fit@hcmus

Flow of Control

1

fit@hcmus

Contents

- Boolean Expression
- Control Structures
- `if .. else` Statements
- `switch` Statement
- `while` Structure
- `do .. while` Structure
- `for` Structure

2

fit@hcmus

Boolean Expressions

3

fit@hcmus

Boolean Expression

- Boolean expression: an expression that is either ***true*** or ***false***.
- Comparison Operators: == , ! =, <, <=, >, >=

4

fit@hcmus

Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	= =	x + 7 = = 2*y	x + 7 = 2y
≠	Not equal to	!=	ans != 'n'	*ans* ≠ *'n'*
<	Less than	<	count < m + 3	*count* < *m* + 3
≤	Less than or equal to	<=	time <= limit	*time* ≤ *limit*
>	Greater than	>	time > limit	*time* > *limit*
≥	Greater than or equal to	>=	age >= 21	*age* ≥ 21

5

fit@hcmus

Building Boolean Expressions

- Combine two comparisons using:
  - AND operator: **&&**

    ```
    ( Boolean_Exp_1) && (Boolean_Exp_2)
    ```

  - OR operator: **||**

    ```
    ( Boolean_Exp_1) || (Boolean_Exp_2)
    ```

- Negate a Boolean expression, NOT operator: **!**

  ```
  ! ( Boolean_Exp)
  ```


6

Evaluating Boolean Expressions

- Be evaluated the same way as arithmetic expressions.
- Examples:

```
bool result = (x < z) && (z < y);
!( (y < 3) || (y > 7))
```

- Rules: **truth tables**.
- `true`, `false`: predefined constants of type `bool`.

7

Precedence Rules

- Boolean expression need not be fully parenthesized.

- Default precedence:
  - First: !
  - Next: relational operations (<, >,..)
  - Next: &&
  - Next: ||
- Examples:
  - `(temperature > 80) && (humidity > 0.9) && (poolGate == OPEN)`
  - `temperature > 80 && humidity > 0.9 && poolGate == OPEN`

8

Precedence of Operators

`::`	Scope resolution operator
`.`	Dot operator
`->`	Member selection
`[]`	Array indexing
`( )`	Function call
`++`	Postfix increment operator (placed after the variable)
`--`	Postfix decrement operator (placed after the variable)
`++`	Prefix increment operator (placed before the variable)
`--`	Prefix decrement operator (placed before the variable)
`!`	Not
`-`	Unary minus
`+`	Unary plus
`*`	Dereference
`&`	Address of
`new`	Create (allocate memory)
`delete`	Destroy (deallocate)
`delete []`	Destroy array (deallocate)
`sizeof`	Size of object
`( )`	Type cast
`*`	Multiply
`/`	Divide
`%`	Remainder (modulo)
`+`	Addition
`-`	Subtraction
`<<`	Insertion operator (console output)
`>>`	Extraction operator (console input)

Highest precedence (done first)

Lower precedence (done later)

9

Precedence of Operators

`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equal
`!=`	Not equal
`&&`	And
`||`	Or
`=`	Assignment
`+=`	Add and assign
`-=`	Subtract and assign
`*=`	Multiply and assign
`/=`	Divide and assign
`%=`	Modulo and assign
`? :`	Conditional operator

Lowest precedence (done last)

10

fit@hcmus

Precedence Rules

- **short-circuit evaluation**: if it gets enough information to determine the value, it does not bother to evaluate other expressions.
  - C++ uses this way.
- **complete evaluation**: all the expressions are evaluated before determining.
  - Some of the languages.
- Examples:

```
(age >= 21) || ( x == 5) //Line 1
(grade == 'A') && (x >= 7)     //Line 2
```


11

fit@hcmus

Control Structures

12

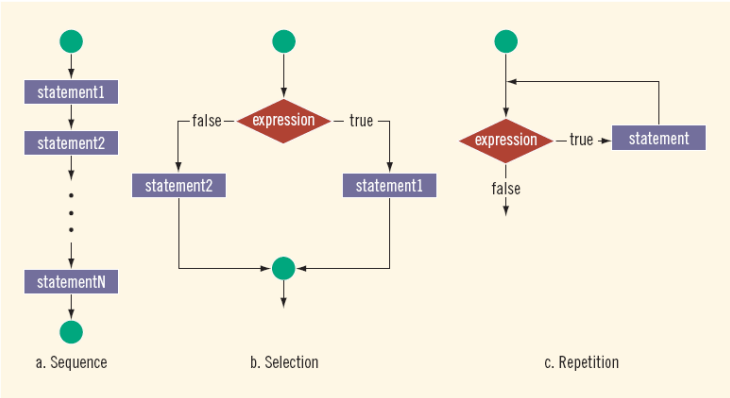
fit@hcmus

Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) - looping
- Some statements are executed only if certain conditions are met
- A condition is met if it evaluates to `true`

