**EXPERIMENT - 1**

**CLASS: TE CMPN A DATE: 17/02/21**

**NAME: REBECCA DIAS ROLL NO. : 19**

**AIM: (1) CASE STUDY OF DIFFERENT SYSTEM PROGRAMS**

**(2) TO CREATE YOUR OWN LIBRARY**

**THEORY:**

**1) Case study of different system programs**

**1.** **Operating System**

● An operating system (OS) is software that manages computer hardware and software resources and provides common services for computer programs.

● The operating system is an essential component of the system software in a computer system.

● Application programs usually require an operating system to function.

● Examples of operating systems :Microsoft Windows, Apple macOS, Linux, Android and Apple's iOS.

**2.** **Device Drivers**

● A device driver is a program that controls a particular type of device that is attached to your computer.

● There are device drivers for printers, displays, CD-ROM readers, diskette drives, and so on.

● When you buy an operating system, many device drivers are built into the product.

● Example of Device Drivers: Computer printers,Graphic cards,Modems,Network cards,Sound cards.

**3.** **Compiler**

● A compiler reads the whole source code at once,creates tokens, checks semantics, generates intermediate code and executes the whole program

● May involve many passes.

● A compiler reads the whole program even if it encounters several errors.

● It takes a large amount of time to analyze the source code but the overall execution time is comparatively faster.

● It generates the error message only after scanning the whole program. Hence

debugging is comparatively hard.

● Programming languages like C, C++ use compilers.

**4.** **Macro-Processor**

● A macro instruction is a notational convenience for the programmer.

● It allows the programmer to write a shorthand version of a program (module programming).

● The macro processor replaces each macro invocation with the corresponding sequence of statements (expanding).

● A macro processor can –

• Recognize macro definitions

• Save the macro definition

• Recognize macro calls

• Expand macro calls

**5.** **Assemblers**

● An assembler translates assembly language programs into machine code.

● The output of an assembler is called an object file.

• The object file contains the data required to place these instructions in memory and information to enable the loader to prepare the program for execution.

**6.** **Linkers**

● Linker is a computer program that links and merges various object files together in order to make an executable file.

● The major task of a linker is to search and locate referenced modules in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

**7.** **Loaders**

● Loader is a part of the operating system and is responsible for loading executable files into memory and executing them.

● It calculates the size of a program (instructions and data) and creates memory space for it.

● It initializes various registers to initiate execution.

**THEORY:**

- Identify the role and functioning of various system programs over application programs.

## Application Programs

These programs perform a particular function directly for the users. Some of the common application programs include Email, web browsers, gaming software, word processors, graphics software, media player etc.

All of these programs provide an application to the end users, so they are known as application programs. For example: a web browser is used to find information while a gaming software is used to play games.

The requests for service and application communication systems used in an application by a programmer is known as an application program interface (API).

## System Programs

The system programs are used to program the operating system software. While application programs provide software that is used directly by the user, system programs provide software that are used by other systems such as SaaS applications, computational science applications etc.

The attributes of system programming are −

● Using system programming, a programmer can make assumptions about the hardware of the system that the program runs on.

● A low level programming language is used in system programming normally. This is so that the programs can operate in low resource environments easily.

● Most system programs are created to have a low runtime overhead. These programs may have a small runtime library.

● Some parts of the system programs may be directly written in assembly language by the programmers.

● A debugger cannot be used on system programs mostly. This problem can be solved by running the programs in a simulated environment.

- **Compare system defined libraries and user defined libraries**

|  |  |
| --- | --- |
| **System Defined Libraries** | **User Defined Libraries** |
| These functions are predefined in the compiler of C language. | These functions are not predefined in the Compiler |
| These functions are not created by the user as their own. | These functions are created by the user as per their own requirement. |
| Library Functions are stored in a special library file. | User-defined functions are not stored in the library file. |
| In this if the user wants to use a particular library function then the user have to add the particular library of that function in header file of the program | There is no such kind of requirement to add the particular library. |
| Execution of the program does not begin from the library function. | Execution of the program begins from the user-defined function |
| **Example:** printf(), scanf(), sqrt(),…etc. | **Example:** sum(), fact(),…etc. |

- **Need for user defined libraries [industry applications]**

User-defined functions allow programmers to create their own routines and procedures that the computer can follow; it is the basic building block of any program and also very important for modularity and code reuse since a programmer could create a user-defined function which does a specific process and simply call it every time it is needed. Their syntax depends entirely on the programming language or application where they are created.

