# C Programming for MSc

# Lecture 3:
# Iteration & The ASCII Code

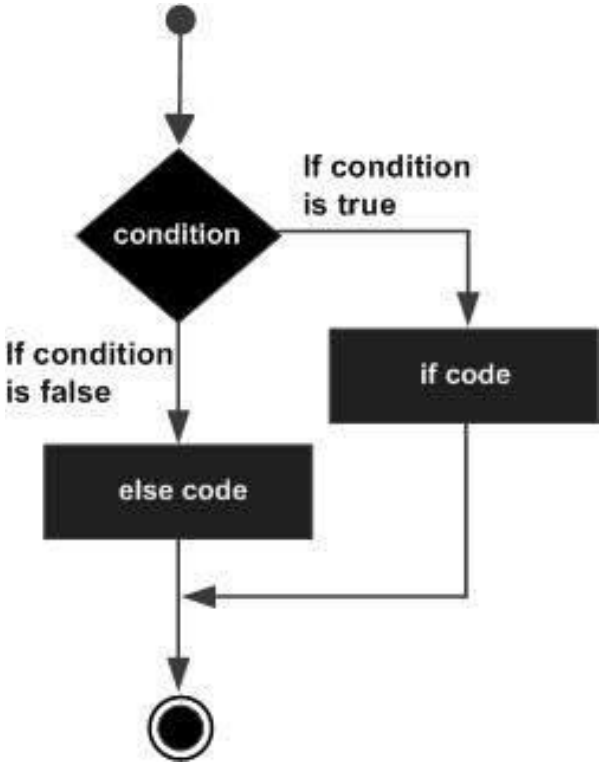
*Prof. Stephen Smith*
Room P/B/005
Ext 2351
E-mail: stephen.smith@york.ac.uk

# Outline

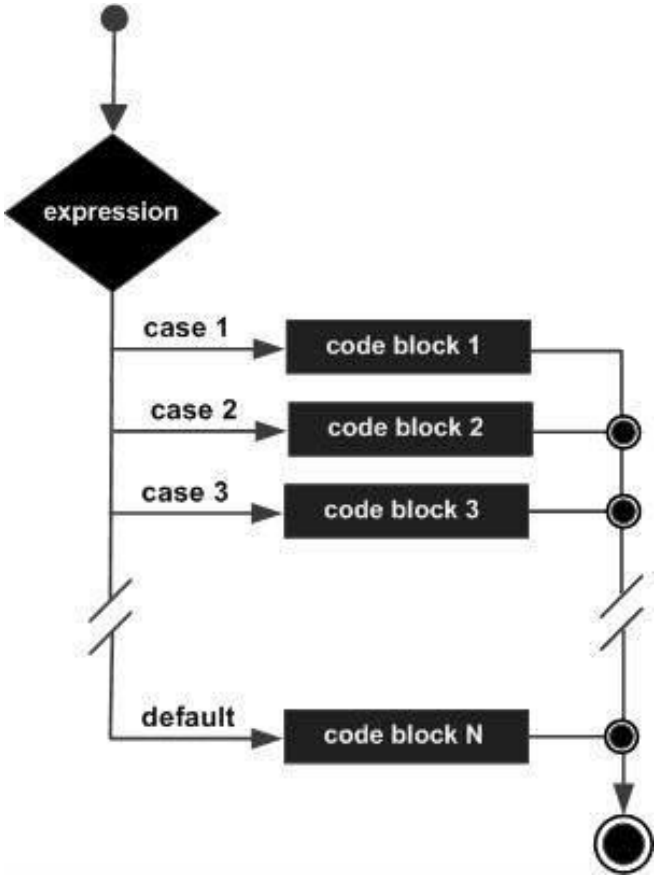
- Recap
- Iteration
  - **While**
  - **Do-While**
  - **For**
- The ASCII Code
- C Assignment shortcuts
- Lab 3

