C Programming for MSc

Lecture 2:
Conditional control
& structured programming

Prof. Stephen Smith
E-mail: stephen.smith@york.ac.uk

Outline

- Statements
  - Variables & assignment
  - Expressions
  - Operations & precedence
  - Conditional statements & flow control
- Block structured programming
- Programming practice – formatting
- Lab 2

Statements

- A *statement* in a high-level programming language is:
  - an instruction to the computer to perform a specified action
  - the smallest programming element expressing such an action
- A *statement* in C is an expression
  - followed by a semicolon
- For example:

```
3.142;

a+b;

if (x == 6) y = 3;

Velocity = 4.2;

i = i + 1;

printf("Hello");
```

Variable Names

- A variable should be a meaningful name
  - that points to a value of a certain type
- Variables are used to form *expressions* in *statements*
- When using a variable we actually refer to the *address* of the memory location where the data is stored
- Variable names in C can contain:
  - letters, digits, underscore.
- They cannot begin with a digit
  - nor should they be pre-defined words (**if**, **main**…).
- All names are *case sensitive*.
  - e.g. a variable called **`count`** is not the same as one called **`NCount`**

Assignment of Variables

- In a program, variables are:
  - assigned scalar values
  - other variables of the same (or compatible) type
  - the results of an expression or of a called function
- An assignment is a directional application of a value or the result of an expression to a variable (or rather to the memory location it points to)
- In C, values are assigned to variables using the *assignment* symbol ‘=‘
- e.g.

```
distance_to_Tokyo = 9720;
sum = 0;
sum = sum + 10;
letter = ‘z’;
```

Comments, function **`main()`** & types

- Comments

```
// Code written by Joe Bloggs © 2018
/* Declare some variables for the calculations */
```

- C programs must have a main *function*

```
int main(void)
{
    return 0;
}
```

- *Variables* are symbols used to hold data. They must have a *type* (int, double, char…)

```
int distance_to_tokyo, distance_to_airport;
int speed_of_plane, speed_of_car;
int time_to_fly, time_to_drive, time_to_tokyo;
int average_speed;
```

Relational operator expressions

- Relational operators compare values or expressions to produce a result, typically used for decision flow control
- In C the relational operators are *binary operators* that take two expressions as operands and produce either the `int` values 0 (*false*) or 1 (*true*)

`<` `>` `<=` `>=`

- If a and b are *arbitrary arithmetic expressions* then the above operators give the following

Expression → **`a - b`** is ↓	**`a < b`**	**`a > b`**	**`a <= b`**	**`a >= b`**
positive	0	1	0	1
zero	0	0	1	1
negative	1	0	1	0

Equality operator expressions

- In C the equality/inequality operators are the binary operators that take two expressions as operands and produce either the **`int`** values 0 or 1

  `==` **`!=`**

- If a and b are *arbitrary arithmetic expressions* then the above operators give the following

Expression → **`a - b`** is ↓	**`a == b`**	**`a != b`**
zero	1	0
nonzero	0	1

Logical operator expressions

- In C there are 3 logical operators that when applied to expressions all produce `int` values 0 or 1

  `!` **`&&`** **`||`**

- If **`a`** and **`b`** are *arbitrary arithmetic expressions* then the above operators give the following:

`a`	**`!a`**
zero	1
nonzero	0

`a`	**`b`**	**`a && b`**	**`a || b`**
zero	zero	0	0
zero	nonzero	0	1
nonzero	zero	0	1
nonzero	nonzero	1	1

Arithmetic operations and precedence

- Some mathematical operations:
- Addition `+` Subtraction `-`
- Multiplication `*` Division `/`
- Mathematical operations have an operator precedence, e.g.

```
6 + 4/2
```

- Parentheses can be used to define your own order, e.g.

```
(6 + 4)/2
```

Conditional flow – statement types

- Conditional statements
  - used to make decisions
  - that will control the flow of statement execution in the program
- Fundamental elements of programming languages:
  - the **`if-then`** statement
  - the **`if-then-else`** statement
  - the **`case`** statement

Simple conditional statements

- In C the conditional statements are called:

  `if` **`if-else`** and **`switch-case`**

- They can be written in simple for compound form
- **`if`** statements in simple form look like this:

```
if (expression_is_true) do_this_statement;
```

- **`if-else`** statements in simple form look like this:

```
if (expression_is_true) do_this_statement;
else do_this_statement;
```

What can be in the round brackets in an `if` statement?

