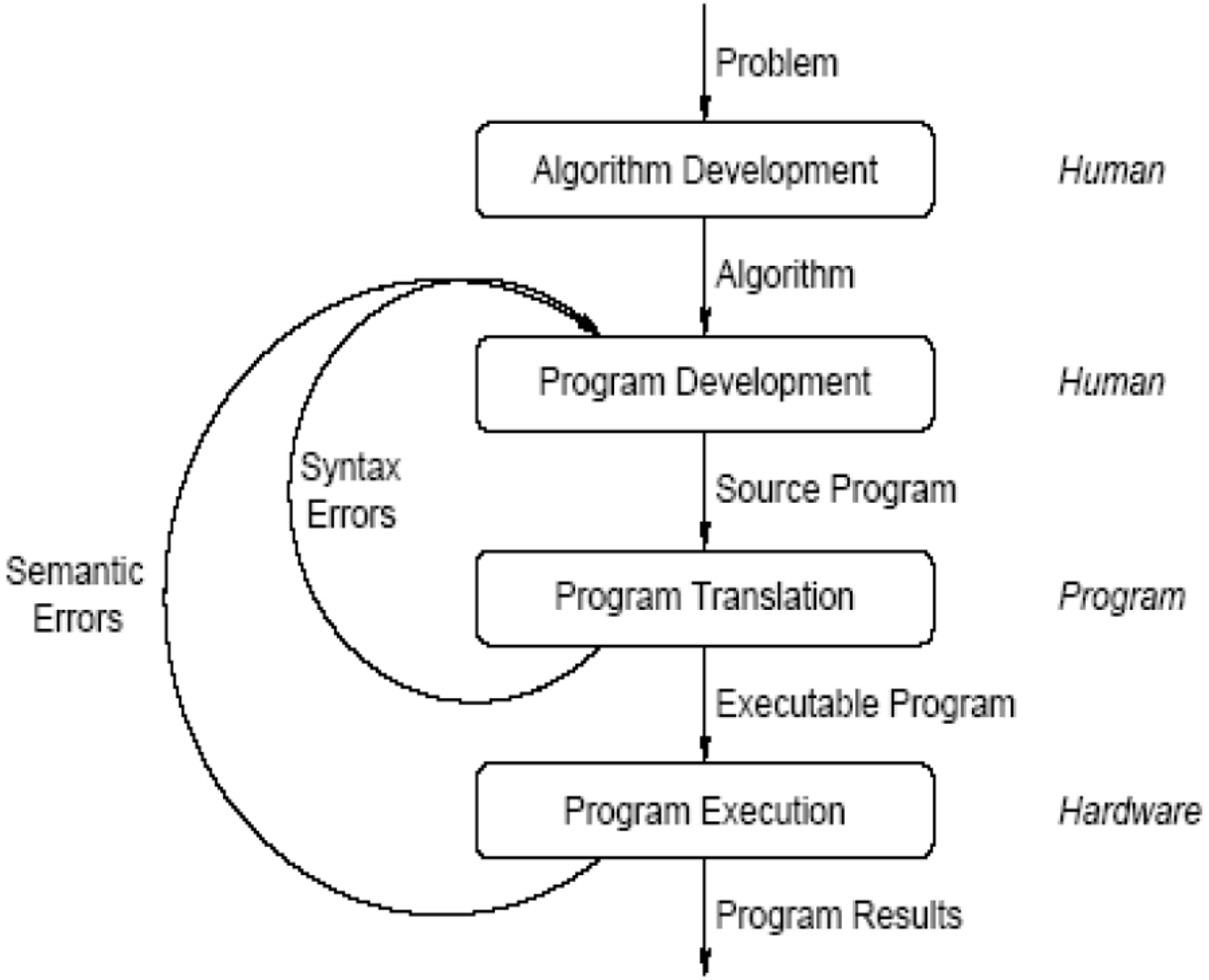
Introduction to C Programming

Learning a Programming Language

- The best way to learn programming is by writing programs.
- Generally, a program is written based on a desire to solve a given problem.



Character Set & Keywords

- Keywords are predefined, reserved words used in programming that have a special meaning to the compiler.
- Keywords are part of the syntax and can not be used for anything else.
- C is case sensitive and all keywords must be written in lower case.

