FIT ICT Software Development

Lecture 7

Control Structures

Lecturer : Charles Tyner

FIT ICT Software Development

Testing Expressions

The *if* keyword performs basic conditional test

Evaluates a given expression and returns Boolean value

Statements within braces following the evaluation will be executed when expression returns True

Syntax:

if (test-expression) {statements-to-execute}

May be multiple statements

Each statement must be terminated with semi-colon

FIT ICT Software Development

Testing Expressions

Sometimes need to evaluate multiple expressions

Can be achieved in two ways:

1. Logical *&&* AND operator ensures statements only executed when both expressions found to be true

Syntax

If((test-expression_1)&&(test-expression_2))
{statements-to-execute

FIT ICT Software Development

Testing Expressions

2. Alternatively, multiple *if* statements can be nested

Syntax:

```
if (test-expression_1)
{
    if (test-expression_2)
        {statements-to-execute}
}
```

If either test returns False, statements are not executed

FIT ICT Software Development

Testing Expressions

Conditional branching – where there are alternative code chunks to execute depending on whether expression is True or False – is provided by appending an *else* statement

```
if (test-expression)
    {statements-to-execute-when-true}
else
    {statements-to-execute-when-false}
```

FIT ICT Software Development

Testing Expressions

1. Begin a new program with a preprocessor instruction to include standard input/output library:

```
#include <stdio.h>
```

2. Add main function that outputs a statement when a tested expression is found to be true:

```
int main() {
    if( 5 > 1) {printf("Yes, 5 is greater than 1\n") ;}
}
```

FIT ICT Software Development

Testing Expressions

3. In main block, output a statement when two tested expressions are found to be true:

```
if( 5 > 1)
{
   if(7 > 2)
      {printf("Yes, 5 is greater than 1 and 7 is greater than 2\n") ;}
}
```

FIT ICT Software Development

Testing Expressions

4. Now output a default statement after two tested expressions are both found to be false:

```
if( 1 > 2)
    {printf("First expression is true\n") ;}
else if(7 > 2)
    {printf("Second expression is true\n") ;}
else
    {printf("Both expressions are false\n") ;}
```

5. Complete exercise in usual manner

FIT ICT Software Development

Branching Switches

Multiple *if…else* statements can be performed more efficiently by *switch* statement when test expression evaluates a single condition

Takes a given value as parameter argument

Tries to match value from a number of case statements

Each *case* terminated with *break* keyword so that further matches are not sought **unless that is a deliberate requirement**.

FIT ICT Software Development

Branching Switches

Optionally, a single *default* statement can be provided after the set of cases to cover the situation where no match is found

Syntax:

```
switch (test-value)
{
  case match-value: statements-to-execute; break;
  case match-value: statements-to-execute; break;
  default: statements-to-execute; break;
}
```

No two case statements can attempt to match same value

FIT ICT Software Development

Branching Switches

Where a number of match-values are each to execute the same code, only the final ***case*** need include the statements to be executed and the ***break*** to exit the block

```
Switch (num)
{ case 0:
  case 1:
  case 2: printf("Less than 3\n"): break;
  case 3: printf("Exactly 3\n"): break;
  default: printf("Greater than 3 or less than 0\n");
}
```

FIT ICT Software Development

Branching Switches

1. Begin a new program with a preprocessor instruction to include standard input/output library:

```
#include <stdio.h>
```

2. Add main function that declares and initialises one integer variable and one of type character:

```
int main() {
    int num = 2; char letter = 'b';
}
```

FIT ICT Software Development

Branching Switches

3. In main function block, insert a switch statement that attempts to match the integer value

```
switch(num)
{
case 1: printf("Number is one\n"); break;
case 2: printf("Number is two\n"); break;
case 3: printf("Number is three\n"); break;
default: printf("Number is unrecognised\n");
}
```

FIT ICT Software Development

Branching Switches

4. Insert another switch statement that attempts to match the character value. Note the more terse syntax

```
switch(letter)
{
case a: case b: case c:
printf("Letter is %c\n", letter); break;
default: printf("Letter is unrecognised\n");
}
```

5. Complete exercise in usual manner

FIT ICT Software Development

Loops

A loop is a block of code that repeats automatically based on a condition

One complete execution of the block within a loop is an '*iteration*' or a '*pass*'

In C, three types of loop

- *for*
- *while*
- *do while*

FIT ICT Software Development

For Loop

Most common loop

Syntax:

For(initialiser; test expression; incrementer***) {statements}***

Initialiser used to set starting value for a counter, traditionally named ***i***