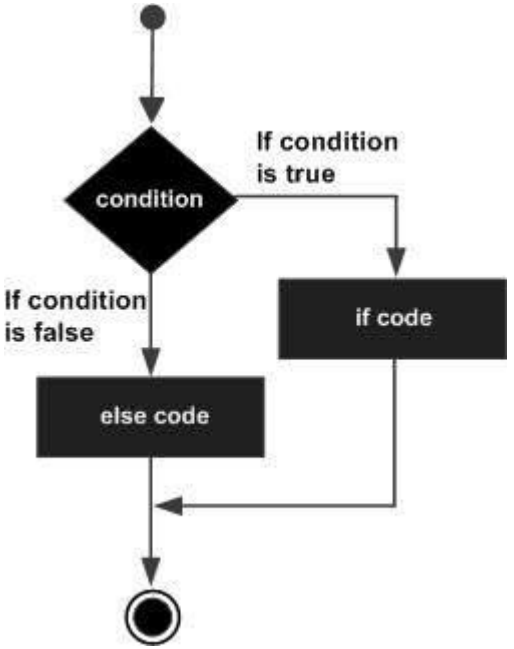
- An *expression*
  - a meaningful combination of constants, variables, operators or function calls
- Examples

```
1  +  2

x  +  y  +  z

3.142 * radius * radius

count = 1

x  >  y

(a > b)  &&  (x > y)

printf("Hello")
```

More complex `if` and `if-else` statements

- `if` statements can be nested
- produces more complex conditional expressions
  - care required over their interpretation

```
if (a == 5)
    if (b == 7) c = 2;
    if (x > 5.3)
    if (y < 4.7)
    printf("Target in range");
    else
    printf("Target not in vertical range");

if (age <= 5) tablets = 1;
else if (age <= 10) tablets = 2;
else tablets = 3;
```



http://www.tutorialspoint.com

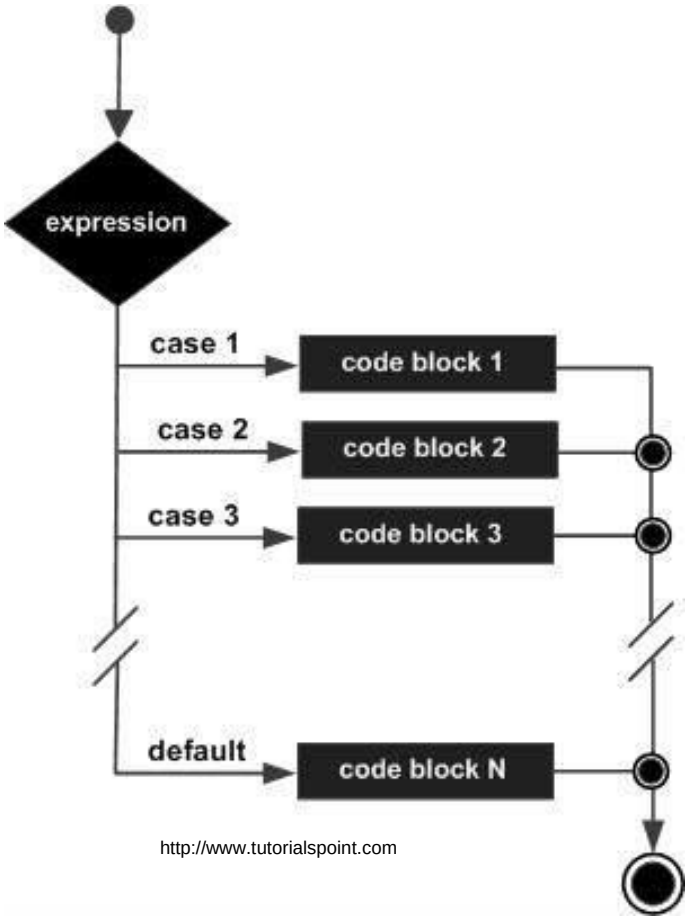
Compound `if` and `if-else` statements

- A number of statements can be executed within conditions by using the curly brackets { }
- For example:

```
if (age <= 5)
{
    tablets = 1;
    drops = 5;
}
else
if (age <= 10)
{
    tablets = 2;
    drops = 10;
}
else
{
    tablets = 3;
    drops = 15;
}
```

switch case statement

```
switch (number_entered)
{
    case 1:
        printf("one\n");
        break;

    case 2:
        printf("two\n");
        break;

    case 3:
        printf("three\n");
        break;

    default:
        printf("number is not between one and three\n");
        break;
}
```



