

**Приклад**. Значенням виразу **'a'<='A'** є **false**, а виразів

**'4'<'9'** та **'A'<'Z'** – **true** (значеннями **int('a')**, **int('A')**, **int('4')**, **int('9')**, **int('Z')** є числа 97, 65, 52, 57, 90, відповідно). Значенням виразу **false==true** є **false**, а виразу **false<true** – **true**. 

#### Вправи

4.1. У чому відмінність виразів **a=3** та **a==3**?

4.2. Обчислити значення виразів а) **1.5<7**; б) **'3'<'a'**;

в) **'Z'<'a'**; г) **false<true**; д) **false<-1**? Які неявні перетворення типів виконуються під час їх обчислення?

##### 4.1.2. Логічні операції

Двомісні операції "і", "або" та одномісна "не" (відповідно булеве множення, або кон'юнкція, булеве додавання, або диз'юнкція, і заперечення) у логіці й математиці називаються **булевими**, або **логічними**, і застосовуються до булевих значень. У мові С++ вони відповідно мають знаки **&&**, **||** та **!**, застосовуються до значень логічного типу й породжують значення **false** або **true**. Їх означення наведено в таблиці.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***A*** | ***B*** | ***A* && *B*** | ***A* || *B*** | **!*A*** |
| **false** | **false** | **false** | **false** | **true** |
| **false** | **true** | **false** | **true** | **true** |
| **true** | **false** | **false** | **true** | **false** |
| **true** | **true** | **true** | **true** | **false** |

**Приклад**. Вирази **(2<3)&&(2<4)**, **(0>1)||(1>0)** та

**!('a'=='x')** мають значення **true**, вирази **(2<-2)&&true** та **!(5==5)** – значення **false**. 

Пріоритет порівнянь нижчий за пріоритет **+**, **-**, **<<** і вищий за пріоритет **&&**, **||**. Оператори **>,** **>=**, **<**, **<=** мають вищий пріоритет ніж **==** та **!=**, а **&&** – вищий ніж **||**.

Докладніше пріоритети операторів див. у додатку Б.

#### Приклади

1. Ураховуючи пріоритети операцій, вираз **(k>0)&&(d>0)** можна записати як **k>0&&d>0**, але перша форма запису сприймається людиною краще.
2. Математичну умову *x*<*y*<*z* можна записати мовою C++ як **(x<y)&&(y<z)**. Запис **x<y<z** виражає зовсім іншу умову, а саме **(x<y)<z**. Обчислимо вираз **(-3<-2)<-1**. Значенням виразу **(-3<2)** є **true**. Далі обчислюємо **true<-1**, починаючи зі зведення аргументів порівняння до типу **int**, тобто в кінці обчислюється

**1<-1** і його значенням є **false**. Проте *з погляду математики* запис -3<-2<-1 істинний.

1. Математичну умову *x*=*y*=*z* того, що числа *x*, *y*, *z* попарно рівні між собою, можна записати мовою C++ як

**(x==y) && (y==z).** Для дійсних змінних **x**, **y**, **z** вираз **x=y=z** еквівалентний виразу **x=(y=z)** і задає присвоювання значення змінної **z** змінним **y** та **x**; його результатом буде значення змінної **z** – деяке дійсне число.

Записуючи вирази, не зловживайте пріоритетами операцій! Час, що економиться на дужках у виразах, потім багаторазово витрачається на розшифрування логіки виразів і пошук помилок.

Компілятори С++ забезпечують так зване **ледаче**[[1]](#footnote-1), або **скорочене**, обчислення булевих операцій **&&** та **||**. Спочатку обчислюється їх перший операнд. Якщо для операції **&&** це **false**, то другий операнд обчислювати не треба, адже результатом усе одно буде **false**. Аналогічно, якщо перший операнд операції **||** має значення **true**, то другий операнд не потрібен.