fit@hcmus

Control Structures



Selection Structures

- `if-else` statements
- `switch-case` statements

15

Repetition Structures

- `while ... do`
- `do ... while`
- `for`

16

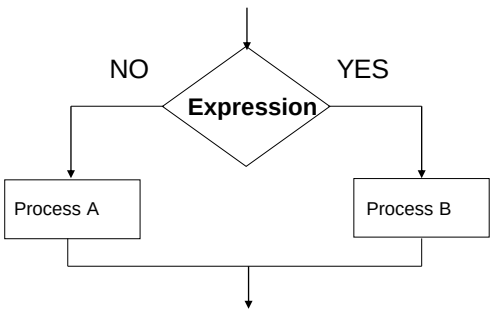
fit@hcmus

if-else Statements

17

fit@hcmus

if-else Statements



18

fit@hcmus

One-Way Selection

- Syntax:

```
if (Boolean_Expression)
   Yes_Statement
```

- The *Yes_Statement* is executed if the value of the *Boolean_Expression* is `true`
- The statement is bypassed if the value is `false`; program goes to the next statement.
- Example:

```
if (score >= 5.0)
    pass = true;
```


19

fit@hcmus

One-Way Selection

```
#include <iostream>

using namespace std;

int main()
{
    int number, temp;

    cout << "Line 1: Enter an integer: ";          //Line 1
    cin >> number;                                 //Line 2
    cout << endl;                                  //Line 3

    temp = number;                                 //Line 4

    if (number < 0)                                //Line 5
        number = -number;                          //Line 6

    cout << "Line 7: The absolute value of "
         << temp << " is " << number << endl;      //Line 7

    return 0;
}
```


20

One-Way Selection

- Some error examples:

```
//Syntax
• if score >= 60
  grade = 'P’;

//Logical
• if (score >= 60);
  grade = 'P';
```

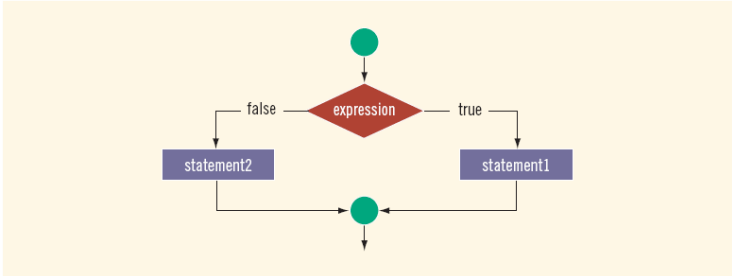

Two-Way Selection

- Two-way selection takes the form:

```
if (Boolean_Expression)
   Yes_Statement
else
   No_Statement
```

- If *Boolean_Expression* is `true`, `Yes_Statement` is executed; otherwise, `No_Statement` is executed
  - `Yes_Statement` and `No_Statement` are any C++ statements

Two-Way Selection



23

Two-Way Selection

```
if (hours > 40.0)                          //Line 1
    wages = 40.0 * rate +
            1.5 * rate * (hours - 40.0);   //Line 2
else                                       //Line 3
    wages = hours * rate;                  //Line 4
```


24

Compound Statements

- A list of statements enclosed in a pairs of braces.
- Compound statement (block of statements):

```
{
   Statement_1;
   Statement_2;

   ...
   Statement_n;
}
```

- A compound statement is a single statement.

25

Compound Statements

- Example:

```
if (age > 18)
{
   std::cout << "Eligible to vote." << std::endl;
   std::cout << "No longer a minor." << std::endl;
}
else
{
   std::cout << "Not eligible to vote." << std::endl;
   std::cout << "Still a minor." << std::endl;
}
```


26

fit@hcmus

Multiple Selections: Nested `if`

- **Nesting:** one control statement in another
- An `else` is associated with the most recent `if` that has not been paired with an `else`
- Syntax:

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
...
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```


fit@hcmus

Multiple Selections: Nested `if`

```
if (balance > 50000.00)              //Line 1
    interestRate = 0.07;             //Line 2
else                                 //Line 3
    if (balance >= 25000.00)         //Line 4
        interestRate = 0.05;         //Line 5
    else                             //Line 6
        if (balance >= 1000.00)      //Line 7
            interestRate = 0.03;     //Line 8
        else                         //Line 9
            interestRate = 0.00;     //Line 10
```

