Abigael	Jeptoo	Bscas/2020/91668.
Susan	Njeri	Bscas/2020/89829.
Enock	Korir	Bscas/2020/89611
Lucas	Waehara	Bscas/2020/89897.
Vivian	Achieng	Bscas/2020/90207.
Fiona	Mutua	Bscas/2020/89888
Kevin	Ngari	Bscas/2020/89762
Faiza	Zahoro	Bscas/2020/90743.
Zulada	Wasai	Bscas/2020/89731
Lawrance	Kipyegon	Bscas/2020/90227
Sharon	Gatwiri	Bscas/2020/69205
Chang	Stephen	BSCAS/2020/92535

Programming Methodology Summary

Chapter 1:

- The unix operating system, the c compiler, and essentially all unix application programs have been written in c.
- It has become a widely used professional language for various reasons i.e.

Easy to learn.
Structured language.
Produces efficient programs.
Can handle low level activities.
It can be compiled on a variety of computer platforms.

examples of the use of c.

- ✓ Operating Systems.
- ✓ Language compilers
- ✓ Assemblers
- ✓ Text Editors
- ✓ Print spoolers
- ✓ Network Drivers
- ✓ Modern programs
- ✓ Databases
- ✓ Language Interpreters
- ✓ Utilities.

Chapter 2 Environment Setup

example using

```
# include <stdio.h>
Int main()
```

```
{ /* my first option program in C*/
  Printf("Hello, world!\n");
  return 0;
  }
```

Local environment setup

This needed to set up your environment for C programming language.

a) Text editor e.g Windows Notepad, OS edit Command, Brief.

b) The C compiler.

<u>Chapter 3: Program Structure.</u>

- Basically consist of the following parts.
- Preprocessor commands.
- Functions
- Variables
- Statement & operations
- Comments.

example

```
# include <stdio.h>
int main()
{ /* my first program in C*/
printf("Hello, world! \n");
return 0;
```

1. The first line of the program #include <stdio.h> is a processor command which tells a C compiler to include stdio.h file before going to actual compilation.
2. The next line int main() is the main function where the program execution begins
3. The next line /*.....*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
4. The next line Printf(...) is another function available in C which causes the message 'Hello world' to be displayed on the screen
5. The next line return 0; terminates the main() function and returns the value 0.

* Compile and Execute C program.
- Open a text editor and add the above-mentioned code
- Save the file as hello.c.
- Open a command prompt and go to the directory where you have saved the file.
- Type gcc hello.c and press enter to compile your

code

- If there are no errors in your code, the command prompt will take you to the next line and would generate a.out executable file.
- Now type a.out to execute your program
- You will see the output "Hello world" printed on the screen.

```
$ gcc hello.c
$ ./a.out
Hello, World!
```

Chapter 4. Basic Syntax.

A program consist of various tokens and a token is either a keyword, an identifier, a constant a string literal or symbol

examples:

```
# Include <stdio.h>
# include <stdio.h>
int main () {
  /* my first program in C */
  Printf ("Hello world\n");
  return 0
}
```

~~It consist of~~ tokens

1. Semicolons - It is statement terminator.

~~Each individual~~

each individual statement must be onded with a semicolon. It indicates the end of one logical entity

e.g Print f ("Hello vivian!\n");

return 0

2) Comments

- Are ignored by the compiler
- They start with /* and terminate */

i.e /* My first day in NMU

3. Identifiers.

Used to identify a variable, function. or any other defined item

Starts with letter A to z', a to z or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9)

4 Keywords

5 Whitespace It is the word used to describe blanks, tabs, newline characters and comments

Chapter 5:

Datatypes:

extensive system used for declaring variables or functions of different types.

Chapter 6:

Variables:

A name given to storage area that our programs can manipulate.

Chapter 7

Constants & Literals:

Refer to fixed values that the program may not alter during its execution. The fixed values are also called Literals.

An integer literal can be a decimal, octal or hexadecimal constant

A prefix specifies the base or radix.

0x or 0X for hexadecimal, 0 for octal and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L for unsigned and long

respectively. The suffix can be uppercase or lowercase and can be in any order.

Floating-point Literals:

Has an integer part, a decimal point, a fractional part and an exponent part.

Character constants

Are enclosed in single quotes, e.g 'x' can be stored in a simple variable of character type.

String literals / constants.

Are enclosed in double quotes

A string contains characters that are similar to character literals: plain characters, escape sequences and universal characters

Defining Constants:

Two ways of defining constants:

- Using # define preprocessor.
- Using constant keyword.

Chapter 8 Storage classes

It defines scope (visibility) and life time of variables and/or functions within a C program. They precede the type that they modify

- Auto
- Register
- Static
- Extern.

Chapter 9: Operators.