**Приклад**. У виразі **(2\*2==5)&&(323345%2209==37)** обчислюється тільки **2\*2==5** (хибність), а у виразі

**(2\*2==4)||(323345%2209==37)** – тільки **2\*2==4** (істина). 

#### Вправи

4.3. Обчислити значення виразу: а) **!true==0;**

б)**!false||(false==1)**; в) **(2>1)&&!true**.

4.4. Обчислити значення виразу **6<3&&7<5||3==5**.

4.5. Що буде виведено на екран за інструкціями?

**int a=3; bool b=(2<3)||(a=-1); cout<<a<<endl;**

##### 4.1.3. Умовні вирази

Вирази, що можуть бути або істинними, або хибними, називаються **умовними**, або **умовами**. У програмуванні вони відіграють особливу роль, оскільки є основою для прийняття рішень із вибору подальшого шляху обчислень. Значення умовного виразу належить логічному типу (або може бути перетвореним до нього).

Наприклад, умовою того, що рівняння *ax*+*b* 0 з коефіцієнтами *a* та *b* має єдиний розв'язок, є нерівність *a*  0. Залежно від значення умови **true** або **false** можна обчислити розв'язок рівняння або з'ясувати, чи має воно нескінченно багато розв'язків, чи не має жодного (для цього може знадобитися умова *b* 0).

Умову часто використовують як *ознаку* деякої властивості. Ознака істинна, якщо властивість має місце, інакше – хибна. Наприклад, умова *a*  0 є ознакою того, що рівняння *ax*+*b* 0 має один розв'язок.

*Найпростіші умови* дуже часто мають вигляд порівнянь. Наприклад, умовою того, що значення цілої змінної **a** парне (ділиться на 2 без остачі), є те, що воно дає остачу 0 від ділення на 2: **a%2==0**.

*Складніші умови* формулюють як системи або сукупності, складені з простіших умов. *Системи умов* записують мовою С++ за допомогою операції **&&**, *сукупності* – за допомогою **||**. **Приклади**

1. Умовою того, що число *x* належить проміжку [*a*; *b*], є математичний вираз *a*  *x*  *b*. Виразимо її так: **(a<=x)&&(x<=b).** Аналогічно умову того, що значення змінної **char c** є десятковою цифрою, можна записати як **('0'<=c)&&(c<='9')**.
2. Припустимо, що значення числових змінних **a**, **b**, **c** зображують довжини відрізків. Умовою того, що з відрізків можна утворити трикутник, є система нерівностей: *a* > 0, *b* > 0, *c* > 0, *a*+*b* > *c*, *a*+*c* > *b*, *b*+*c* > *a*.

Цій системі відповідає вираз (дужки не обов'язкові, але додають наочності)

**(a>0)&&(b>0)&&(c>0)&&(a+b>c)&&(a+c>b)&&(b+c>a)**

1. Той факт, що цілі ненульові числа *a*, *b*, *c* утворюють геометричну прогресію, математично можна виразити як *b*/*a*=*c*/*b*. Результатом ділення цілих у С++ є ціле, тому вираз **(b/a)==(c/b)** узагалі *не відповідає математичній умові*. Вираз

(**double(b)/a)==(double(c)/b)** вирішує проблему з діленням цілих, але теж *не є прийнятним*. Справа в тому, що тип **double** може містити лише наближення справжніх числових значень *b*/*a* та *c*/*b*, тому результат порівняння значень типу **double** може відрізнятися від математичного порівняння. Щоб уникнути цього, запишемо вираз у математично еквівалентній формі: *bb* *ac*. Отже, умова того, що числа *a*, *b*, *c* утворюють геометричну прогресію, у C++ має вигляд **b\*b==a\*c**. 

Перш ніж записувати математичну умову мовою програмування, спробуйте провести математичні перетворення.

#### Вправи

4.6. Написати умову того, що значенням змінної **a** символьного типу є: а) цифра **1**; б) літера **h**.