To avoid excessive indentation, the code in Example 4-18 can be rewritten as follows:

```
if (balance > 50000.00)              //Line 1
    interestRate = 0.07;             //Line 2
else if (balance >= 25000.00)        //Line 3
    interestRate = 0.05;             //Line 4
else if (balance >= 1000.00)         //Line 5
    interestRate = 0.03;             //Line 6
else                                 //Line 7
    interestRate = 0.00;             //Line 8
```

Confusion between == and =

- C++ allows you to use any expression that can be evaluated to either `true` or `false` as an expression in the `if` statement:

```
if (x = 5)
  std::cout << "The value is five." << std::endl;
```

- The appearance of = in place of == resembles a *silent killer*
  - It is not a syntax error
  - It is a **logical error**

30

Conditional Operator (?:)

- Conditional operator (`?:`) takes three arguments
  - Ternary operator
- Syntax for using the conditional operator:

```
expression1 ? expression2 : expression3
```

- If `expression1` is `true`, the result of the conditional expression is `expression2`
  - Otherwise, the result is `expression3`

31

fit@hcmus

`switch` Statement

32

fit@hcmus

`switch` Statement

- `switch` structure: alternate to `if-else`
- `switch` (integral) expression (*controlling expression*) is evaluated first.
- Value of the controlling expression determines which corresponding action is taken.

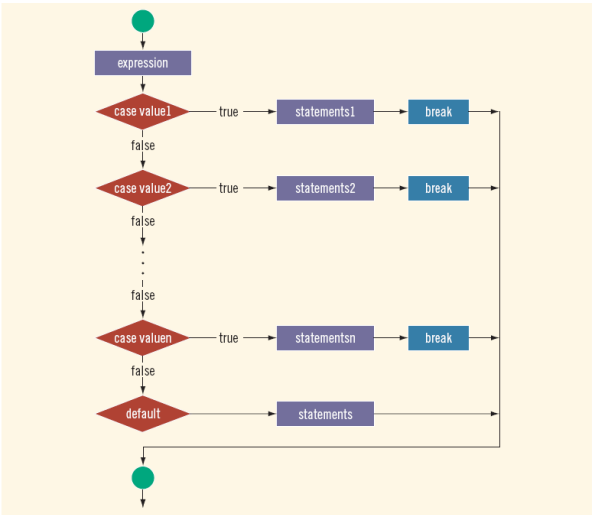
33

fit@hcmus

switch Statement

```
switch (Controlling_Expression)
{
        case Constant_1:
                Statement_Sequence_1
                break;
        case Constant_2:
                Statement_Sequence_n
                break;
        ...
        case Constant_n:
                Statement_Sequence_n
                break;
        default:
                Default_Statement_Sequence
}
```


34



35

fit@hcmus

`switch` Statement

- One or more statements may follow a **`case`** label.
- Braces are not needed to turn multiple statements into a single compound statement.
- The `break` statement may or may not appear after each statement.
- `switch`, `case`, `break`, and `default` are reserved words.

fit@hcmus

Enumeration Types

- Enumeration type: a type whose values are defined by a list of constants type `int`.
- Syntax:

```
enum Name {enumerator_1, enumerator_2,...};
enum Name {enumerator_1 = constant_1, enumerator_2 =
constant_2,...};
```

- Examples:
  - `enum Direction {NORTH, SOUTH, EAST, WEST};`
  - `enum Direction {NORTH=0, SOUTH=1, EAST=2, WEST=3};`
  - `enum MyNum {ONE = 17, TWO, THREE, FOUR = -3, FIVE}`

fit@hcmus

Enumeration Types

- Example:

```
enum Color { red, green, blue };
Color r = red;
switch(r)
{
    case red  : std::cout << "red\n";   break;
    case green: std::cout << "green\n"; break;
    case blue : std::cout << "blue\n";  break;
}
```


40

fit@hcmus

`assert` function

Terminating a Program with the `assert` Function

41

fit@hcmus

Terminating a Program with the `assert` Function

- Certain types of errors that are very difficult to catch can occur in a program
  - Example: division by zero can be difficult to catch using any of the programming techniques examined so far
- The predefined function, `assert`, is useful in stopping program execution when certain elusive errors occur.

