COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Input-validation loop and the do-while statement

Outline

- Input-validation loop
  - Do-while statement
  - Handling repeated input failures
  - While vs do-while statements

Input-validation loop

Input-validation Loop

- This type of loop is used to *validate* a user input.
  - Loop continues until the user inputs a valid value.
- It keeps on prompting the user for a data value until a valid value is entered.
  - E.g. the user enters a negative number for GPA, or enters a value greater than 31 for some day of a month etc.
- Such kind of loop has the following general structure:

```
Print an initial prompting message.
Get the input value.
while the input value is not valid
    Print a warning and another prompting message.
    Get the input value.
```

Input-validation Loop – example

- Example: input number of hot days observed in previous summer, where hot day means the day on which temperature > 40.
- The loop structure according to the above general form will be:

```
Print an initial prompting message.
Get the number of hot days.
while the number of hot days is negative
    Print a warning and another prompting message.
    Get the number of hot days.
```

Input-validation Loop – example

- What are the initialization, condition and update step in the above example?
  - Initialization: Get the number of hot days.
  - Condition: the number of hot days is negative
  - Update: Get the number of hot days.
- Interestingly, initialization and update steps are usually the same in an input validation loop.
  - To start with: we can use a while loop for this purpose.
- Complete Loop:

```
cout << "Enter number of hot days > ";
Cin >> num_hot;    /* initialization       */
while (num_hot < 0) {
   cout << Negative number invalid; try again> ";
   cin >> num_hot;      /* update           */
}
```

Input-validation Loop – problem

- Do you see any *redundancy* in the general form of the loop?
- The input-validation loop structure:

```
Print an initial prompting message.
Get the number of hot days.
while the number of hot days is negative
    Print a warning and another prompting message.
    Get the number of hot days.
```

- The prompt and input operation are being repeated before and inside the loop.
  - This is because an input operation should be performed at least once before checking the loop condition, so do-while loop is a better choice in this case (which always runs at least once).

Input-validation Loop – problem

- So, we can update the general loop design as follows:
- The input-validation loop structure:

```
do
    Print a generic prompt message.
    Get the number of hot days.
while the number of hot days is negative
```

- The generic prompt message, now, will be the same for first and subsequent inputs.
  - It is a good idea to show valid input range in the prompt message, e.g. “Enter number of hot days (a +ve value): ”

Example: Input Validation loop using the do-while Loop

- Complete code using do-while:

```
do {
    cout << “Enter number of hot days (a +ve value): ”;
    cin >> num_hot;         /* init and update    */
} while (num_hot < 0);      /* condition */
```

`do-while` Loop: Example 2

```
/* continue until even number is input */
do{
  cout << “Enter an even value: ”;
  cin >> num;
}while (num % 2 !=0);
```

This loop will keep on repeating if the user inputs an odd number.

`do-while` Loop: Example 3

```
/* continue until a letter from ‘A’ to ‘E’ is input */
do {
     cout << “Enter a letter (A-E) > ”;
    cin >> letter_choice;
}while(letter_choice < ’A’ || letter_choice > ’E’);
```

Exercise

- Write down input validation loops to input the following values: (identify the valid range of values yourself in each case):
  - Student marks in a subject out of 50
  - Total Number of days in current month
  - A vowel letter
  - Gender of a person (Male/Female)
  - Today's Temperature (in centigrades)

Handling *repeated* input failure

- Remember, we used the ignore and clear function of cin in an if-statement to *detect* the input failure.
- We can also handle the input failure inside the if statement (see Example on next slide)

Handle Input Failure

- The following example handles **1-time** input failure.

```
int main()
{
    int x;
    char f = ‘*’;
    cin>>x;
    if(!cin) // if invalid value for x is entered, e.g. a
char is entered
    {
        cin.clear();
        cin.ignore();
        cout<<"Enter x again: ";
        cin>>x;
    }
    cin>>f;
    cout<<x <<"  "<<f;
    return 0;
}
```

Handling *repeated* input failure

- Note that this example handles only **1-time** input failure.
  - What if the input failure happens repeatedly (more than 1 times)?
- We can use a modified *input-validation loop* to handle the repeated input failure.
  - We now check for the *input stream status* instead of *valid inputs range*.

Handle repeated Input Failure

- The following example handles **repeated** input failure.

```
int main()
{
    int x;
    char f = '*';
    do // if invalid value for x is entered, e.g. a char is entered
    {
        cin.clear();
        cin.ignore();
        cout<<"Enter x: ";
        cin>>x;
    } while(!cin);
    cin>>f;
    cout<<x <<"  "<<f;
    return 0;
}
```

Input Failure + validation

- The following example handles **repeated** input failure and also validates input (positive numbers only) at the same time.

```
int main()
{
    int x;
    char f = ‘*’;
    do {
        cin.clear();
        cin.ignore();
        cout<<"Enter x: ";
        cin>>x;
    } while( !cin || x<0 );
    cin>>f;
    cout<<x <<"  "<<f;
    return 0;
}
```

Difference with while loop

While loop

1. Condition is evaluated before entering the loop body.
2. The loop body will execute first time depending on condition.
   - Result: loop may execute even 0 times, if the condition is false at start
3. No semi-colon anywhere in the while statement
   - Adding semi-colon at the end of while part is a common logical error.
4. Suitable for counter-controlled and sentinel-controlled loops.