4.7. Написати умову того, що значення цілої змінної **b** ділиться на 3 без остачі.

4.8. Написати умову того, що значення цілих змінних **a** та **b** мають однакову парність.

4.9. Дійсні змінні **a**, **b** задають кінці відрізка дійсної прямої. Написати вираз, який задає ознаку того, що довжина цього відрізка менше або дорівнює 0,001.

4.10. Написати умову того, що значенням змінної **c** символьного типу є:

а) велика латинська літера (від **'A'** до **'Z'**);

б) шістнадцяткова цифра.

4.11. Написати умову того, що значення дійсних змінних **x**, **y**, **z** попарно різні.

4.12. Написати умову того, що ціле **a** ділиться на ціле **b** без остачі. Не забудьте врахувати, що значенням **b** може бути **0**.

4.13. Написати умову того, що відрізки [*a*, *b*] та [*c*, *d*] осі *Ox*:

а) не мають спільних точок;

б) мають хоча б одну спільну точку.

4.14. Написати умову того, що точки з координатами (*x*1, *y*1) та (*x*2, *y*2): а) збігаються; б) не збігаються.

4.15. Написати умову того, що точки з координатами (*x*1, *y*1), (*x*2, *y*2) та (*x*3 *y*3) лежать на одній прямій.

4.16. Пряму на площині можна задати дійсними коефіцієнтами *a*, *b*, *c* рівняння *ax*+*by*+*c* 0. Написати умову того, що дійсні числа *a*, *b*, *c*задають:

а) пряму; б) вертикальну пряму;

в) горизонтальну пряму;

г) пряму, що проходить через початок координат.

4.17. Пряму на площині задано дійсними коефіцієнтами *a*, *b*, *c*рівняння *ax*+*by*+*c* 0. Написати умову того, що точки з координатами (*x*1, *y*1) та (*x*2, *y*2) лежать у різних півплощинах відносно заданої прямої.

4.18. Написати умову того, що дві прямі, задані трійками коефіцієнтів рівняння *ax*+*by*+*c* 0: а) збігаються; б) паралельні; в) паралельні й не збігаються; г) перетинаються;

д) перпендикулярні.

4.19. Дійсні змінні **a** та **b** задають коефіцієнти рівняння *ax*+*b* 0. Написати умову того, що воно:

а) має єдиний розв'язок; б) не має жодного розв'язку;

в) має нескінченно багато розв'язків.

4.20. Дійсні змінні **a**, **b**, **c** задають коефіцієнти рівняння *ax*2+*bx*+*c* 0. Написати умову того, що воно: а) має рівно два дійсні корені;

б) має єдиний дійсний корінь;

в) не має жодного розв'язку;

г) має нескінченно багато розв'язків.

4.21. Написати умову того, що прямокутну цеглину з довжинами ребер *a*, *b*, *c*можна просунути в прямокутне вікно *x* на *y*так, щоб її грані були паралельні сторонам вікна.

##### 4.1.4. Операція розгалуження

Мова С++ містить операцію **?:**, яка називається **операцією розгалуження**, або **умовною**, і використовує умови. Вираз розгалуження виглядає так: **вираз\_1 ? вираз\_2 : вираз\_3**

За його виконання обчислюється значення першого виразу та зводиться до логічного типу. Якщо це **true**, то значенням виразу розгалуження буде значення другого виразу, інакше – третього виразу. В обох випадках значення перетворюється до типу, більшого з типів другого й третього виразів.

Пріоритет операції розгалуження нижче ніж у логічних операцій, але вище ніж у присвоювання.

#### Приклади

1. Значенням виразу **(x>=0?x:-x)** є модуль числового значення змінної **x**.
2. Значенням виразу **(a<b?a:b)** є мінімальне зі значень числових змінних **a** та **b**. 

