BSc (Hons) in Software Engineering

IT1101: Introduction to Computer Programming

Lecture 4

Iterations

Topics to be Covered

- Iterations
  - ***for*** Loop
  - ***while*** Loop
  - ***do-while*** Loop
- ***break***, ***continue*** and ***goto*** Statements

Iterations

- Iterations are also called as loops
- Looping statements allow the program to repeat a section of code any number of times or until some condition occurs.
- The looping constructs supported by the C language are:
  - ***for*** loop
  - ***while*** loop
  - ***do while*** loop

for loop

```
for( initialization ; test ; increment)
{
   /* code */
}
```

- The *initialization* statement is executed exactly once.
- The *test expression* is evaluated each time before the code in the ***for*** loop executes.
- After each iteration of the loop, the *increment* statement is executed.

Example *for* Loop:

```
for (n = 1; n <= 10; ++n )
{
  triangularNumber += n;
  printf("%d  %d\n", n,triangularNumber);
}
```

Exercise

- Assume that you need a program to calculate and display the first 10 triangular numbers.
- If T_n is the n^{th} triangular number;

$$T_n = 1+2+3+\ldots+n$$

- The easiest way to implement this is to use a ***for*** loop. *(refer ForLoop.c)*

for loop with multiple statements

```
int i, j, n = 10;

for(i = 0, j = 0; i <= n; i++, j+=2)
  printf("i = %d , j = %d \n",i,j);
```

- It is also common to use the comma operator in for loops to execute multiple statements

Using a variable as the statement in *for* loops

```
for (t = list_head; t; t = NextItem(t) )
{
     /*body of loop */
}
```

- If the variable has a value of 0 or NULL, the loop exits, otherwise the statements in the body of the loop are executed.

forever loop

```
for( ; ; )
{
  /* block of statements */
}
```

- This is called a forever loop since it will loop forever unless there is a break statement within the
- statements of the ***for*** loop. The empty test condition effectively evaluates as true.

Exercise – Write down the Output

Ex1:

```
for(i = 1; i <= 10; i++)
{
    printf("%d ", i);
}
```

Ex2:

```
for (i = 5; i > 0; i)
{
    printf("%d ",i);
}
```

while Loop

- A ***while*** loop can be used to do the same thing as a ***for*** loop.
- However a ***for*** loop is a more condensed way to perform a set number of repetitions since all of the necessary information is in a one line statement.

Syntax

```
init_expressions;
while ( loop_condition )
{
      /* code */
     loop_expressions;
}
```

Infinite *while* loop

- A situation where the conditions for exiting the loop will never become true is called an infinite loop.

```
int a=1;
while(42)
{
      a = a*2;
}
```

Exercise *while* Loop

Ex1:

- We need a program that converts pounds to kilograms, we must repeat the conversions until user types -1, which signify to terminate the converter. *(refer WhileLoop.c)*

Ex2

- Write down a program that prints out all the exponents of two less than 100.

Controlling a loop via *break* & *continue*

```
int a=1;

while (42){ // loops until the break statement in the loop is executed
      printf("a is %d ",a);
      a = a*2;
      if(a>100)
            break;
      else if(a==64)
            continue; // Immediately restarts at while, skips next step
      printf("a is not 64\n");
}
```

do while Loop

- The For Loop and the While Loop both make a test of the conditions before the loop is executed.
- Therefore, the body of the loop might never be executed at all if the conditions are not satisfied.
- A ***dowhile*** loop is a post-check while loop, which means that it checks the condition after each run.
- As a result, even if the **condition is zero** (false), it will **run at least once**.

do while Loop

Syntax

```
do
{
     /* do stuff */
} while (condition);
```

- The program statements are executed first.
- Next, the loop_condition inside the parentheses is evaluated.
- If the result of evaluating the loop_condition is TRUE, the loop continues and the program statements are once again executed.

Example: *do while* Loop

- Example:
  - We need a program that converts pounds to kilograms. Read the number of pounds from the user convert it to kilo grams and display the result. Then the program should ask the user that he wants to continue conversion, as far as the answer is yes, read the number of pounds from the user convert it to kilo grams and display the result.
  - Here we must ask the user whether repetition is needed only after performing one conversion. *(refer DoWhileLoop.c)*

Exercise

- Build up a for loop using a ***while*** loop.

```
for (expr1;expr2;expr3)
        Statement
```

break, continue and *goto* Statements

Avoiding a infinite loop

- It is very important to note, once the controlling condition of a loop becomes 0 (false), the loop will not terminate until the block of code is finished and it is time to revaluate the conditional.
- If you need to terminate a loop immediately upon reaching a certain condition, consider using ***break***.

break Statement

- The break statement causes immediate exit from a loop while the program is executing, whether it’s a ***for***, ***while***, or ***do while*** loop.
- Subsequent statements in the loop are skipped, and execution of the loop is terminated.
- Execution continues with whatever statement follows the loop.

break Statement

- Example
  - We need a program that converts pounds to kilograms. Read the number of pounds from the user convert it to kilo grams and display the result. But if the number of pounds entered by the user is -1 we need to terminate conversion. *(refer BreakStatement.c)*

continue Statement

- This is also used in loops to skip the rest of the statements in the loop and continue from the next iteration

Example: *continue*

- We need a program that converts pounds to kilograms. Read the number of pounds from the user convert it to kilo grams and display the result. But if the number of pounds entered by the user is a negative value, we need to ask the number of pounds again. If -1 is entered we need to terminate conversion.

(refer ContinueStatement.c)

goto Statements and Labels

- ***goto*** is a very simple and traditional control mechanism
- C provides the ***goto*** statement, which allows to unconditionally branch to a labeled section of the code.
- Code that relies on ***goto*** statements is generally harder to understand and to maintain
  - Thus it is rarely used.
  - One common usage of ***goto*** statements is error handling.

goto Statements and Label Syntax

```
MyLabel:
      /* some code */
goto MyLabel;
```

- It is a statement used to immediately and unconditionally jump to another line of code.
- First declare or add the Label trailing with a “” :”
- When ever you need to route or direct your program to the declared label, use the ***goto*** keyword.

Example: *goto* Statement

- We need a program that converts pounds to kilograms. Read the number of pounds from the user convert it to kilo grams and display the result. But if the number of pounds entered by the user is a negative value, we need to ask the number of pounds again. If is entered -1 we need to terminate conversion. *(refer GotoLabel.c)*

Summary

This lecture was mainly focused on the Looping/Iterating structures

- Iterations
  - ***for*** Loop
  - ***while*** Loop
  - ***do-while*** Loop
- ***break***, ***continue*** and ***goto*** Statements

Thank You.