# Recap: conditional flow

`if` & `if-else`



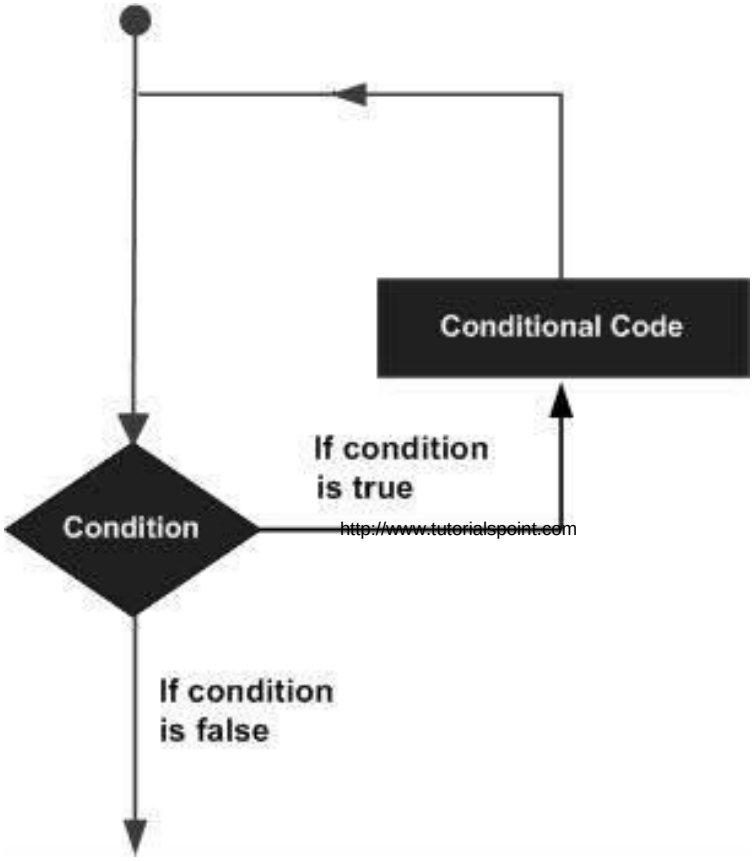

`Switch - case`




http://www.tutorialspoint.com

# Iteration: The **while** loop

```
while condition
{
    statement 1;
    …
}
```

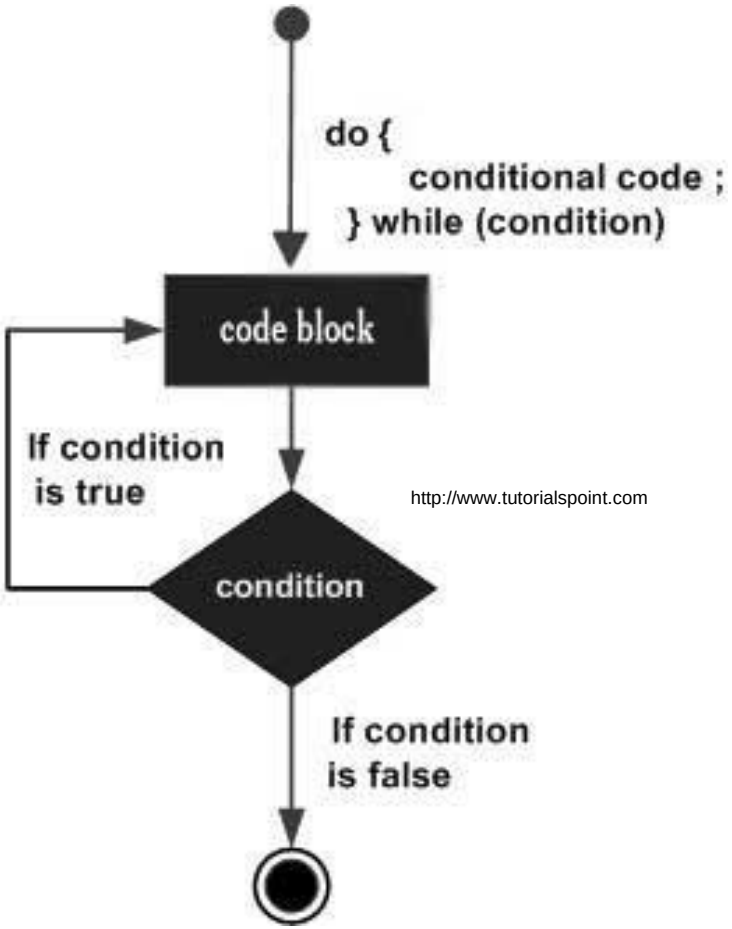



- Code block will be repeated as long as condition is TRUE
- When *condition* is FALSE execution will jump to the next statement after the block.
- Executes block from **0 to infinity** number of times.
- Hence, to avoid an infinite loop you must ensure that *condition* becomes FALSE during an iteration of the block.