**Вправа 4.22**. Написати вираз, значенням якого є максимальне зі значень числових змінних **a** та **b**.

### 4.2. Інструкції розгалуження

Рівняння *ax*+*b* 0, залежно від конкретних значень *a*, *b*, може розв'язуватися одним із трьох способів. Спосіб обирається після визначення, чи справджується умова *a*  0; якщо це не так – то чи дійсна умова *b* 0. Розглянемо засоби, що дозволяють указати вибір способу обчислень залежно від тих або інших умов, і скористаємося ними для розв'язання рівняння.

Вибір одного з двох можливих шляхів обчислення можна задати за допомогою **інструкції розгалуження** (**умовної інструкції**). Інструкція розгалуження в **повній формі** має вигляд **if (*умова*) *інструкція1* else *інструкція2***

Слова **if** та **else** є зарезервованими, дужки навколо умови обов'язкові. Цій інструкції відповідає блок-схема на рис. 4.1, а. Інструкція виконується так. Обчислюється значення умови. Якщо це **true**, то виконується ***інструкція1*** і виконання закінчується. Якщо ж це **false**, то виконується ***інструкція2***, записана після **else**. Кожна з цих інструкцій може бути присвоюванням, розгалуженням або інструкцією іншого вигляду.

**Скорочена форма** інструкції розгалуження: **if (*умова*) *інструкція***

Якщо обчислення умови дає значення **false**, то виконання інструкції розгалуження закінчується (див. блок-схему на рис. 4.1, б), інакше виконується ***інструкція***.

*інструкція*

1

*інструкція*

2

**true**

**f**

**alse**

*умова*

**а**

**)**

**б**

**)**

*інструкція*

**t**

**rue**

**f**

**alse**

*умова*

#### Рис. 4.1. Блок-схеми двох форм інструкції розгалуження

**Приклад**. Розглянемо такі інструкції:

**int n, z; cin >> n;**

**if (n%2==0) z=1; else z=-1;**

Якщо введено невід'ємне значення **n**, то обчислення виразу **n%2==0** (перевірка умови) дає значення **true**, і змінна **z** отримує значення **1**. Якщо ж уведене значення від'ємне, то вираз **n%2==0** має значення **false**, і значенням **z** стає **-1**.

Замість інструкції **if (n%2==0) z=1; else z=-1;** можна записати такі: **z=-1; if (n%2==0) z=1;**

Якщо значення **n** непарне, то значенням **z** залишиться **-1**.

Зазначимо: наведений фрагмент коду обчислює значення виразу (-1)*n* і присвоює його змінній **z**.

Щоб програма легше сприймалася, інструкцію розгалуження часто записують так:

**if (*умова*)** Або **if (*умова*) *інструкція1*  *інструкція1* else *інструкція2*** **else**

***інструкція2***

**Приклад 4.1.** 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*; за умови *a* 0, якщо *b* 0, то рівняння має нескінченно багато розв'язків, інакше – жодного.

Отже, усі вхідні дані коректні, тому обробка помилок не потрібна.

**Проектування програми**. Найчастіше алгоритм створюють, поступово уточнюючи поняття, пов'язані із задачею, і необхідні дії. Тоді кажуть, що розробку ведуть **згори донизу**. У загальному вигляді алгоритм такий:

1. Отримати вхідні дані. 2. Обробити вхідні дані.

1. Вивести результат обробки.

Уточнимо кожен із кроків алгоритму.

"Отримати вхідні дані".

* 1. Вивести запрошення на введення даних.
  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*. В іншому випадку, якщо *b* 0, то множина розв'язків нескінченна, інакше кількість дорівнює 0. Кількість розв'язків присвоїмо цілій змінній **n**, при цьому значення **-1** зображуватиме нескінченну множину розв'язків. Розв'язок запам'ятаємо в дійсній змінній **x**.

"Вивести результат обробки".

