STRUCTURED PROGRAMMING

DEFINITION OF TERMS

1. **HEADER FILE** - A header file is a file with extension **.h** which contains C function declarations to be shared between several source files. A header file is used in a program by including it with the use of the preprocessing directive **#include**, which comes along with the compiler. **#include<stdio**.h>

2. **EXPRESSION –** These are statements that return a value. Expressions combine variables and constants to create new values or logical conditions which are either true or false e.g.
   `x + y, x <= y etc`

3. **KEYWORD -** A keyword is a reserved word in C. Reserved words may not be used as constants or variables or any other identifier names. Examples include auto, else, Long, switch, typedef ,break etc

4. **IDENTIFIER -** A C **identifier** is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9). C does not allow punctuation characters such as @, $, and % within identifiers.

5. **COMMENT -** These are non-executable program statements meant to enhance program readability and maintenance- they document the program.

6. **FUNCTION -** A function is a group of statements, enclosed within curly braces, which together perform a task.

7. **STATEMENT-** Statements are expressions, assignments, function calls, or control flow statements which make up C programs. Statements are terminated using a semicolon.

8. **SOURCE CODE –** Program instructions in their original form. C source code files have an extension **.c**

9. **OBJECT CODE –** Code produced by a compiler from source code and exists in machine readable language.

10. **EXECUTABLE FILE –** Refers to a file in a format that a computer can directly execute and is created by a compiler.

11. **STANDARD LIBRARY –** Refers to a collection of precompiled functions/routines that a program can use. The routines are stored in object format and contain descriptions of functions to perform I/O, string manipulations, mathematics etc.

12. **SIGNED INTEGER –** This is an integer that can hold either positive or negative numbers.

13. **COMPILER –** This is a program that translates source code into object code.

14. **PREPROCESSOR COMMAND -** The C preprocessor modifies a source file before handing it over to the compiler for instance by including header files with #include as I #include <stdio.h>

15. **LINKER/Binder/Link Editor –** This is a program that combines object modules to form an executable program. The linker combines the object code, the start up code and the code for

C DATA TYPES

In the C programming language, data types refer to a system used for declaring variables or functions of different types. **A data type is, therefore, a data storage format that can contain a specific type or range of values**. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The basic data types in C are as follows:

Type	Description
Char	Character data and is used to hold a single character. A character can be a letter, number, space, punctuation mark, or symbol - 1 byte long
Int	A signed whole number in the range -32,768 to 32,767 - 2 bytes long
Float	A real number (that is, a number that can contain a fractional part) – 4 bytes
Double	A double-precision floating point value. Has more digits to the right of the decimal point than a float – 8 bytes
Void	Represents the absence of type. i.e. represents "no data"

USING C'S DATA TYPE MODIFIERS

The five basic types (int, float, char,double and void) can be modified to your specific need using the following specifiers.

- **Signed**
  Signed Data Modifier implies that the data type variable can store positive values as well as negative values.
  The use of the modifier with integers is redundant because the default integer declaration assumes a signed number. The signed modifier is used with char to create a small signed integer. Specified as signed, a char can hold numbers in the range -128 to 127.
- **Unsigned**
  If we need to change the data type so that it can only store positive values, "unsigned" data modifier is used.
  This can be applied to char and int. When char is unsigned, it can hold positive numbers in the range 0 to 255.

- **Long**
  Sometimes while coding a program, we need to increase the Storage Capacity of a variable so that it can store values higher than its maximum limit which is there as default.
  This can be applied to both **int** and **double.** When applied to **int,** it doubles its **length**, in bits, of the base type that it modifies. For example, an integer is usually 16 bits long. Therefore a **long int** is 32 bits in length. When **long** is applied to a double, it roughly doubles the precision.
- **Short**
  A "short" type modifier does just the opposite of "long". If one is not expecting to see high range values in a program.
  For example, if we need to store the "age" of a student in a variable, we will make use of this type qualifier as we are aware that this value is not going to be very high

The type modifier precedes the type name. For example this declares a **long integer.**

long int age;

After Pass #3:	94	91	86	84	76	69	1
After Pass #4 (done):	94	91	86	84	76	69	1

First Pass: *d* = (6 + 1) / 2 = 3. Compare 1st and 4th , 2nd and 5th, and 3rd and 6th items since they are 3 positions away from each other))
Second Pass: value for *d* is halved *d* = (3 + 1) / 2 = 2. Compare items two places away such as 1st and 3rd ……
Third Pass: value for *d* is halved *d* = (2 + 1) / 2 = 1. Compare items one place away such as 1st and 2nd …..
Last Pass: sort continues until *d* = 1 **and** the pass occurs without any swaps.

This sorting process, with its comparison model, is an efficient sorting algorithm.

//Shell Sort Function for Descending Order

```
void main()
{
    int I,d , temp,  length[5];
     while( (d > 1))      // boolean flag (true when not equal to 0)
    {
        d = (d+1) / 2;
        for (i = 0; i < (5 - d); i++)
      {
            if (num[i + d] > num[i])
            {
                  temp = num[i + d];      // swap positions i+d and i
                  num[i + d] = num[i];
                  num[i] = temp;
                  flag = 1;                  // tells swap has occurred
            }
       }
    }
    return;
}
```


Quick Sort

The **quicksort** is considered to be very efficient, with its "divide and conquer" algorithm. This sort starts by dividing the original array into two sections (partitions) based upon the value of the first element in the array. Since our example sorts into descending order, the first section will contain all the elements greater than the first element. The second section will contain elements less than (or equal to) the first element. It is possible for the first element to end up in either section.

Let's examine our same example

Array at beginning:	84	69	76	86	94	91
= 1st partition	86	94	91	84	69	76
= 2nd partition	94	91	86	84	69	76
	94	91	86	84	69	76

elearninginkenya@gmail.com Whatsapp: 0705437337