Though part of any programming language, user-defined functions more commonly refer to the special functions that a user creates as scripts or programs in large systems such as databases or spreadsheets like Microsoft Excel. This is because most of the functions being used in a database system or spreadsheet are built-in functions that the user simply has to call and provide parameters, and most of what the application can do is already being done by one or more built-in functions. In this case, user-defined functions are special custom functions that are meant to do something not normally done by the built-in functions. In programming languages such as C, C++ and Java almost every part of the program is user-defined, hence, these functions are no longer referred to as "user-defined" to separate them from the built-in functions, they are simply called functions.

- **Advantage of using macros in header files**

Macros are used as text substitution. One advantage is that You can use simple small names for a number of statements. You can even define macros for functions.

When writing macros for functions, they save a lot of time that is spent by the compiler for invoking / calling the functions. Hence, The advantage of a macro over an actual function is speed. No time is taken up in passing control to a new function, because control never leaves the home function. The macro just makes the function a bit longer. Macros may make compiling slower but the compiled programs are faster.

- **Steps for creating your own library**

o Open Turbo C (Or any other text-editor )

o Create a file called, for example, myLibrary.h ( The extension ".h" tells that it is a header file)

o Put your functions prototypes in that.

Example in myLibrary.h:

1. #ifndef WHATEVER\_H\_INCLUDED

2. #define WHATEVER\_H\_INCLUDED

3. int foo(int a);

4. #endif

● But you need the implementation of these prototypes, so you create a file called myLibrary.cpp

Example in myLibrary.cpp:

1. #include "myLibrary.h"

2. int foo(int a) {

3. return a + 1;

4. }

● To use your own functions in another program, do like that (this program is saved as "example.cpp":

1. #include "myLibrary.h"

2. int main(){

3. printf("Plus one: %d\n", foo(10)); //Prints 11

4. return 0;

5. }

**IMPLEMENTATION:**

Calculator.h [add, sub, mul, div]

#include<math.h>

int add(int a, int b){

int c ;

c=a+b;

return c;

}

int sub(int a, int b)

{

int c;

c=a-b;

return c;

}

int mult(int a, int b)

{

int c;

c=a\*b;

return c;

}

int div(int a,int b)

{

int c;

c=a/b;

return c;

}

int power (int a ,int b)

{

int c;

c=pow(a,b);

return c;

}

int squareroot(int a)

{

int c;

c=sqrt(a);

return c;

}

int cuberoot(int a)

{

int c;

c=ceil(pow(a,1.0/3.0));

return c;

}

Exp1.C

#include <stdio.h>

#include <conio.h>

#include "calculator.h";

int main()

{

int a,b,c,choice;

clrscr();

printf("Main Menu \n1.addition\n2.subtraction\n3.multiplication\n4.division\n5.power\n6.square root\n7.cube root\n");

scanf("%d",&choice);

printf("Enter a: ");

scanf("%d",&a);

printf("Enter b: ");

scanf("%d",&b);

switch(choice)

{

case 1: c=add(a,b);

printf("Addition : %d \n",c);

break;

case 2: c=sub(a,b);

printf("Subtraction : %d\n",c);

break;

case 3: c=mult(a,b);

printf("Multiplication : %d\n",c);

break;

case 4:c=div(a,b);

printf("Division : %d\n",c);

break;

case 5: c=power(a,b);

printf("Power : %d\n",c);

break;

case 6:c=squareroot(a);

printf("Squareroot : %d\n ",c);

break;

case 7: c=cuberoot(a);

printf("Cuberoot : %d \n",c);

break;

default :

printf()

}

getch();

return 0;

}

**Area.h**

#define square\_area(x) (x\*x)

#define rectangle\_area(x,y) (x\*y)

#define triangle\_area(b,h) ((1.0/2.0)\*b\*h)

#define circle\_area(r) (22\*r\*r/7)

**Area.C**

#include <stdio.h>

#include <conio.h>

#include "area.h";

int main()

{

int choice,a,b,r,ans;

clrscr();

printf("Main Menu \n1.Area of square\n2.Area of rectangle\n3.Area of triangle\n4.Area of circle\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter the side of square : ");

scanf("%d",&a);

ans=square\_area(a);

printf("Area of square : %d",ans);

break;

case 2:printf("Enter the sides of rectangle : ");

scanf("%d %d",&a,&b);

ans=rectangle\_area(a,b);

printf("Area of rectangle : %d",ans);

break;

case 3: printf("Enter base and height of triangle : ");

scanf("%d %d",&a,&b);

ans=triangle\_area(a,b);

printf("Area of triangle : %d",ans);

break;

case 4: printf("Enter radius of a circle : ");

scanf("%d",&a);

ans=circle\_area(a);

printf("Area of circle : %d",ans);

break;

}

getch();

return 0;

}

**Convert.H**

#define c\_to\_f(a) (0.0328 \* a)