- The C character set is a set of alphabets (lower & upper case), letters and some special characters valid in the C language.

Special Characters in C Programming

,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
'	&	{	}	"
^	!	*	/	|
-	\	~	+	

Identifiers, Constants & Variables

- An identifier is a name given to an entity such as a variable, function, structure, etc.
- A variable is a container (storage area) for holding data.
  - ✓ Variable names are the symbolic representation of a memory location.
  - ✓ The value of a variable can be changed, hence the name variable.
  - ✓ A variable name can only have letters (lower and upper case), digits and underscores.
  - ✓ Digits can not start a variable name.
  - ✓ Variable names need to be meaningful.
- A constant/literal is an identifier whose value can not be altered in the program.

Scope and Lifetime of Variables

- Every variable in C programming has 2 properties
  1) Type
  2) Storage class
- The type property refers to the data type of the variable.
- The storage class property determines the storage, visibility and lifetime of the variable.
- 4 types of storage classes exist
  - ✓ Automatic/local: variables defined inside a function. Such variables are destroyed when the function exits.
  - ✓ External/global: variables declared outside any function and are thus accessible to any function within the program.
  - ✓ Static: variables whose values persist until the end of the program.
  - ✓ Register: these declare register variables that were once believed to be faster than local variables. This is not the case any more.

Data Types

- Data types define the type and size of data associated to a given variable or function.
- Data types can be of the primitive/fundamental kind or of the derived/aggregate kind.
  - ★ Fundamental data types are provided in the C compiler and include:
    - ✓ int, float, double, char, bool etc.
    - ✓ short, long, signed and unsigned are data types that modify fundamental data types.
    - ❑ What is the difference between float and double?
    - ❑ What values can the short, long, signed and unsigned modifiers hold?
  - ★ Derived data types are derived from the fundamental data types and include:
    - ✓ arrays, pointers, structures, union, etc.

Writing & Running C Programs

1. Write text of program (source code) using an editor and save the file with a *.c* extension, for example: *program1.c*

2. Run the compiler to convert the program from source code to an executable or binary file.

3. If all is great, well and good, otherwise the compiler gives errors and warnings, edit source file, fix it and re-compile.

4. Run the program again and see if it works.

C Syntax: The Smallest Program

```
#include <stdio.h>
    //The simplest C program – (comment style for 1-line comments).
    /*The simplest C program that I could write – (comment style for
    multi-line comments)*/
int main ()
{
        }
```

Header Files (.h)

- Files with the *.h* extension are called header files
  - ★ Header files are precompiled files that contain definitions needed to interface to libraries and code in other *.c* files.
- *#include* is a preprocessor directive that inserts header files such as *stdio.h* into your program where required.
- If a header file is included more than once, the compiler would process the contents of the header files as many times as it has been included and this would result into errors.
  - ★ List at least 5 other header files and determine the overall functionality they provide in a program.
  - ★ Why are header files inserted in angle brackets?
  - ★ What is the purpose of the # in the #include statement?

stdio.h

- *stdio.h* is the standard input output header file.
  - ★ It contains prototypes/functions related to the input/output operations such as *printf(), scanf(), getchar()* etc.
- Without including the *stdio.h,* one would not be able read input from a keyboard or write output to a monitor.
- The *stdio.h* file contains 3 variable types, several macros and various functions for performing input/output operations.
- When the compiler finds the *#include* (preprocessor directive), it dumps the code in the requested *.h* file into position just before the program is compiled.
- Without including the stdio.h file, the compiler wouldn't understand what functions such as *printf(), scanf()*, etc do.
  - ★ List at least 10 other stdio.h functions that support input/out operations.

Documentation

- Documentation refers to the process of adding comments within one's program to facilitate usability and reusability.
- Comments in a program can be inserted in any of the following two ways.