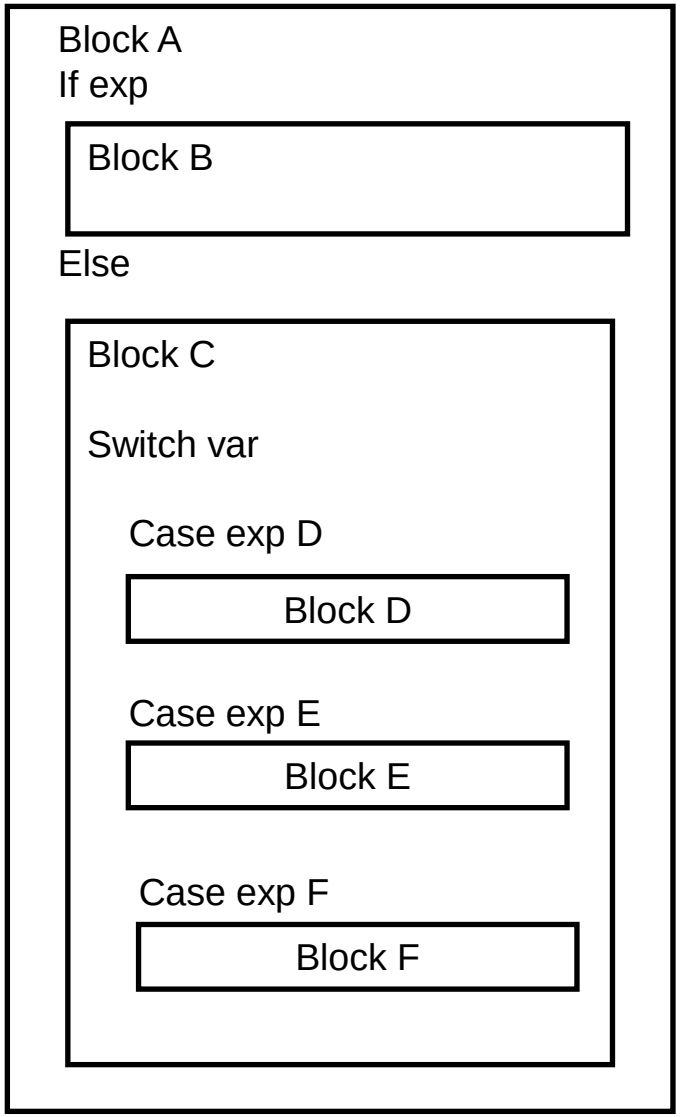
C Program Structure

```
/* Top level header comments                                                    */

Preprocessor directives    /*  Special instructions (e.g. #Include for header files) on how to prepare  */
                           /*  program for compilation.                                                 */
Global declarations        /*  Global instructions on memory allocation.                                */

/* main program header comments                                                 */
main ()
{
   Local declarations      /*  Describes the data that will be used in main.                            */
   Statements              /*  Code to perform a specific task, function calls.                         */
}
Function 1 ()              /* Declares a function and its arguments.                                    */
{
   Local declarations      /* Describes the data used and only visible to Function 1                    */
   Statements              /* Code that uses the data to perform the task of Function 1                 */
                  . . .
}
Function n ();
{
   Local declarations ;
   Statements ;
}
```

Block Structured Programming

- A block is a section of code that is grouped together, consisting of declarations and statements.
- The structured programming approach uses control structures, functional elements that control program flow between blocks.
- C is a block-structured programming language as it readily permits the creation of blocks, and nested blocks within blocks.
- In C, we use the { and } symbols, called ‘braces’ or ‘curly brackets’

Nested Conditional Blocks



Two C programs
(semantically identical)
which do you prefer to work with?

```c
/* program demonstrating good format */

#include <stdio.h>

int main(void)
{
    int x, y;

    x = 5;
    y = 7;

    printf("The value of x+y is %d\n",x+y);

    return 0;

}
```

```c
/* pro
gram d
emo
nstrating bad format */

#include <stdio.h>

int main        (void)

                    {int x,
y;

            x = 5;
    y =
        7;

    printf
        ("The value of x+y is %d\n",
x+y);

    return

        0;

}
```

Programming practice - formatting

- C programs are ‘free format’
  - the compiler looks for semicolons and curly brackets as statement delimiters
- Programs should be formatted and designed so that they are:
  - readable
  - Indicate structured program flow
  - are efficient
  - ‘elegant’ – intuition acquired with experience
- A trade-off
  - except for specialised applications
  - readability trumps efficiency as an objective!

Header Files

- C has lots of inbuilt functions that can be used in programs. You do this by using the **`include`** statement e.g.

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

- The *header file* **`stdio.h`**
  - contains definitions of many functions
  - related to input and output from the keyboard
  - can be used to improve the appearance of output
  - e.g. the screen printing function **`printf`**

Lab 2: Conditional statements

- Getting input from the user
  - the **`scanf`** function
  - More on the **`getch`** function
- Making decisions in programs
  - **`if`** statements and **`switch`** statements
- Decision expressions
  - Relational operators `<` `>` `==` `!=` `<=` `>=`
  - Logical operators `&&` `||` `!`
- Compound expressions
  - curly brackets `{` `}`