fit@hcmus

The `assert` Function

- Syntax:

```
assert(expression);
```

`expression` is any logical expression

- If `expression` evaluates to `true`, the next statement executes
- If `expression` evaluates to `false`, the program terminates and indicates where in the program the error occurred
- To use `assert`, include `cassert` header file.

fit@hcmus

The `assert` Function

- `assert` is useful for enforcing programming constraints during program development.
- After developing and testing a program, remove or disable *assert* statements.
- The preprocessor directive `#define NDEBUG` must be placed before the directive `#include <cassert>` to disable the assert statement.

fit@hcmus

Programming Example

fit@hcmus

Cable Company Billing

- This programming example **calculates a customer's bill for a local cable company.**
- There are two types of customers:
  - Residential
  - Business
- Two rates for calculating a cable bill:
  - One for residential customers
  - One for business customers

46

fit@hcmus

Cable Company Billing - Rates

- For residential customer:
  - Bill processing fee: $4.50
  - Basic service fee: $20.50
  - Premium channel: $7.50 per channel
- For business customer:
  - Bill processing fee: $15.00
  - Basic service fee: $75.00 for first 10 connections and $5.00 for each additional connection
  - Premium channel cost: $50.00 per channel for any number of connections

47

Cable Company Billing - Requirements

- Ask user for account number and customer code
- Assume `R` or `r` stands for residential customer and `B` or `b` stands for business customer.
- The program should print the billing amount to two decimal places

48

Cable Company Billing - Input/Output

- Input:
  - Customer account number
  - Customer code
  - Number of premium channels
  - For business customers, number of basic service connections
- Output:
  - Customer's account number
  - Billing amount

49

fit@hcmus

Repetition Structures

50

fit@hcmus

Why Is Repetition Needed?

- Repetition allows you to efficiently use variables
- Can input, add, and average multiple numbers using a limited number of variables.
- For example, to add five numbers:
  - Declare a variable for each number, input the numbers and add the variables together
  - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

51

fit@hcmus

Repetition Structures

- Three loop statements:
  - `while ... do`
  - `do ... while`
  - `for`
- **loop body**: the code repeated in a loop.
- **iteration**: each repetition of a loop.
- **Infinite loop:** continues to execute endlessly
  - Avoided by including statements in loop body that assure exit condition is eventually `false`

52

fit@hcmus

`while` Structures

53

fit@hcmus

while Structure

- Single-statement body:

```
while (expression)
    statement
```

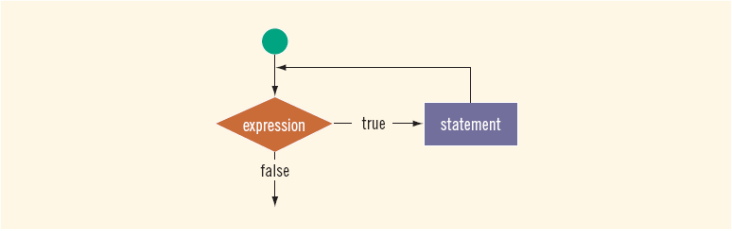
- Multi-statement body:

```
while (expression)
{
    Statement_1
    Statement_2
    ...
    Statement_Last
}
```


54

fit@hcmus

while Structure



55

while Structure

- Example:

```
int i = 0;
while (i <= 20)
{
     std::cout << i << " ";
     i = i + 5;
}
```

- Example:

```
int i = 20;
while (i < 20)
{
     std::cout << i << " ";
     i = i + 5;
}
```


56

do..while Structure

- Single-statement body:

```
do
    Statement
while (Boolean_Expression);
```

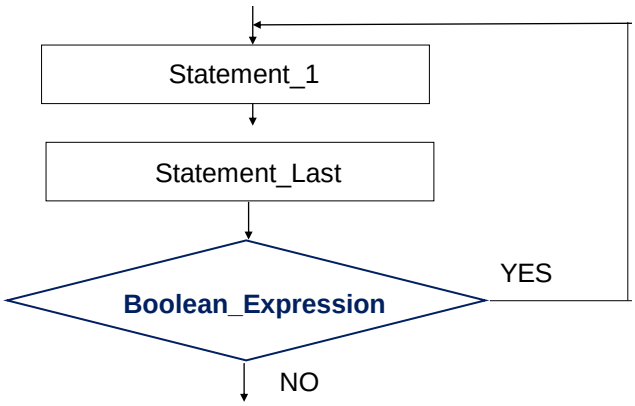
- Multi-statement body:

```
do
{
    Statement_1
    Statement_2
    ...
    Statement_Last
} while (Boolean_Expression);
```