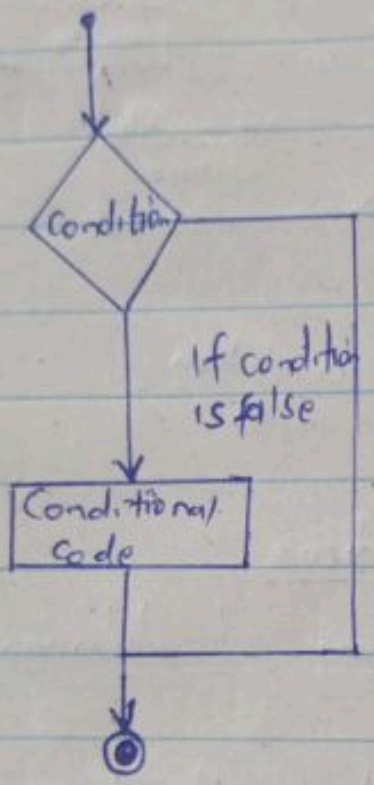
An operator is a symbol that tells the compiler to perform specific mathematical or logical function. C language is rich in built in operators and provides the following types of operators

- Arithmetic operators.
- Relational operators.
- Logical operators.
- Bitwise operators.
- Misc operators.

Chapter 10: Decision Making.

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is

determined to be false.



C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

Chapter 11: Loops.

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages.

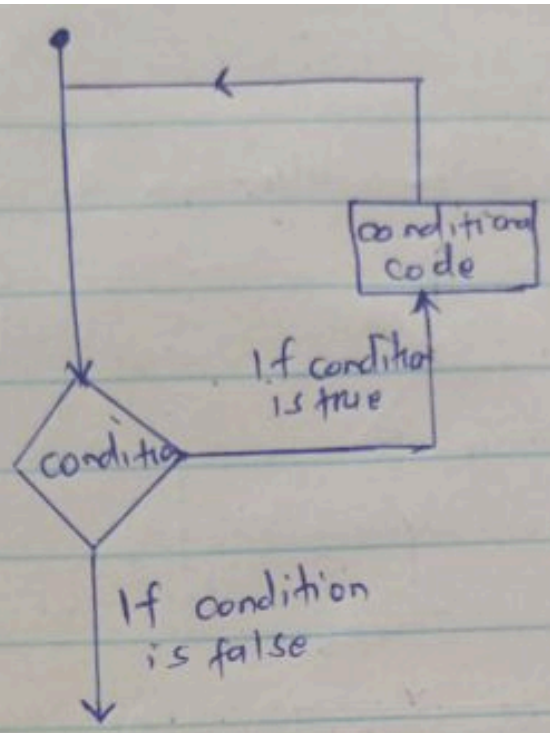
accessed. There are three places where variables can be declared in C programming language.

- Inside a function or a block which is called local variables
- Outside of all functions which called global variables
- In the definition of function parameters which are called formal parameters

Chapter 14: Arrays

Arrays are kind of data structure that can store a fixed-size sequential collection of elements of the same type.

An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.



Chapter 12: Functions

A function is a group of statements that together perform a task.

Every C program has atleast one function, which is main (), and all the most trivial programs can define additional functions.

A function declaration tells the compiler about a function's name, return type and parameters

A function definition provides the actual body of the function.

Chapter 13: Scope Rules

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be

variable.cpp | constants and literals.cpp | Program Structure.cpp

```c
#include <stdio.h>

// Variable declaration:
extern int e, f;
extern int g;
extern float h;

int main()

{
    /*variable defination: */
    int e, f;
    int g;
    float h;

    /* actual initialization */
    e = 30;
    f = 40;

    g = e + f;
    printf("value of e : %f \n{", e);

    h = 50.0/1.0;
    printf("value of h : %h \n", h);

    return 0;
}
```

Messages

variable.cpp | constants and literals.cpp | Program Structure.cp

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

int main ()
{
    /*my first program in C*/
    printf("hi, five! \n");

    return 0;
}
```

Falcon C++

Edit Search View Project Run Tools Help

variable.cpp

```
// Variable declaration:
extern int e, f;
extern int g;
extern float h;

int main()

{
    /*variable defination: */
    int e, f;
    int g;
    float h;

    /* actual initialization */
    e = 30;
    f = 40;

    g = e + f;
    printf("value of e : %f \n(", e);

    h = 50.0 / 2.0;
    printf("value of h : %h \n", h);

    return 0;
}
```

Messages

File	Line	Notification
variable.cpp	21	format '%f' expects type 'double', but argument 2 has type 'int'
variable.cpp	24	unknown conversion type character 0x20 in format
variable.cpp	24	too many arguments for format

variable.cpp

Ln : 27 Col : 2 Sel : 0

le Edit Search View Project Run Tools Help

variable.cpp | constants and literals.c | Program Structure.cpp