* 1. Вивести рівняння, уведене користувачем.
  2. За допомогою значень змінних **n** та **x** вивести кількість розв'язків рівняння й розв'язок, якщо він один. Нарешті, можна кодувати.

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

**#include <iostream> using namespace std; int main(){ double a=0, b=0;**

**int n; //кількість розв'язків рівняння ax+b=0;**

**// -1 позначає "нескінченно багато" 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;**

**// обробити введені дані if (a!=0) n=1; else if (b==0) n=-1; else n=0; if (n==1) x=(-b)/a; // повідомити результат cout<<"Equation "<<a<<"x"; if(b>=0) cout<<"+"<<b; else cout<<b; cout<<"=0 ";**

**if (n==1) cout<<"has one solution "<<x<<endl; else if (n==0) cout<<"has no solution\n"; else cout<<"has each real as a solution)\n"; system("pause"); return 0;**

**} prog007.cpp** 

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 кількість розв'язків, то перевірки умов скоротилися б удвічі. 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.

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

*x*, *якщо* 3 *x*  7,



*x*  2, *якщо x*  7,

*f x*( )  

2*x*, *якщо* 5  *x*  3, 0, *якщо x*  5.

По-перше, перепишемо формулу обчислення *f*(*x*) у еквівалентному вигляді. 0, *якщо x*  5,



2*x*, *якщо* 5  *x*  3, *f x*( )  

*x*, *якщо* 3 *x*  7, *x*  2, *якщо* 7  *x*.

За формулою запишемо 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*) ...**

#### Вправи

4.23. Що виводить програма, якщо введено: а) 1; б) 2; в) 3; г) 4?

**#include <iostream> using namespace std; int main(){ int x,y;**

**cout<<"Enter one integer:"; cin >> x;**

**if (x==1) y=16; else if (x==2) y=256; else if (x==3) y=4096; else y=10000; cout << y <<endl; system("pause"); return 0;**

**}**

4.24. Касиру потрібно видати деяку цілу суму грошей *n* монетами номіналом по 5 і 2 (яких є необмежений запас) і витратити якомога менше монет. Написати програму, що обчислює й виводить кількості монет номіналом 5 і 2, які має видати касир. Якщо суму видати неможливо, то вивести відповідне повідомлення.

### 4.3. Блок

Щоб написати кілька інструкцій там, де за правилами мови має бути одна, наприклад, як гілку в умовній інструкції, використовують **блок** – послідовність інструкцій у дужках **{}**. Він має такий загальний вигляд:

**{ *інструкція***

**... *інструкція* }**

Виконання блоку полягає в послідовному виконанні інструкцій, записаних у ньому.

**Приклад**. У прикладі 4.1 (див. с. 74), обробляючи введені дані, кількість розв'язків рівняння й розв'язок можна обчислити разом в одному блоці (саме це відповідає математичній розробці алгоритму розв'язання).

**if (a!=0) {n=1; x=(-b)/a;} else if (b==0) n=-1; else n=0;**



### 4.4. Область дії оголошення імені

Тіло головної, як і будь-якої іншої функції, є блоком. Блок містить послідовність інструкцій, які, у свою чергу, можуть містити блоки і т. д. У кожному блоці можна оголошувати імена змінних і деяких інших об'єктів. Виникає питання про те, в яких місцях програми діє те чи інше оголошення, а в яких ні.

**Область дії оголошення імені** – це сукупність місць у програмі, в яких це ім'я позначає саме те (змінну або інший об'єкт), що описано в оголошенні.

Область дії оголошення визначається таким правилом.

Оголошення діє від місця запису до кінця блоку, в якому записане. Проте, якщо всередині цього блоку є ще один блок (**вкладений**) і в ньому оголошене це саме ім'я, то це внутрішнє оголошення діє до кінця вкладеного блоку. Іншими словами, оголошення імені в блоці "перекриває" оголошення цього ж імені за межами блоку.