57

fit@hcmus

do..while Structure



58

fit@hcmus

do..while Structure

- Example:

```
int i = 0;
do
{
     std::cout << i << " ";
    i = i + 5;
} while (i <= 20);
```

- Example:

```
int i = 20;
do
{
     std::cout << i << " ";
    i = i + 5;
} while (i < 20);
```


59

fit@hcmus

Counter-Controlled `while` Loops

- If you know exactly how many pieces of data need to be read, the `while` loop becomes a counter-controlled loop.

```
counter = 0;          //initialize the loop control variable

while (counter < N) //test the loop control variable
{
    .
    .
    .
    counter++;        //update the loop control variable
    .
    .
    .
}
```


60

fit@hcmus

Sentinel-Controlled `while` Loops

- *Sentinel variable* is tested in the condition and loop ends when sentinel is encountered.

```
cin >> variable;              //initialize the loop control variable

while (variable != sentinel)  //test the loop control variable
{
    .
    .
    .
    cin >> variable;        //update the loop control variable
    .
    .
    .
}
```

- Example:
  - Read some positive numbers and average them, but you do not have the preset number of data items in mind. Suppose ***-999*** marks the end of the data.

61

Flag-Controlled `while` Loops

- A flag-controlled `while` loop uses a `bool` variable to control the loop.
- The flag-controlled `while` loop takes the form:

```
found = false;       //initialize the loop control variable

while (!found)       //test the loop control variable
{
   .
   .
   .
   if (expression)
      found = true; //update the loop control variable
   .
   .
   .
}
```

- Example: **Number guessing game**.

Infinite Loop

- Example:

```
int x = 1;
while (x != 12)
{
   std::cout << x << std::endl;
   x = x + 2;
}
```

fit@hcmus

Infinite Loop

- Example:

```
int x = 1;
while (x != 12)
{
    std::cout << x << std::endl;
   x = x + 2;
}
```

How to fix this loop?

64

fit@hcmus

The Comma Operator

- **Comma operator** (,)
  - evaluates a list of expressions
  - returns the value of the last expression.

- Examples - What are the value of these variables after running:
  - `result = (first = 2, second = first + 1);`
  - `result = (first = 2, second = first + 1, first = second + 7);`
  - `result = ((first = 2, second = first + 1), third = second + 1);`

65

fit@hcmus

for Structure

66

fit@hcmus

for Structure

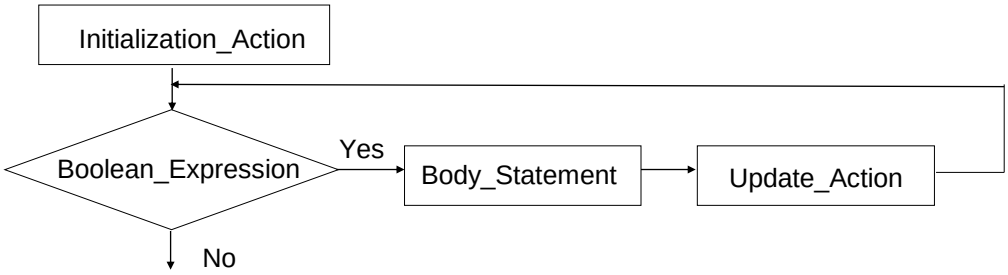
- The general form of the `for` statement is:

```
for (Initialization_Action; Boolean_Expression; Update_Action)
  Body_Statement
```

- The *Initialization_Action*, *Boolean_Expression*, and *Update_Action* are called `for` loop control statements
  - `Initialization_Action` usually initializes a variable (called the `for` **loop control**, or `for` **indexed**, **variable**)
- In C++, `for` is a reserved word

67

for Structure




```
for (Initialization_Action; Boolean_Expression; Update_Action)
        Body_Statement
```


68

Examples

- Example:

```
for (int i = 0; i < 10; i ++)
    std::cout << i << " ";

std::cout << endl;
```


69

fit@hcmus

Examples

- Example:

```
int i = 0;
for ( ; i < 10; i ++)
    std::cout << i << " ";

std::cout << std::endl;
```


70

fit@hcmus

Examples

- **Example:**

```
for (int i = 1; i <= 5; i++)
{
    std::cout << "Hello !" << std::endl;
    std::cout << "*" << std::endl;
}
```

- **Example:**

```
for (int i = 1; i <= 5; i++)
    std::cout << "Hello !" << std::endl;
std::cout << "*" << std::endl;
```


