Foundation of Computer Science: Loop structure

Kafi Rahman
Assistant Professor
Truman State University
Monday 23 Sep 2019

Objectives

- Introducing inc/dec operators
- Learning loop statement
- Simple while loop statement
- Group work

Puzzled GPS


Canada was definitely running out of ideas for street names.
urie Ln
7
Hwy 7
This St
That St
The Other St

Increment and Decrement Operators

- These operators increase or decrease a value by one.
- One of the common operations in programming is to increment or decrement the value of a variable:
  - x += 1; and x -= 1;
- There are operators just for these operations:
  - the increment operator ++ and
  - the decrement operator --
  - they are unary operators
- Assuming x is a variable with a value:

```
unsigned x = 5;
x++; // now x is 6
x--; // now x is 5 again
```

Increment and Decrement Operators (cont)

- Assuming x is an unsigned variable :

```
unsigned x = 5;
x++;        // now x is 6
cout<<x ; // will display 6
x--;        // now x is 5 again
cout<<x ; // will display 5
```

Prefix and Postfix

- each operator comes in two forms, a prefix form and a postfix form

```
unsigned x = 5;
x++; // now x is 6
++x; // now x is 7
```

- in the example above, both prefix and postfix forms do the same thing

- but they work differently!

Prefix and Postfix (contd.)

```
unsigned x = 5, y = 10;

cout << x++ << endl;
cout << ++y << endl;
```

- The difference is in when they operate
- in each case, number is being used in an expression and also being incremented

Prefix and Postfix (contd.)

```
unsigned x = 5, y = 10;

cout << x++ << endl;
cout << ++y << endl;
```

- in the prefix form
  - the increment happens first
- in the postfix form
  - the increment happens last

Prefix and Postfix (contd.)

```
unsigned x = 5, y = 10;

cout << x++ << endl; //output 5, x is 6
cout << ++y << endl; //output 11, y is 11
```

- in the prefix form
  - the increment happens first
- in the postfix form
  - the increment happens last

Prefix and Postfix (contd.) Another example

```
unsigned foo = 6;
unsigned bar;

bar = foo++;
unsigned door = --foo;
```

- after this code runs
  - bar has the value:
  - door has the value:
  - foo has the value:

Prefix and Postfix (contd.) Another example

```
unsigned foo = 6;
unsigned bar;

bar = foo++;
unsigned door = --foo;
```

- after this code runs
  - bar has the value: 6
  - door has the value: 6
  - foo has the value: 6

Increment and Decrement with Floating Point

- unary increment and decrement also work with floating point value

```
double x = 2.25;
x++; // now x is 3.25
```

- Its controversial whether programmers should use these operators with floating point variables.
  - Use at your own discretion

Increment and Decrement Operators: mixing

- the following code snippets are completely legal:

```
x *= y / ++z;
if (a++ > 10) {}
```

- but they are very confusing and hard to read
  - Avoid using this style!

- a best practice of programming is that each statement or expression should do only one thing
- these are doing two things at once and hence, they should be split into separate statements

Non-Linear Control Flow II

- The focus of chapter 4 was non-linear program flow using the concept of the if statement technically called branching
- The focus of chapter 5 is a different form of non-linear program flow: looping

Loop Structure: Motivation

- Write a program to display the numbers from 1 to 3:

```
int main()
{
        cout<< "1 2 3" <<endl;
}
```

- What about writing a program to display the numbers between 1 and 5 million.
  - Conventional methods of writing the numbers would not work in this case, we need to use the loop.

Loop Structure (cont'd.)

- Three types of loops
  - while
    - The loop-controlling Boolean expression is the first statement
  - for
    - A concise format in which to execute loops
  - do⋯while
    - The loop-controlling Boolean expression is the last statement

loop demo video

The effectiveness of loops in computer programming languages

The while Loop

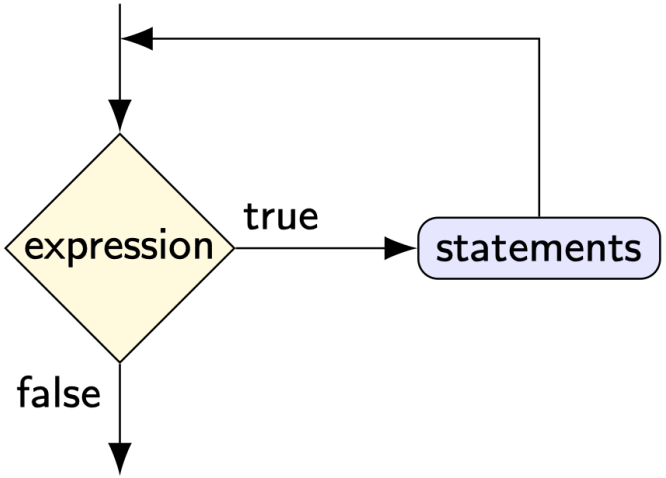
- The while loop is extremely similar in both structure and action to the simple if statement
- The difference in structure is one word