1) //. . . . . Used for single line comments. When the compiler finds the //, it ignores anything after it on the same line.
2) /* . . . . . .*/ Used for multi-line comments. When the compiler finds the /* it ignores anything else after it until the closing */

int main()

- *int main()* is the entry point of a program and is the starting point during execution.
  - ✓ It is compulsory in every C program.
- After the *main()* function, the program flows as per the coder's choice.
- The *main()* is a function of the *int* data type and thus returns an integer value to the system to signify the successful (0) or unsuccessful (-1) execution of a program.
- The return value from the main() function provides the exit status to the operating system.
  - The *main()* function can be called without any arguments or with any number of arguments (to be discussed later).
  - Other variations of declaring the main() function may work but are not recommended.

Lexical Scope

- Blocks of code defined within curly braces define lexical scope.
- The scope of the *main()* function applies from the first curly brace to the matching closing brace.
- Lexical scope is generally defined through functions, which define a block of code to perform a specific task.

Functions

- A function is a block of code that performs a specific task. Functions can be in-built in C (standard library functions) or user-defined when required.

- Standard library (C library) functions:
  - ✓ Are in-built to handle tasks such as mathematical computations, input/output (I/O) processing, string handling, etc.
  - ✓ Have their prototypes and data functionalities defined within appropriate header files.
  - ✓ For example, printf(), scanf(), fprintf(), getchar() are standard library functions defined in the stdio.h header file.
  - ✓ All standard library functions are available for use in a program after the appropriate header file(s) are included.
  - ✓ All C programs must have the main() function for proper execution.

- Library functions are advantageous because.
  - ✓ They work! They have been tested rigorously.
  - ✓ They are optimized for performance. They allow you to create the most efficient code optimized for maximum performance.
  - ✓ They save considerable development time. Don't reinvent the wheel by development a function that performs a task that has already been developed.
  - ✓ The functions are portable. They will work everywhere any time.

User-defined Functions

- User-defined functions are developed/defined by the programmer according to their need.
- Such functions are written based on a standard syntax (prototype) that specifies a function's name, parameters and return type.
  - ✧ The function prototype is just the declaration of the function that gives information to the compiler to indicate that the function would be used later on in the program.
  - ✧ Function prototypes follow the format:

    returnType functionName(dataType1 argument1, dataType2 argument 2, ... dataTypeN argumentN);
- All functions must have a return type and may/ may not have defined arguments.
  - ✧ After declaring a user-defined function as above, the inner workings of the function itself are defined within the main() under a defined lexical scope for the function.
  - ✧ Since execution of the program starts at the main(), when the compiler encounters a user-defined function within the main, control of the program jumps to the user function.
- User-defined functions are advantageous because:
  - ✓ They are easier to understand, maintain and debug; easily reusable; and allow for a large program to be divided into smaller modules.

Function Returns & Arguments

returnType functionName(dataType1 argument1, dataType2 argument 2, ... dataTypeN argumentN);

- Based on the above function prototype, one can define a function that gives the summation of numbers as:

  int sumNumbers(int number1, int number2, ... int numberN)

- Such a function provides the following information to the compiler:
  - ✓ Function name: *sumNumbers*
  - ✓ Function return type: int
  - ✓ Function arguments: several with the same data type of *int*
- Control of the program is transferred to the user-defined function by calling it. (To be discussed later).
- The operation specifications of a user-defined function are defined within the functions scope.
- Arguments in a function refer to the variable(s) that a function needs to operate on for a specified task.
  - ✓ The arguments passed to a function must match the data types defined in the function declaration, otherwise the compiler throws errors. A function can also be called without passing arguments.
- Once a function has executed, it returns a value to the calling function thus transferring program control back to the calling function.
  - ✓ The type of the value returned to the calling function must match that which was defined for the called (user-defined) function.

User-defined Function - Example

Example: Program to add two integers

```
#include<stdio.h>
int add2Numbers(int a, int b);   //user-defined function prototype
int main()
  {
        int firstNumber, secondNumber, numberSum;
        printf("Enter 2 numbers separated by a space:  ");
        scanf("%d %d", &firstNumber, &secondNumber);

        numberSum = add2Numbers(firstNumber, secondNumber);      //function call

        printf("The sum of the numbers entered = %d",numberSum);

        return 0;
   }

int add2Numbers(int a, int b)     //Function definition
  {
        int sum;
        sum = a +b;
        return sum;
  }
```