Do-while loop

1. Condition is evaluated after entering (and completing) the loop body
2. The loop body will execute first time without any dependence on condition.
   - Result: loop will execute at least 1 time
3. Semi-colon is MUST at the end of do-while statement
   - Missing semi-colon results in compile-time (syntax) error
4. More suitable for input-validation loops.

Similarities with while loop

1. Further execution of loop body (after 1st iteration) depends on the loop condition.
2. Any sort of condition can be specified.
   - Simple or complex
3. The body of both statements can be a single statement or a compound statement.
   - In case of single statement, brackets can be omitted. E.g.
     - do

```
    cout << ++i;
while(i < 10);
```

`break` & `continue` Statements

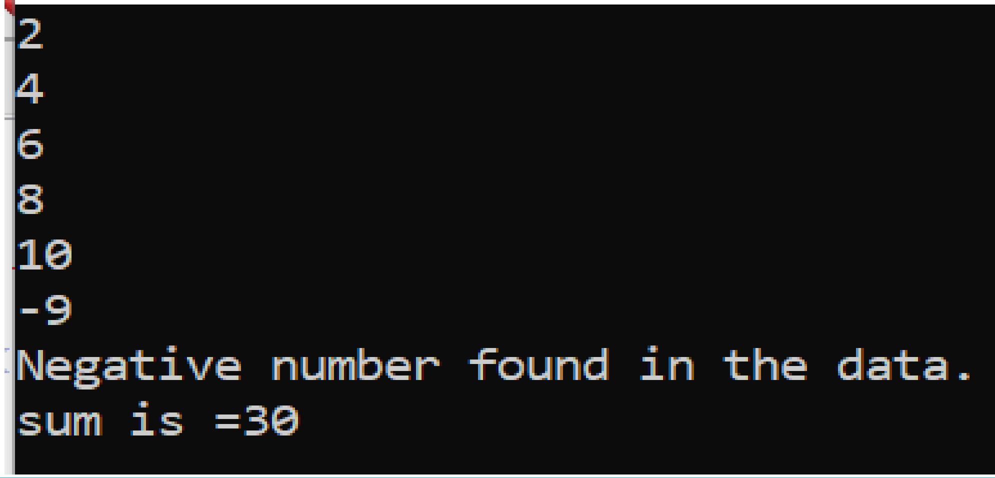
- `break` and `continue` alter the flow of control
- When the `break` statement executes in a repetition structure, it immediately exits
- The `break` statement, in a `switch` structure, provides an immediate exit
- The `break` statement can be used in `while`, `for`, and `do`…`while` loops

`break` & `continue` Statements (continued)

- The `break` statement is used for two purposes:
  1. To exit early from a loop
  2. To skip the remainder of the switch structure
- After the `break` statement executes, the program continues with the first statement after the structure
- The use of a `break` statement in a loop can eliminate the use of certain (flag) variables

