EESC 388

C Programming Fundamentals

Basic C Program Structure:

```c
#include <stdio.h>

int main() {
printf("Hello World!");
return 0;
}
```

Explanation:

```c
#include <stdio.h> // Header and library files provide a set of
predefined functions to do different common tasks. Except standard
headers and libraries like stdio (standard input output) we can use
different pre-defined libraries like math.c for making our program
shorter. User can also add/create user defined header/library files//

int main() { // This is the main function which will be executed when
the program runs. All of your future code (until we learn about
functions) must be written inside this code block {..}//
printf("Hello World!"); // Will explain it later
return 0; // We will not return anything from this function
}
```

Topic 1: Variables, Data Types, and Constants

Variables are containers for storing data values, like numbers and characters. The data type specifies the size and type of information the variable will store. Basic data types include: char, int, long, float, double, void and unsigned version of each of them. To learn more about size and other types click *here*.

Syntax:

- **`type variableName = value;`**

Examples:

- Create a variable called myNum of type int and assign the value 15 to it:

```c
int myNum = 15;
```

- You can also declare a variable without assigning the value, and assign the value later:

```
// Declare a variable
int myNum;

// Assign a value to the variable
myNum = 15;
```

- Printing any string/character/variable:

```
int myNum = 15;
char myLetter = 'D';

printf("My number is %d and my letter is %c",myNum, myLetter);
```

- Here, %d and %c are called format specifiers and are used to print/scan anything to/from console. A list of format specifier used in C program is given *here*.
- Also, **\n** is used to create new line and **\t** is used to create tab in the string during printing.
- Declare the variable as constant when you have values that are immutable (unable to be changed after created):

```
const int minutesPerHour = 60;
const float PI = 3.14;
```

Topic 2: Operators

Operators are used to perform operations on variables and values.

Operators	Name	Description	Example
+	Addition	Adds two values together	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decrease the value of a variable by 1	--x
=	Equals	Assign value to variables	A = x
==	Comparison	Compare between two variables whether they are equal	A == B
!=	Not Equal	Compare between two variables whether they are not equal	A != B
>, >=	Greater than, Greater than/equal to	Compare between two variables whether one is greater than/ greater than or equal to the other	A > B A >= B
<, <=	Less than, Less than/equal to	Compare between two variables whether one is smaller than/smaller than or equal to the other	A < B A <= B
&&	Logical and	Returns true if both statements are true	A && B
||	logical and	Returns true if one of the statements are true	A || B
!	Logical not	Returns the result, returns false if the result is true	!A
&	Bitwise and	Copies a bit to the result if it exists in both operands.	(A & B) = 100, (A: 110, B: 101)
|	Bitwise or	Copies a bit if it exists in either operand.	(A | B) = 111, (A: 110, B: 101)
~	Bitwise not	Binary not Operator that complements each bit of the operand	~A= 001, (A: 110)
<<	Bitwise Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 100 (A: 011)
>>	Bitwise Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 001 (A: 110)

&	Address/Reference Operator	Returns the memory address of a variable.	&a (if a is a variable)
sizeof()	Size Operator	Returns the size of a variable	sizeof(a)
*	Dereference Operator	Dereferences the pointer. Used to retrieve data in memory at the address the pointer holds.	*a

Topic 3: Conditions

Use the if statement to specify a block of code to be executed if a condition is true.

Syntax:

```
if (condition)
   {
   // block of code to be executed if the condition is true
   }
```

Example: Compare the values of two variables and print the greater one.

```
int x = 20;
int y = 18;

if (x > y)
{
printf("x is greater than y");
}
```

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:

```
if (condition) {
  // block of code to be executed if the condition is true }
else {
  // block of code to be executed if the condition is false }
```

Example: Prints Good day if the time is less than 18 else print Good evening:

```
int time = 20;

if (time < 18) {
printf("Good day.");
} else {
  printf("Good evening.");
}
// Outputs "Good evening."
```

Use the else if statement to specify a new condition if the first condition is false.

Syntax:

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and
condition2 is true
} else {
  // block of code to be executed if the condition1 is false and
condition2 is false
}
```

Example: Extending the previous example with Good morning if time < 10:

```
int time = 8;

if (time < 10) {
printf("Good morning.");
} else if (time < 20) {
printf("Good day.");
} else {
  printf("Good evening.");
}
// Outputs "Good morning."
```

Instead of writing many if..else statements, you can use the switch statement. The switch statement selects one of many code blocks to be executed:

Syntax:

```
switch(expression)
{
   case x:
      // code block
      break;
   case y:
      // code block
      break;
   default:
    // code block
}
```

Example: The example below uses the weekday number to calculate the weekday name:

```
int day = 4;
switch (day)
{
case 1:
   printf("Monday");
   break;
case 2:
```