http://www.tutorialspoint.com

# Iteration: The **repeat-until** loop (C: `do-while`)

```
repeat
{
    statement 1;
    …
}
until condition
```




- The statement or block of statements will be repeated until *condition* is TRUE (C: non-zero).
- As soon as *condition* is FALSE (C: zero) execution will jump to the next statement after the block.
- Executes block from 1 to ‘infinity’ number of times. Can call this the “at least once loop.”
- Hence, to avoid an infinite loop you must ensure that *condition* becomes FALSE during an iteration of the block.

# C Example: `while` loop iteration

```
/* while example that repeatedly reads integers
    greater than or equal to 10 and when not reports  */
int main (void);
{
   int number;
   number = 10;
   while (number >= 10)
   {
      printf(“\nPlease enter an integer greater than ten: ”);
      scanf(“%d”, &number);
    }
   printf(“\nNumber entered was less than ten.”);
   return 0;
}
```

# C Example: `do-while` loop iteration

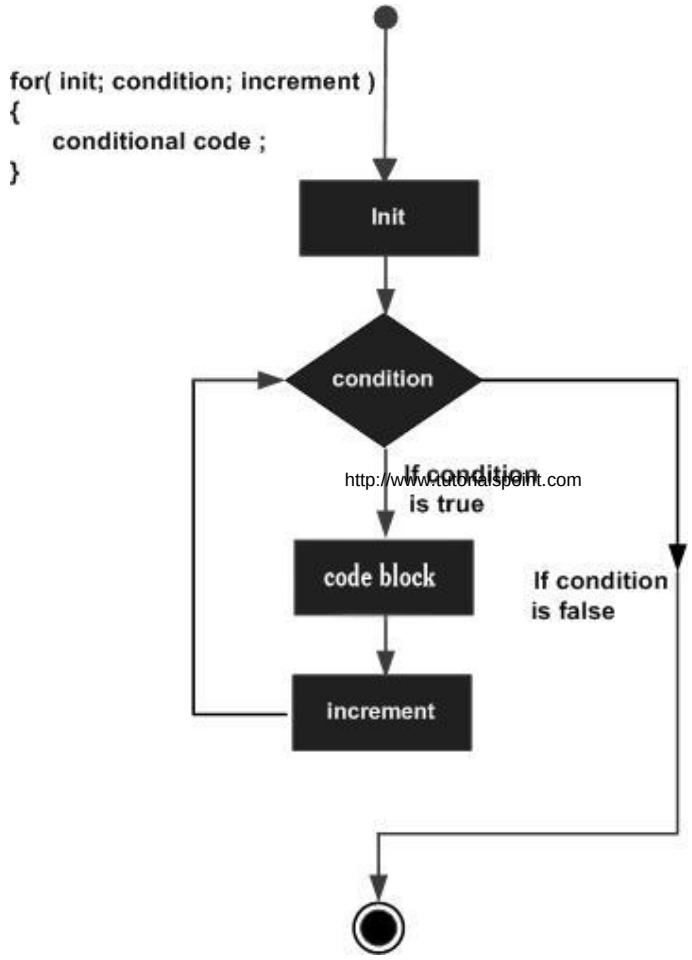
```
/* do-while example that repeatedly reads integers
    greater than or equal to 10 and when not reports */
int main (void);
{
   int number;
   number = 10;
   do
   {
     printf("\nPlease enter an integer greater than ten: ");
      scanf("%d", &number);
    }
   while (number >= 10);
   printf("\nNumber entered was less than ten.");
   return 0;
}
```

# Which is the more “elegant” iterative form to use in these examples: `while` or `do-while`?

- Both perform identical functions.
- Both well documented,  have same memory allocation & efficiency.
- Ask yourself:
    - “if I had to read this code, which form is easier to follow?”
    - “to make sense, does the iterated block need to execute at least once?”
- So, which form is the more elegant - `while` or `do-while`? (hint: one of them is “forcing” the initial execution to gain first user entry, the other asks for entry before considering it)
    - Answer can depend on specification you were given  - does the specification call for user input at start, are there agreed programming standard (e.g. PWS)?
- These are the kinds of choices you will have as you design your code, in this case for iteration.