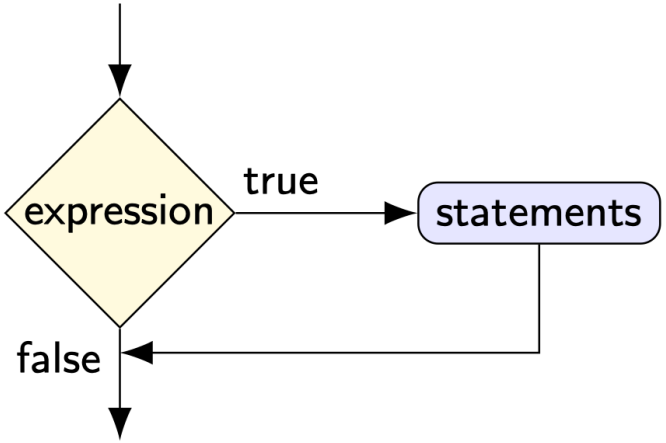
```
while (expression)
{
    statement;
    statement;
    ...
}
```

- All the rules of style and the common mistakes are identical to those for the if statement

Loop Structure: Flow Chart

- flowchart of while is almost identical to if
  - but there is one crucial difference!
- if sends the program forward around a detour
- while sends the program around a detour and then backwards
  - this allows for the program to repeat a block of statements more than once
- looping aka iteration aka repetition





while-loop Statement: Loop Control Variable

```
//Gaddis Program 5-3 page 238
#include <iostream>
using namespace std;

int main() {
    int i =0 ;
    cout<<"Before the loop"<<endl;
    while(i<4)
    {
        cout<<"Hello World"<<endl;
        i++;
    }
    cout<<"After the loop"<<endl;
}
```

while-loop Statement: Pretest

- The while loop is a pretest loop the
- Boolean expression is tested each time before the loop body is executed
  - The loop body may be executed only zero times if the condition is false in the first time

```
int i =5;

while(i<4)
{
    cout<<" "<<i;
    i = i - 1;
}
```

while-loop Statement: Infinite Loop

- If the loop condition never resolves to be false, it continues to execute infinite number of times.
- An infinite loop — usually caused by
  - incorrect logic
  - failure to modify the loop control variable

```
int i = 1;

while(i>0)
{
    cout<<" "<<i;
    i = i + 1;
}
```

while-loop Statement: Infinite Loop

- If the loop condition never resolves to be false, it continues to execute infinite number of times.
- An infinite loop — usually caused by
  - incorrect logic
  - failure to modify the loop control variable

```
int i = 1;

while(i>0)
{
    cout<<" "<<i;
}
```

while-loop Statement: Infinite Loop

- If the loop condition never resolves to be false, it continues to execute infinite number of times.
- An infinite loop — usually caused by
  - semicolon after the loop condition in while loop

```
int i = 0;

while(i<=3);
{
    cout<<" "<<i;
    i = i + 1;
}
```

while-loop Statement: Counters and Accumulators

- The variable count is a counter
  - It keeps track of the number of times the loop body is executed
  - Its value is incremented each time the loop body executes

```
int count = 0;

while(count<5)
{
  count ++ ;
}
// display the counter
cout<<" "<<count;
```

while-loop Statement: Counters and Accumulators

- The variable sum is an accumulator
  - It keeps a running sum of the values that were entered
  - It is added to each time the loop body executes

```
int count = 0, value;
int sum = 0;

while(count < 3)
{   cout<<"\nEnter a value: ";
    cin>>value;

    sum = sum + value;
    count ++ ;
}

// display the sum
cout<<" "<<sum;
```

Homework Practice Programs

- Use a loop to display
  - $1, 4, 7, \cdots, 25$
- Use a loop to display
  - $2, 4, 6, 8, \cdots, 100$
- Use a loop to display
  - $2, 1, 4, 3, 6, 5, 8, \cdots, 100$

Thank you

Please let me know if you have any questions.