Зовнішнє оголошення (розташоване за межами будь-якого блоку) діє за цими самими правилами від місця запису до кінця файлу, в якому воно записане.

Правило, що визначає область дії оголошення імені, дозволяє в різних частинах програми (точніше, у різних блоках) давати різним змінним однакові імена. Однак жодне ім'я змінної не може бути оголошене в блоці більше одного разу.

**Приклад**. Розглянемо програму.

**#include <iostream> using namespace std;**

**int a=99; // "зовнішня" змінна int main(){**

**cout << a << ' '; // вихід: 99 int a=1; // змінна в блоці функції { // вкладений блок: початок cout << a << ' '; // вихід: 1 int a=2; // ще одна змінна cout << a << ' '; // вихід: 2**

**} // вкладений блок: кінець cout << a << endl; // вихід: 1 system("pause"); return 0;**

**}**

**prog008.cpp**

Під час її виконання буде виведено **99 1 2 1.** 

Змінна, визначена зовнішнім оголошенням, є глобальною та статичною. З погляду C++, змінні, оголошені за межами блоків, є **глобальними** (у межах файлу). **Статичними** називаються змінні, що створюються один раз на початку виконання програми та знищуються в кінці, тобто існують протягом усього виконання програми.

Не варто всі змінні програми робити глобальними (хоча це й можливо). Кожна частина програми повинна мати доступ тільки до тих змінних, які їй необхідні. Це дозволяє уникати їх неправильного використання (наприклад, коли змінні, що зображують певні сутності, "раптом" стають допоміжними в обчисленнях). Глобальні змінні найчастіше призначаються для збереження загальних налаштувань програми або її окремих частин. У сучасному програмуванні рекомендується за можливістю *уникати глобальних змінних*, а для обміну даними між частинами програми застосовувати інші засоби.

#### Вправи

4.25. Що буде надруковано за програмою, якщо ввести: а) 0; б) 1?

**#include <iostream> using namespace std; int main(){ int a, b=99, c=777; cin >> a; if(a)**

**{ int b=1; cout << b << ' ' << c << ' '; } else**

**{ c=0; cout << b << ' ' << c << ' '; } cout << b << ' ' << c << '\n'; system("pause"); return 0;**

**}**

4.26. Що буде надруковано за програмою, якщо ввести: а) 0; б) 1?

**#include <iostream> using namespace std; int b=99; int main(){ int a, c=777; cin >> a; if(a)**

**{ int b=22; cout << b << ' ' << c << ' '; } else**

**{ c=0; cout << b << ' ' << c << ' '; } cout << a << ' ' << c << '\n'; system("pause"); return 0;**

**}**

### 4.5. Вибір із кількох варіантів

У ситуації, коли варіант шляху обчислень визначається одним із кількох значень цілого або символьного типу, обчислення можна описати за допомогою **інструкції вибору варіанта**, або **перемикача**. Розглянемо її на прикладі.

**Приклад**. Розв'яжемо задачу: увести з клавіатури дійсне число, знак операції (**+**, **-**, **\*** або **/**), ще одне число й надрукувати результат застосування операції до цих чисел. Наприклад, після введення **2**, **+**, **3** виводиться **5**, а після **2**, **/**, **3** – приблизно **0.667**.

Для спрощення припустимо, що користувач програми правильно вводить числа, але може набрати недопустимий знак операції. Ще однією помилкою може бути операнд **0** після знака **/**, тобто спроба поділити на 0.

Операнди зобразимо дійсними змінними **op1** та **op2**, знак операції – символьною **sign**. Результат операції збережемо в дійсній змінній **res**. Нехай ціла змінна **state** зображує стан процесу обчислення:

* за коректних вхідних даних її значенням є **0** (відсутність помилки);
* якщо введено недопустимий знак операції, то її значенням стає **1**;
* якщо задано ділення на 0, то її значенням стає **2**.