```
   printf("Tuesday");
   break;
case 3:
   printf("Wednesday");
   break;
case 4:
   printf("Thursday");
   break;
case 5:
   printf("Friday");
   break;
case 6:
   printf("Saturday");
   break;
case 7:
   printf("Sunday");
   break;
default:
   printf("Please enter a day value!");
   break; }

// Outputs "Thursday" (day 4)
```

Notes:

- When C reaches a break keyword, it breaks out of the switch block.
- The default keyword specifies some code to run if there is no case match

Topic 4: Loops

A) While Loop: The while loop loops through a block of code as long as a specified condition is true.

Syntax:

```
while (condition) {
  // code block to be executed
}
```

Example: In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

```
int i = 0;

while (i < 5) {
printf("%d\n", i);
i++;
}
```

B) Do While Loop: The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do {
  // code block to be executed
}
while (condition);
```

Example: The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested.

```
int i = 0;

do {
printf("%d\n", i);
i++;
} while (i < 5);
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

C) For Loop: When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.

Syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.

Example:

```
int i;

for (i = 0; i < 5; i++) {
printf("%d\n", i);
}
```

D) Break/Continue: Just like switch, the break statement can also be used to jump out of a **loop**.

Example: This example jumps out of the **for loop** when i is equal to 4,

```
int i;
for (i = 0; i < 10; i++) {
   if (i == 4) {
      break;
   }
  printf("%d\n", i);
}
```

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Example: This example skips the value of 4,

```
int i;
for (i = 0; i < 10; i++) {
   if (i == 4) {
      continue;
   }
   printf("%d\n", i);
}
```

Topic 5: Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To create an array, define the data type (like int) and specify the name of the array followed by **square brackets []**.

To insert values to it, use a comma-separated list, inside curly braces:

```
int myNumbers[] = {25, 50, 75, 100};
```

To access an array element, refer to its **index number**. Array indexes start with **0**: [0] is the first element. [1] is the second element, etc.

This statement accesses the value of the **first element [0]** in myNumbers:

```
int myNumbers[] = {25, 50, 75, 100};
printf("%d", myNumbers[0]);

// Outputs 25
```

You can loop through the array elements with the for loop. The following example outputs all elements in the myNumbers array:

```
int myNumbers[] = {25, 50, 75, 100};
int i;

for (i = 0; i < 4; i++) {
printf("%d\n", myNumbers[i]);
}
```

Another common way to create arrays, is to specify the size of the array, and add elements later:

```
// Declare an array of four integers:
int myNumbers[4];

// Add elements
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;
```

Topic 6: String

Strings are used for storing text/characters. For example, "Hello World" is a string of characters. Unlike many other programming languages, C does not have a **String type** to easily create string variables. Instead, you must use the char type and create an array of characters to make a string in C:

```
char greetings[] = "Hello World!";
```

To output the string, you can use the printf() function together with the format specifier %s to tell C that we are now working with strings:

```
char greetings[] = "Hello World!";
printf("%s", greetings); //Note that we have to use %c for
single character
```

Topic 7: User Inputs

To get **user input**, you can use the scanf() function.

Here are some examples for taking different types of user input,

Example 1: Single integer input

```
// Create an integer variable that will store the number we get
from the user
int myNum;

// Ask the user to type a number
printf("Type a number: \n");

// Get and save the number the user types
scanf("%d", &myNum);

// Output the number the user typed
printf("Your number is: %d", myNum);
```

Example 2: Multiple input (1 integer and 1 character)

```
// Create an int and a char variable
int myNum; char myChar;

// Ask the user to type a number AND a character
printf("Type a number AND a character and press enter: \n");

// Get and save the number AND character the user types
scanf("%d %c", &myNum, &myChar);

// Print the number
```

```
printf("Your number is: %d\n", myNum);

// Print the character
printf("Your character is: %c\n", myChar);
```

Example 3: Input a string,

```
// Create a string char
firstName[30];

// Ask the user to input some text
printf("Enter your first name: \n");

// Get and save the text
scanf("%s", firstName);

// Output the text
printf("Hello %s", firstName);
```

Topic 8: Memory Location

When a variable is created in C, a memory address is assigned to the variable. The memory address is the location of where the variable is stored on the computer. When we assign a value to the variable, it is stored in this memory address.
To access it, use the reference operator (&), and the result represents where the variable is stored:

```
int myAge = 43;
printf("%p", &myAge); // Outputs 0x7ffe5367e044
```

Note: The memory address is in hexadecimal form (0x..). You will probably not get the same result in your program, as this depends on where the variable is stored on your computer. You should also note that &myAge is often called a "pointer". A pointer basically stores the memory address of a variable as its value. To print pointer values, we use the %p format specifier.

Topic 9: Pointers

Pointers are important in C, because they allow us to manipulate the data in the computer's memory - this can reduce the code and improve the performance. Pointers are one of the things that make C stand out from other programming languages, like Python and Java.
A pointer is a variable that stores the memory address of another variable as its value. A pointer variable points to a data type (like int) of the same type, and is created with the * operator.
The address of the variable you are working with is assigned to the pointer:

```
int myAge = 43;     // An int variable
int* ptr = &myAge;  // A pointer variable, with the name ptr,
that stores the address of myAge

// Output the value of myAge (43)
printf("%d\n", myAge);
```

```
// Output the memory address of myAge (0x7ffe5367e044)
printf("%p\n", &myAge);

// Output the memory address of myAge with the pointer
(0x7ffe5367e044)
printf("%p\n", ptr);
```

In this example, we created a pointer variable with the name ptr, that points to an int variable (myAge). Note that the type of the pointer has to match the type of the variable you're working with (int in our example).

Use the & operator to store the memory address of the myAge variable, and assign it to the pointer. Now, ptr holds the value of myAge's memory address.

Dereferencing:

In the example above, we used the pointer variable to get the memory address of a variable (used together with the & reference operator). You can also get the value of the variable the pointer points to, by using the * operator (the dereference operator):

```
int myAge = 43;     // Variable declaration
int* ptr = &myAge;  // Pointer declaration

// Reference: Output the memory address of myAge with the pointer
(0x7ffe5367e044)
printf("%p\n", ptr);

// Dereference: Output the value of myAge with the pointer (43)
printf("%d\n", *ptr);
```

Note that the * sign can be confusing here, as it does two different things in our code:

- When used in declaration (int* ptr), it creates a pointer variable.
- When not used in declaration, it act as a dereference operator.

Also, there are two ways to declare pointer variables, but the first way is recommended:

```
int* myNum; // Recommended
int *myNum;
```

Pointers and Arrays:

You can also use pointers to access arrays. in C, the name of an array, is actually a pointer to the first element of the array.
The following examples illustrates how the memory address of the first element is the same as the name of the array:

```c
int myNumbers[4] = {25, 50, 75, 100};

// Get the memory address of the myNumbers array
printf("%p\n", myNumbers);

// Get the memory address of the first array element
printf("%p\n", &myNumbers[0]);
```

Result:

```
0x7ffe70f9d8f0
0x7ffe70f9d8f0
```

Since myNumbers is a pointer to the first element in myNumbers, you can use the * operator to access it:

```c
int myNumbers[4] = {25, 50, 75, 100};

// Get the value of the first element in myNumbers
printf("%d", *myNumbers); // Output: 25
```

To access the rest of the elements in myNumbers, you can increment the pointer/array (+1, +2, etc):

```c
int myNumbers[4] = {25, 50, 75, 100};

// Get the value of the second element in myNumbers
printf("%d\n", *(myNumbers + 1));

// Get the value of the third element in myNumbers
printf("%d", *(myNumbers + 2));

// and so on..
```

Topic 10: Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

So, it turns out you already know what a function is. You have been using it the whole time while studying this tutorial! For example, main() is a function, which is used to execute code, and printf() is a function; used to output/print text to the screen:

```c
int main() {
printf("Hello World!");
return 0;
}
```

To create (often referred to as *declare*) your own function, specify the name of the function, followed by parentheses () and curly brackets {}.

Syntax:

```
void myFunction() {
  // code to be executed
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed when they are called. To call a function, write the function's name followed by two parentheses () and a semicolon ; In the following example, myFunction() is used to print a text (the action), when it is called:

```
void myFunction() {
  printf("I just got executed!");
}


int main() {
myFunction(); // call the function
myFunction(); // calling again
myFunction(); // call as many times you want!
return 0;
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Information can be passed to functions as a parameter. Parameters act as variables inside the function. The following function that takes a string of characters with name as parameter. When the function is called, we pass along a name, which is used inside the function to print "Hello" and the name of each person.

```
void myFunction(char name[]) {
   printf("Hello %s\n", name);
}

int main() {
myFunction("Liam");
myFunction("Jenny");
myFunction("Anja");
return 0;
}

// Hello Liam
// Hello Jenny
// Hello Anja
```

Inside the function, you can add as many parameters as you want:

```
void myFunction(char name[], int age) {
  printf("Hello %s. You are %d years old.\n", name, age);
}

int main() {
myFunction("Liam", 3);
myFunction("Jenny", 14);
myFunction("Anja", 30);
return 0;
}

// Hello Liam. You are 3 years old.
// Hello Jenny. You are 14 years old.
// Hello Anja. You are 30 years old.
```

Topic 11: Compiling and Running

First, save your program as a file with the **.c** extension (for e.g. **myProgram.c**). If you are using VSCode, you can compile and run by simply clicking the Run button on VSCode. A new terminal will pop up showing the outputs of the program, and you should be able to type in any inputs in the same terminal. You can also use the online C IDE provided at the start of the lab sheet.

Finally, if you are comfortable with the command line, you can also compile your program using **gcc**. In the terminal navigate to the same directory as your **.c** file. Then:

```
gcc myProgram.c -o myProgram
```

Then to execute it:

```
./myProgram
```