#define m\_to\_f(a) (3.28 \* a)

#define l\_to\_c(a) (a / 28.317)

**Convert.C**

#include<stdio.h>

#include<conio.h>

#include "convert.h" ;

int main()

{

float a, ans;

int choice;

clrscr();

printf("Convert \n1.Centimeter to feet\n2.Metre to feet\n3.Litre to cubic feet\nEnter choice : ");

scanf("%d",&choice);

printf("Enter number:");

scanf("%f",&a);

switch(choice)

{

case 1:ans=c\_to\_f(a);

printf("Centemeter to feet : %.2f",&ans);

break;

case 2:ans=m\_to\_f(a);

printf("Meter to feet : %.2f",&ans);

break;

case 3:ans=l\_to\_c(a);

printf("Litre to cubic feet : %.2f",&ans);

break;

}

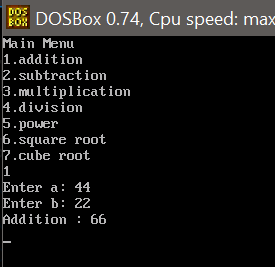
getch();

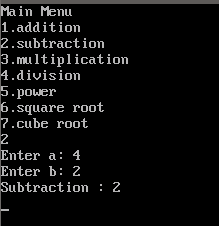
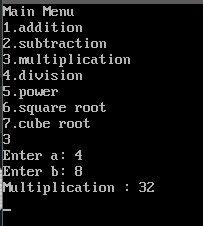
return 0;

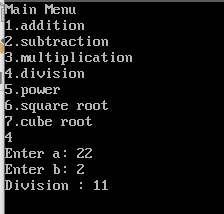
}

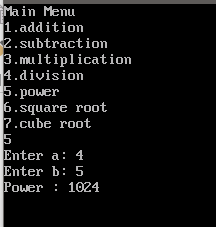
**OUTPUT:**

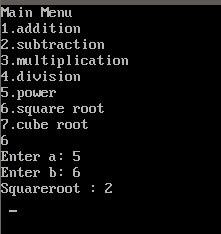
Calculator

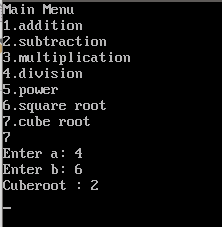


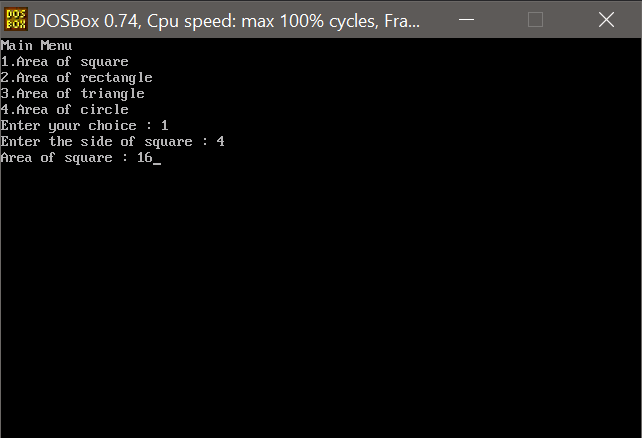


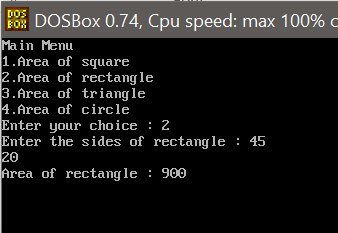


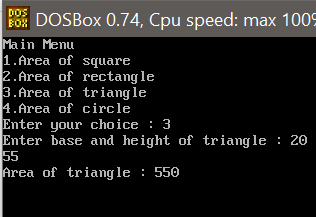


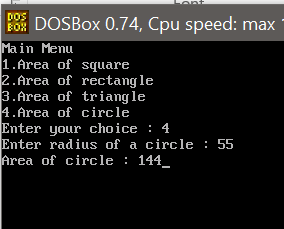


Area

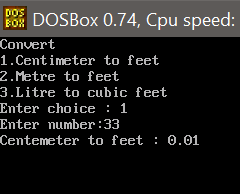


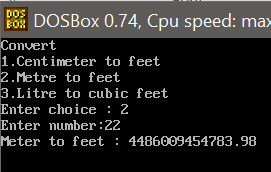


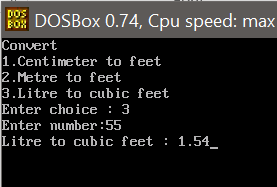




Convert







**CONCLUSION:**

Created user defined header files in C, and used them.