Спочатку, коли вхідних даних ще немає, то немає й помилки (не будемо наперед звинувачувати користувача ), тому початковим значенням **state** є **0**. Уведемо операнди та знак. Потім за значенням **sign** виберемо одну з операцій (додавання, віднімання, множення або ділення) і виконаємо обчислення, одночасно визначаючи стан процесу обчислення. У кінці повідомимо результати обробки. Вибір варіанта дій за знаком операції спочатку опишемо ланцюжком розгалужень.

**// програма обчислення результату операції**

**// за її знаком (+,-,\*,/) і операндами**

**#include <iostream> using namespace std; int main(){ double op1, op2, res; char sign; int state=0; // уведення**

**cout << "Enter double, sign (+,-,\*,/)," <<**

**" and double\n"; cin >> op1 >> sign >> op2;**

**// обчислення if (sign=='+') res=op1+op2; else if (sign=='-') res=op1-op2; else if (sign=='\*') res=op1\*op2; else if (sign=='/') if (op2!=0) res=op1/op2; else state=1; else state=2; // повідомлення результату if (state==0) cout << "===\n" << res << endl; else if (state==1) cout << "Division by zero\n"; else cout << "Wrong operator\n"; system("pause"); return 0;**

**}**

**prog009.cpp**

Розглянемо інший опис вибору варіанта обчислення. Розгалуження за значеннями **sign** опишемо так:

**switch(sign){ case '+': res=op1+op2; break; case '-': res=op1-op2; break; case '\*': res=op1\*op2; break; case '/': if (op2!=0) res=op1/op2; else state=1; break; default: state=2;**

**}**

Аналогічно запишемо інструкцію повідомлення:

**switch(state) { case 0: cout << "===\n" << res<<endl; break; case 1: cout << "Division by zero\n"; break; default: cout << "Wrong operator\n";**

**}**



Слова **switch**, **case**, **default** є зарезервованими; вони позначають відповідно "перемикач", "випадок", "за відсутності". Інструкція **break;** закінчує виконання інструкції, в якій її записано (тут – уся інструкція-перемикач). Вираз у дужках після слова **switch** (у прикладі це ім'я **sign**) називається **селектором варіантів**.

За інструкцією вибору варіанта спочатку обчислюється значення селектора. Потім воно послідовно порівнюється зі значеннями (**мітками варіантів**), указаними після слів **case**. Тільки-но значення селектора збігається з міткою, виконується послідовність інструкцій, записана після цієї мітки та двокрапки, до найближчої інструкції **break**, яка закінчує виконання всієї інструкції вибору варіанта. Якщо значення селектора не збіглося з жодною міткою варіанта, то виконуються інструкції, записані після слова **default**, а якщо слова **default** немає, то виконання перемикача закінчується.

Селектор варіантів є виразом цілого (*integral*) типу або такого, який однозначно перетворюється до цілого, мітки варіантів – константними значеннями того самого типу, що й у селектора.

Варіант з міткою **default** можна записати будь-де, але рекомендується записувати його останнім.

Послідовність інструкцій після мітки варіанта може бути порожньою. За збігу значення селектора з цією міткою виконуються найближчі інструкції після наступних міток (або нічого не виконується, якщо мітка остання).

Якщо деякий варіант не містить інструкції **break**, то після інструкцій цього варіанта виконуються інструкції, записані в подальших варіантах – до найближчого **break** або до кінця перемикача. В останньому варіанті писати **break** немає сенсу.

#### Вправи

4.27. Описати, як вихід залежить від значення змінної **char c**, коли виконується така інструкція:

**switch(c) { case 'A': cout << 'A'; break; default: cout << '\*';**

**case 'B': case 'C': cout << "[B or C]"; }**

1. Принцип ледачих обчислень: "Обчислюю тільки те, що треба". Девізом же енергійних обчислень є: "Обчислюю все, що можу". [↑](#footnote-ref-1)