# Iteration: The **for-do** loop (C: `for`)

**for** (*iterator := start_val to end_val)* **do**
{
    statement 1;
    …
}




- The **for-do** loop (aka **do** loop) in its simplest form iterates a statement or block of statements a fixed number of times.
- The number of incremental iterations is (end_val–start_val)+1, where end_val, start_val are integers, and end_val >= start_val.
- Other forms include decremental iteration, conditional iteration etc.

# C Example: `for` loop iteration

for (initialization; condition; increment)

{ statement(s)…}

- The `for` loop repeats a block of statements a fixed number of times.
- *condition* part is required.
- *Initialization* and *increment* parts are optional in C.

Examples:

```
int i, initial = 1, final = 100;
int sum = 0;
for (i = initial; i <= final; i = i + 2)
    sum = sum + i;
```

```
int i, initial = 50, final = 1;
double sum = 0;
for (i = initial; i >= final ; i = i - 1)
    sum = sum + 1.0/i ;
```

# The ASCII Standard

- Stands for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (ASCII)
- Developed by the ASA in 1960 as an agreed (standard) code for computer text characters applicable across all computing hardware & software
- Required because otherwise any keyboard, display, printer, i/o device, comms channel, etc, may not communicate correctly, share code, interface protocols, support devices etc.
- It was the most common character encoding standard on the www until 2007; and newer codes incorporate (supersets of) ASCII.
- ASCII is a 7-bit code – added symbols and characters required an 8-bit code (UTF-8 is most popular).

# ASCII Codes

| Backspace | Return/Enter | Space |
|---|---|---|
| 8 | 13 | 32 |

There are 128 allocated codes (7 bits)

| ! | “ | # | ? | % | & | ’ | < | > | * | + | , | - | . | / |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |

| 0 | 1 | 2 | … | 9 |
|---|---|---|---|---|
| 48 | 49 | 50 | … | 57 |

| : | ; | < | = | > | ? | @ |
|---|---|---|---|---|---|---|
| 58 | 59 | 60 | 61 | 62 | 63 | 64 |

| A | B | C | … | Z |
|---|---|---|---|---|
| 65 | 66 | 67 | … | 90 |

| [ | \ | ] | ^ | _ | ‘ |
|---|---|---|---|---|---|
| 91 | 92 | 93 | 94 | 95 | 96 |

| a | b | c | … | z |
|---|---|---|---|---|
| 97 | 98 | 99 | … | 122 |

# Using ASCII in C

- Char variables are really numbers! (0 to 255)
- Example:

```
char x = 'a';
```

- We can say:

```
x = 'a' + 1;
if (x > 'b')
```

- `getch()` is a useful function not in standard C:
  - Defined in `conio.h` in MS Windows
  - Reads a character from the console (keyboard) without a screen echo.
  - Will pause execution until a character is pressed (but not numlock, shift etc).

# C Shortcuts: counting expressions (useful for iterating)

- Shorthand notations are a handy way of tightening your code, but…
- …should be avoided in spec documentation, for example in pseudocode, because they are language-specific
- Incrementing: `i = i + 1;`

  can also be written: `i++;`
- Decrementing: `j = j – 1;`

  can also be written `j--;`

# C Shortcuts: assignment operators

- Certain special operators allow shorthand for more complex assignments.

variable = variable op expression;

can also be written:

variable op= expression;

where op can be +, -, *, / (and a few others).

Examples

```
int k = 0;
k += 2:
```

```
double x = 5.0, y = 3.2;
x /= v+ 2.5;
```

# Lab 3: Iteration

- Repeating groups of instructions by iteration
- Using the three ways to iterate in C
  - while() { }
  - do { } while ();
  - for (i = start ; i < end; i++) { }