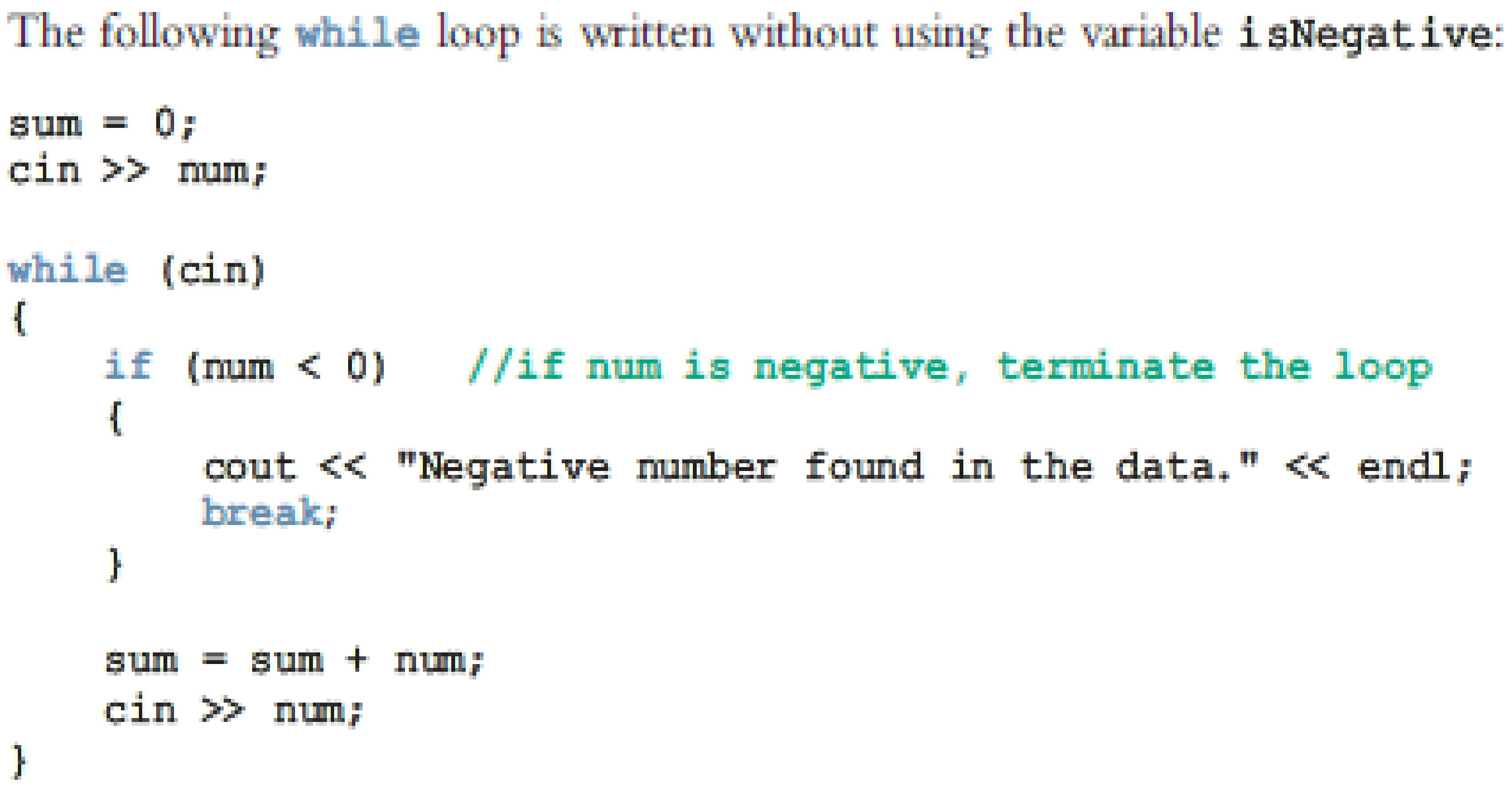
break & continue Statements (continued)

```
sum = 0;
isNegative = false;

cin >> num;

while (cin && !isNegative)
{
    if (num < 0)    //if num is negative, terminate the loop
                    //after this iteration
    {
        cout << "Negative number found in the data." << endl;
        isNegative = true;
    }
    else
    {
        sum = sum + num;
        cin >> num;
    }
}
```



```
2
4
6
8
10
-9
Negative number found in the data.
sum is =30
```

break & continue Statements (continued)



The following `while` loop is written without using the variable `isNegative`:

```
sum = 0;
cin >> num;

while (cin)
{
    if (num < 0)    //if num is negative, terminate the loop
    {
        cout << "Negative number found in the data." << endl;
        break;
    }

    sum = sum + num;
    cin >> num;
}
```

`break` & `continue` Statements (continued)

- `continue` is used in `while`, `for`, and `do…while` structures

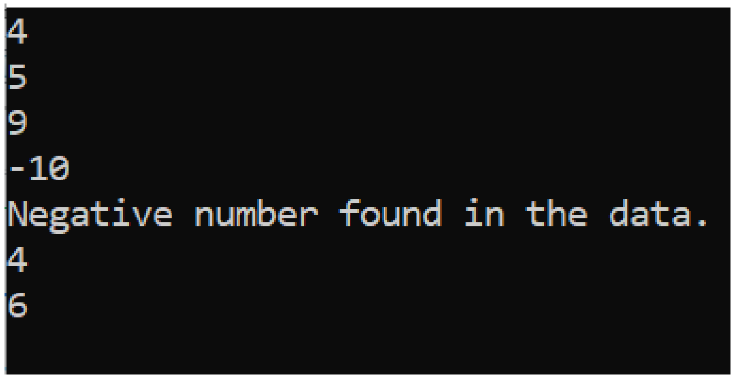
- When executed in a loop
  - It skips remaining statements and proceeds with the next iteration of the loop

`break` & `continue` Statements (continued)

- In a `while` and `do…while` structure
  - Expression (loop-continue test) is evaluated immediately after the continue statement
- In a `for` structure, the update statement is executed after the `continue` statement
  - Then the loop condition executes

break & continue Statements (continued)

```
sum = 0;
cin >> num;

while (cin)
{
    if (num < 0)
    {
        cout << "Negative number found in the data." << endl;
        cin >> num;
        continue;
    }

    sum = sum + num;
    cin >> num;
}
```



```
4
5
9
-10
Negative number found in the data.
4
6
```