Flow Control Through Loops

For, While, Do-While

Loops

- Loops in C cause a section of the program to be executed repeatedly while an expression is *true*.
- When the expression becomes *false*, the loop terminates and the control passes on to the statement following the loop.
- A loop consists of two segments, one is the *control statement* and the other is the *body of the loop*.
- There are the following three kinds of loops in C:
    - ***for***
    - ***while***
    - ***do-while***

for Loop

The for loop is useful while executing the statement a number of times.

- The first component, `i=1` is executed only once *initialization*.
- The second component `i<=10` is evaluated once before every execution for the statement within the loop *test expression*.
  - Expression `true` statement within loop executes.
  - Expression `false` statement terminates and control is transferred to the statement following the for loop.
- The third component i++ is executed once after every execution of the statement within the loop *update exression*

```
#include<stdio.h>
void main()
{
    int i;
    for(i=1; i<=10; i++)
        printf("%i", 5*i);
}
```

The general syntax of `for` loop:

```
for(initial expression; test
expression; update expression)
    statement/compound statement.
```

Syntax of the *for* loop

```
i.   for(j=0;j<25;j++)
             statement;   □ single statement body
ii.  for(j=0;j<25;j++)
     {
             statement1;
             …          □ single statement body
             statement n;
     }□ No semicolon here…
```

Syntax of the *for* loop

iii. `for(j=0;j<25;j++);` □ loop with no body
`printf("%i",j);`

iv. `for(i=0,j=0;j<25;i+=5,j++)` □ Multiple initialization
`printf("%i, %i",i,j);` and multiple update
using coma operator.

v. `for(; j<25; j++)` □ Initialization expression not used
`printf("%i",j);`

vi. `for(;; j++)` □ Initialization & test expression not
`printf("%i",j);` used

vii. `for(;;)` □ Initialization, test & update `printf("Infinite`
`loop");` `expression not used`

while Loop

- The *for* loop is more natural in places where the precise number of times of the *loop* is to be executed is known before it is executed.
- The while loop is often used when the number of times the loop is to be executed is not known in advance.

General Syntax:

```
while(test expression)  |while(test expression)
    {
    statement;          | statement;
                  | statement;            }
```

do-while Loop

- The *while* loop is ***top-tested***, i.e. evaluates the condition before any of the statements in its body.
- The *do-while* loop is ***bottom-tested***, i.e. evaluates the condition after the execution of the statements in its construct.
- The statement within the do-while loop is executed at least once.

General Syntax:

```
do                    |do{
   statement;             |  statement;
   while(test expression);    |  statement;
                    } while(test expression);
```

break STATEMENT

- A *break* statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop.
- break statement is a very useful tool if the user does not know the number of times the loop will run and helps in terminating the infinite occurrence of a loop.

switch Statement

- A *switch* statement allows the user to choose a statement (or a group of statements among several alternatives.
- The switch statement is useful when a variable is to be compared with different constants, and if it is equal to a constant, a set of statements are to be executed.
- The constants in case statements can be of char or int data type only.

UNIVERSITY OF ENGINEERING & MANAGEMENT
Good Education, Good Jobs

THANK YOU