Controls number of iterations made by loop

While test expression returns True, count incremented and statements executed

When False returned, loop ends and control passes to next code outside loop

Loops can be nested

FIT ICT Software Development

For Loop

1. Begin a new program with a preprocessor instruction to include standard input/output library:

```
#include <stdio.h>
```

2. Add main function that declares two integer variables for use as iteration counters:

```
int main()
{
    int i, j;
}
```

FIT ICT Software Development

For Loop

3. Insert a *for* loop to output the loop counter value on each iteration

```
for(i=1; i< 4; i++)
{
printf("Outer loop iteration %d\n", i);
}
```

4. Complete exercise in usual manner

FIT ICT Software Development

For Loop

5. In loop block, insert a nested loop immediately after existing output statement, to output the loop counter value on each inner loop iteration

```
for(j=1; j< 4; j++)
{
printf("\tInner loop iteration %d\n", j);
}
```

6. Save, compile and run again to see output from outer and inner loops

FIT ICT Software Development

Looping While True

While loops also require initialiser, test-expression and incrementer, but different syntax

Initialiser must appear before start of loop block

Test-expression must appear in parentheses after *while* keyword

Statements to be executed and incrementer enclosed in braces follow

initialiser

while(test-expression)

{statements-to-be-executed; incrementer}

FIT ICT Software Development

Looping While True

Do while loops subtly different in syntax and effect

Keyword ***do*** placed before loop block (outside braces) and while statement placed after block

initialiser

do{statements-to-be-executed; incrementer}

while(test-expression)

This means that statements to be executed will always be executed at least once because test-expression not evaluated till afterwards

FIT ICT Software Development

Looping While True

1. Begin a new program with a preprocessor instruction to include standard input/output library:

```
#include <stdio.h>
```

2. Add main function that declares a counter variable, then declares and initialises an array of three elements:

```
int main()
{
    int i, arr[3] = {10, 20, 30};
}
```

FIT ICT Software Development

Looping While True

3. In main function block, insert a *while* loop that outputs each element number and the value it contains:

```
i = 0;
While (i < 3)
{
    printf("While: arr[%d] =%d\n", i, arr[i]); i++;
}
```

FIT ICT Software Development

Looping While True

4. Now insert a ***do while*** loop that also outputs each element number and the value it contains:

```
i = 0;
do
{
    printf("\nDo while: arr[%d] =%d", i, arr[i]); i++;
}
while (i < 3)
```

FIT ICT Software Development

Looping While True

5. Save, compile and run. Study output and make sure you understand why the output is as it is

Note how counter is initialised twice

Note alternative positioning of ***\n*** escape sequence

6. Edit program and initialise counter to 3 in each initialisation. Save, compile, run and study output. What is different? Can you explain the reason?

FIT ICT Software Development

Breaking Out Of Loops

Loops will repeat until test-expression returns False

Essential that loop body contains code that will, at some point, change result and exit loop

Otherwise, infinite loop created that will lock the system

Additionally, *break* keyword can be used to terminate a loop when specified condition is met

Placed inside loop block and preceded by text expression

FIT ICT Software Development

Breaking Out Of Loops

1. Begin a new program with a preprocessor instruction to include standard input/output library:

```
#include <stdio.h>
```

2. Add main function that declares two integer variables for use as iteration counters:

```
int main()
{
    int i, j;
}
```

FIT ICT Software Development

Breaking Out Of Loops

3. Insert a two nested loops to output their counter value on each of three iterations

```
for(i=1; i< 4; i++)
{
        for(j=1; j< 4; j++)
        {
          printf("Running i =%d j =%d\n", i, j);
        }
}
```

4. Complete exercise in usual manner. Execute program to see output from nested loops

FIT ICT Software Development

Breaking Out Of Loops

5. Edit program by inserting following code at start of inner loop

```
if(i==2 && j == 1)
{
    printf("Breaks inner loop when i =%d and j =%d\n", i, j);
    break;
}
```

6. Complete exercise in usual manner. Execute program and study output

FIT ICT Software Development

Breaking Out Of Loops

The ***continue*** keyword can be used to skip a single iteration of a loop when specified condition is met

Placed inside loop block and preceded by text expression

When test returns True, that iteration ends

FIT ICT Software Development

Breaking Out Of Loops

7. Edit program by inserting following code at start of inner loop

```
if(i==1 && j == 1)
{
    printf("Continues inner loop when i =%d and j =%d\n", i, j);
    continue;
}
```

8. Save, compile, run and study.