COMP1511 NOTES

T1 2019

C basics

Variables

- 4 bytes are used to store an int variable (32 bits so 2^{32} possible values)
- Note: illegal to store a value outside the range that can be represented
- 8 bytes are used to store a double variable (64 bits so 2^{64} possible values)
- *Declare:* first time a variable is mentioned, we need to specify its type
- *Initialise:* before using a variable, we need to assign it a value
- Variable names can be made up of letters, digits and underscores
    - Use a lower case to start your variable names
    - Note: variable names are CASE SENSITIVE
    - Avoid keywords such as if, while, return, int and double
- Using values in printf() and scanf():
    - %d – integer value
    - %lf (or %g for printf only) – double value
    - %x or %X – hexadecimal value (lowercase or uppercase)
- scanf() is used to read in 1 value at a time

#define

- Give constants a name to make your program more readable
- #define statements go at the top of your program after #include statements
- #define names should be in all capitals with underscores (if necessary)

Mathematics in C

- Usual maths operations + - * /
    - Use brackets when in doubt of order of operations
    - BEWARE: division may not be what you expect
        - If either number is a double, the result will be a double
        - Dividing 2 integers is an integer
        - The fractional part is discarded (NOT ROUNDED)
- % modulo gives the remainder after division
- Other mathematical functions are included in math.h:
    - sqrt(), sin(), cos(), log(), exp()

Linux commands

cp – copies files and directories

cp sourceFile destination – copies file

cp -r sourceDir destination – copies directory

mv – moves or renames a file

mv source destination – moves a file

- If destination is an existing file, file is overwritten
- If destination is an existing directory, file is copied into directory

rm – removes a file

rm filename

rm -r directoryName

- Be careful and have backups – no undo or recycling bin

Conditional execution

- There is no Boolean type in C: 0 is FALSE and anything non-zero is TRUE

if and else statement

- if statements allow us to execute code 0 or 1 times

```
if (expression) {
        statement1;
} else {
        Statement2;
}
```

- statement1 is executed if expression is non-zero
- statement2 is executed if expression is zero
- Multiple if statements can be chained together with else if (expression)

Relational operators

- `> < >= <= != ==`
- Be careful with comparing doubles for equality as they are approximations
- Relational operators return: 0 for FALSE and 1 for TRUE

Logical operators

- && (and), || (or), ! (not)
- Always evaluate left-hand side and only evaluate right-hand side if needed

while statement

- While statements execute their body until controlling expression is false
- A loop counter may be used to count loop repetitions and execute n times

```
int loop_counter = 0
while (loop_counter < 5) {
        printf("%d", loop_counter);
        loop_counter++;
}
```

- Often a sentinel variable is used to stop a while loop when a condition occurs in the body of the loop

```
int stop_loop = 0
while (stop_loop != 1) {
        if (expression) {
                stop_loop = 1;
        }
}
```

- If nesting while loops, a separate loop counter is needed for each loop

Array

- An array is a collection of variables called array elements
    - All elements must be the same type and don’t have a name
    - Array elements are accessed by the array index
        - Valid indices for an array with n elements are 0 … n-1
        - Array elements must be initialised
    - Can only scanf/ printf array elements, not whole arrays

```
// reading arrays

#define ARRAY_SIZE 42

int i = 0;

int array[ARRAY_SIZE] = {0};

while (i < ARRAY_SIZE) {

       scanf(“%d”, &array[i]);

       i++;

}

// printing arrays

#define ARRAY_SIZE 42

int i = 0;

int array[ARRAY_SIZE] = {0};

while (i < ARRAY_SIZE) {

       printf(“%d\n”, array[i]);

       i++;

}

// copying arrays – array assignment is not allowed

int array[5] = {1, 2, 3, 4, 5};

int array2[5];

int i = 0;

while (i < 5) {

       array2[i] = array[i]);

       i++;

}
```

Arrays of arrays (matrix)

```
int matrix[3][3] = { {1, 2, 3}

                     {4, 5, 6}

                     {7, 8, 9} };

printf(“%d\n”, matrix[1][1]);
```

```
// read a 2D array
#define SIZE 42
int matrix[SIZE][SIZE];
int i = 0;
while (i < SIZE) {
        int j = 0;
        while (j < SIZE) {
                scanf(“%d”, &matrix[i][j]);
                j++;
        }
        i++;
}
// print a 2D array z
#define SIZE 42
int matrix[SIZE][SIZE];
int i, j;
i = 0;
while (i < SIZE) {
        j = 0;
        while (j < SIZE) {
                printf(“%d”, matrix[i][j]);
                j++;
        }
        printf(“\n”);
        i++;
}
```

Function

- Functions allow you to:
    - Separate out and reuse code that serves a single purpose
    - Test and verify a piece of code
    - Shorten code for easier modification and debugging
- Function prototypes allow a function to be called before it is defined

```
int function(int x);
```

    - Specifies: return type, name, number and type of parameters
    - Allows top-down order of functions for readability
    - Allows function definition in separate file