```
1   #include <stdio.h>
2
3   int main ()
4   {
5       const int LENGTH = 20;
6       const int WIDTH = 15;
7       const int HEIGHT = 10;
8       const char NEWLINE = '\n';
9       int volume;
10
11      volume = LENGTH*WIDTH*HEIGHT;
12      printf("value of volume : %d" , volume);
13      printf("%c", NEWLINE);
14
15      return 0;
16  }
17
18
```

C
Demo.c

CODE OUTPUT

```c
    #include <stdio.h>
int main(){
  //printing information
  printf("Hello C");
  return 0;
  }
```

TAB { } ( ) " & RUN

C

Demo.c

CODE OUTPUT

```
Hello C
```

```c
#include <stdio.h>
#include <string.h>
struct Employee
{
  int Empolyee_ID;
  int age;
  char Name[50];
  char Department[20];
  float Salary;
};
int main()
{
  struct Employee emp1 = { 101, 25, "Dave", "IT", 25000.50 };
  struct Employee emp2;

      emp2.Empolyee_ID = 102;
      emp2.age = 28;
      strcpy(emp2.Name, "Christofer");
      strcpy(emp2.Department, "Science");
      emp2.Salary = 32000.70;

  printf(" Details of the Employee1 \n " );
  printf(" Employee Id = %d \n ", emp1.Empolyee_ID );
  printf(" Employee Age = %d \n ", emp1.age );
  printf(" Employee Name = %s \n ", emp1.Name );
  printf(" Employee Department = %s \n ", emp1.Department );
  printf(" Employee Salary = %.2f \n\n ", emp1.Salary );

  printf(" Details of the Employee1 \n " );
  printf(" Employee Id = %d \n ", emp2.Empolyee_ID );
  printf(" Employee Age = %d \n ", emp2.age );
  printf(" Employee Name = %s \n ", emp2.Name );
  printf(" Employee Department = %s \n ", emp2.Department );
  printf(" Employee Salary = %.2f \n ", emp2.Salary );
  return 0;
}
```

C:\Users\Hp\Documents\Projects\NewFile.e...

```
Details of the Employee1
 Employee Id = 101
 Employee Age = 25
 Employee Name = Dave
 Employee Department = IT
 Employee Salary = 25000.50

 Details of the Employee1
 Employee Id = 102
 Employee Age = 28
 Employee Name = Christofer
 Employee Department = Science
 Employee Salary = 32000.70

Process returned 0   execution time : 0.602 s
Press any key to continue.
```

```c
#include<stdio.h>

//PROGRAM TO FIND AVERAGE OF n NUMBERS USING ARRAYS
 int main()
 {
int marks[10],i,sum=0,n,average;

printf("enter number of elements ");
scanf("%d",&n);
for(i=0;i<n;++i)
{
    printf("Enter number %d ",i+1);
    scanf("%d",&marks[i]);
    sum+=marks[i];
    /*meaning we use braces after for since its anothet statement*/

}
average=sum/n;
printf("Avarage=%d",average);

     return 0;
 }
```

C:\Users\Hp\Documents\Projects\NewFile.exe

```
enter number of elements
```

```c
#include <stdio.h>
#include<stdlib.h>
int main(){
    int age = 30;
    printf("memory address for age is:%p\n",&age);
    return 0;
}
```

C:\Users\Hp\Documents\Projects\NewFile.exe

```
memory address for age is:0060FEFC

Process returned 0   execution time : 0.380 s
Press any key to continue.
```

```c
int add( int, int);


int main()
{
    int m = 20,n = 30,sum;
    sum = add(m,n);
    printf("sum is %d",sum);


}
int add(int a,int b)
{
    return(a+b);
}
```

C:\Users\Hp\Documents\Projects\NewFile.exe

```
sum is 50
Process returned 9   execution time : 2.167 s
Press any key to continue.
```

```c
#include <stdio.h>
main()
{
    int m = 5;
    int n = 0;
    while (m > n)
    {
        printf("m = %d n = %d\n",m,n);
        m--;
        n++;}
    }
```

"C:\Users\Hp\Desktop\programming methodology\tttt\bin\Debug\tttt.exe"

```
m = 5 n = 0
m = 4 n = 1
m = 3 n = 2

Process returned 0 (0x0)   execution time : 1.481 s
Press any key to continue.
```

```c
#include<stdio.h>
int main()
{
    int num=19;
    if(num<10)
    {
        printf("The value is less than 10");
    }
    else
    {
        printf("The value is greater than 10");
    }
    return 0;
}
```

C:\Users\Hp\Documents\Projects\NewFile.exe

```
The value is greater than 10
Process returned 0   execution time : 0.995 s
Press any key to continue.
```

Ln : 14 Col : 2 Sel : 0 Compilation success in 4.0.219 s