71

fit@hcmus

Examples

- Example:

```
for (int i = 1; i <= 5; i++);
    std::cout << "Hello !" << std::endl;
```

- Example:

```
for ( ; ; )
    std::cout << "Hello !" << std::endl;
```


72

fit@hcmus

Examples

- Example:

```
for (int i = 10; i >= 0; i--)
    std::cout << " " << i;
std::cout << std::endl;
```

- Example:

```
for (int i = 1; i <= 20; i = i+2)
    std::cout << " " << i;
std::cout << std::endl;
```


73

fit@hcmus

Choosing the Right Loop Structure

- All three loops have their place in C++
  - If you know or can determine in advance the number of repetitions needed, the **`for`** loop is the correct choice.
  - If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a **`while`** loop.
  - If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a **`do...while`** loop

74

fit@hcmus

Nested Loops

- It is legal to nest one loop statement inside another loop statement.
- Example:

```
for (i = 1; i <= 5 ; i++)
{
          for (j = 1; j <= i; j++)
              std::cout << "*";
         std::cout << std::endl;
}
```


75

fit@hcmus

break and continue

76

fit@hcmus

break and continue Statements

- **break** and **continue** alter the flow of control.
- **break** statement is used for two purposes:
  - To exit early from a loop
    - Can eliminate the use of certain (flag) variables
  - To skip the remainder of the **switch** structure
- After the **break** statement executes, the program continues with the first statement after the structure.

77

`break` and `continue` Statements

- **`continue`** is used in **`while`**, **`for`**, and **`do`**…**`while`** structures.
- When executed in a loop
  - It skips remaining statements and proceeds with the next iteration of the loop

78

Examples

```
1  #include <iostream>
2  using namespace std;

3  int main( )
4  {
5      int number, sum = 0, count = 0;
6      cout << "Enter 4 negative numbers:\n";

7      while (++count <= 4)
8      {
9          cin >> number;

10         if (number >= 0)
11         {
12             cout << "ERROR: positive number"
13                  << " or zero was entered as the\n"
14                  << count << "th number! Input ends "
15                  << "with the " << count << "th number.\n"
16                  << count << "th number was not added in.\n";
17             break;
18         }

19         sum = sum + number;
20     }

21     cout << sum << " is the sum of the first "
22          << (count - 1) << " numbers.\n";

23     return 0;
24 }
```


79

Examples

fit@hcmus

```
1  #include <iostream>
2  using namespace std;

3  int main( )
4  {
5      int number, sum = 0, count = 0;
6      cout << "Enter 4 negative numbers, ONE PER LINE:\n";

7      while (count < 4)
8      {
9          cin >> number;

10         if (number >= 0)
11         {
12             cout << "ERROR: positive number (or zero)!\n"
13                  << "Reenter that number and continue:\n";
14             continue;
15         }
16         sum = sum + number;
17         count++;
18     }

19     cout << sum << " is the sum of the "
20          << count << " numbers.\n";
21     return 0;
22 }
```


80

fit@hcmus

Programming Example

81

Checking Account Balance

- A local bank in your town needs a program to calculate a customer's checking account balance at the end of each month
- Data are stored in a file in the following form:

```
467343 23750.40
W 250.00
D 1200
W 75.00
I 120.74
```


82

Checking Account Balance

- The first line of data shows the account number followed by the account balance at the beginning of the month
- Thereafter each line has two entries:
  - Transaction code
  - Transaction amount
- Transaction codes
  - `W` or `w` means withdrawal
  - `D` or `d` means deposit
  - `I` or `i` means interest paid by the bank

83

Checking Account Balance

- Program updates balance after each transaction
- During the month, if at any time the balance goes below $1000.00, a $25.00 service fee is charged.

84

Checking Account Balance

- Program prints the following information:
  - Account number
  - Balance at the beginning of the month
  - Balance at the end of the month
  - Interest paid by the bank
  - Total amount of deposit
  - Number of deposits
  - Total amount of withdrawal
  - Number of withdrawals
  - Service charge if any

85

fit@hcmus

Checking Account Balance

- **Input:** file consisting of data in the previous format
- **Output** is of the following form:

```
Account Number: 467343
Beginning Balance: $23750.40
Ending Balance: $24611.49
Interest Paid: $366.24
Amount Deposited: $2230.50
Number of Deposits: 3
Amount Withdrawn: $1735.65
Number of Withdrawals: 6
```


86

fit@hcmus

Questions and Answers

87