NAME: SARAVANA KUMAR B

USN: 1NH17CS743

PROJECT NAME: EMPLOYEE NETPAY GENERATON

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

- C++ is an object-oriented programming language whereas C is a procedural language.
- C is emphasized on algorithms. On the other hand, C++ is emphasized on data rather than procedure.
- In C language, large programs are divided into smaller programs which is known as functions. On the other hand, C++ programs are divided into what are known as objects.
- In C language, data moves openly around the system from function to function. In C++, new data and functions can be easily added whenever necessary.
- In C language, employs top-down approach in program design. In C++ language, it follows bottom up approach in program design.

1.2 PROBLEM DEFINITION

- Netpay is defined as a method of administrating employees' salaries in the organizations.
- The process consists of calculation of salaries and tax deductions of the employees, administrating the retirement benefits and disbursements of salaries to employees.
- It can also be called as an accounts activity which undertakes the salary administration of employees in the organization.
- Administrating the employees' salaries is not an easy task, the HR and accounts department work together to calculate and disburse the salary to the employees.

1.3 OBJECTIVE OF THE MINI PROJECT

The project is designed to fulfill requirements of employees, saving and retrieving information and salary expectations processing.

- Provides the Adding, Displaying, Modifying and calculations of new pay
- Work out Netpay calculations and deductions quicker
- Calculate Dearness Allowance, House Rent Allowance, Conveyance Allowance, Medical Reimbursement Allowance, City Compensatory Allowance, Performance Incentives, Leave Travel Allowance, Food Allowances
- Generate accurate payslips
- Its reduce the paper work and stores in digital form ,So we can access and analyse the data becomes simple
- The salaries of the employees are determined by the credits they obtain based on their performance

1.4 REQUIRMENTS

Name of the component	Specification
Operating system	Windows 8 and above,Linux
Language	C++
IDE	Turbo C++,Code blocks,Dev-c++
Processor	Intel i3 or above
RAM size	2 GB or above
Hard Disk	200 GB or above
Monitor	15” colour or above
Keyboard	122 keys

Others Knowledge of Basic C++ Programming, Input and accessing data,objects,classes,inheritance and other functions of c++ .

CHAPTER 2

OBJECT ORIENTED FEATURES

OOP consists of many features as- Class, Object, Data encapsulation and Abstraction, Polymorphism, Inheritance etc. These features have been tried to implement in this project. The details about the features are-

2.1 POLYMORPHISM

'Polymorphism' means same name but different functionality. In OOP Polymorphism means functions with same name but different functionality. Polymorphism is of two types. – 1. Compile- time- Polymorphism, 2. Run- time- Polymorphism

Example:

```
 void employee::Display()
{
        printf("Employee Name: %s  ," ,name);
        printf("Employee Designation: %s\n",designation);
       pay::Display();
       printf("\n\n");
}

void Add(char n[30],char d[20],double p,int c)
 {
        int x=no++;
        employee e(n,d,p,c);
        emp[x]=e;
  }

void Search(char n[30])
{
```

```
                for(int i=0;i<no;i++)
 {
              if (stricmp(emp[i].name,n)==0)
{
             printf("ENTRY FOUND:-\n\n");
         emp[i].Display();
                return;
       }
  }
   printf("NO SUCH ENTRY WAS FOUND.\n\n\n");
               return;
 }

      int ESearch(char n[30])
 {
       for(int i=0;i<no;i++)
{
          if (stricmp(emp[i].name,n)==0)
{
             return i;
}
     }
     return -1;
 }
```

This is compile-time polymorphism using function overloading. There are three functions with a same name level but different symptoms to be analysed. Operator overloading with pre-increment operator has been used. Here two Type-1 have two functions with same name but different arguments. Hence it is working as function-overloading.

2.2 DATA ABSTRACTION

Data abstraction means hiding unnecessary data. Here all the data members are in protected mode. All the data are hidden from the outside world which forms the data abstraction.

2.3 INHERITANCE

Inheritance means to inherit something. If one class acquires the properties of another class, then it is called inheritance.

Syntax: class derived-class-name: visibility-mode base-class-name

```
{
        …//
        // members of derived class
};
```

```
class employee: public pay
{

public:
        char name[30];char designation[20];

public:

friend void modify(employee &e);

employee()
{
}

employee(char n[30],char d[20],double p,int c):pay(p,c)
{
```

```
   strcpy(name,n);
        strcpy(designation,d);
}
```

2.1 DATA ENCAPSULATION

Abstraction is one of the key concepts of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation

2.2 OBJECT

Objects are the basic run-time entities in an object-oriented programming system. Object consists of attributes and operations. The attributes hold information as they act as data member. The functions that operate on this data members are called member functions. Syntax- Classname object;

2.4 CLASS

Class is a blue-print of an object. Classes are user defined data types.

```
SYNTAX: Class Classname
                {
                        //Data member;
                        //Member function;
                };
    Example :   class pay
                {

                protected:
```

```
    double basic;int credit;

    inline double newpay()        //inline function
    {
      return basic+ basic*credit*0.02;
    }

public:
    pay()
    {
      basic=0.0;credit=0;
    }
    pay(double b,int c)
    {
      basic=b;
      credit=c;
    }

    void Display()
    {
        cout<<"Employee Basic Pay: "<<basic<<endl;
                        cout<<"Employee Credits: "<<credit<<endl;
        cout<<"Employee New Pay: "<<newpay()<<endl;
    }
};
```

CHAPTER 3

PROJECT DESIGN

3.1 ALGORITHM

Step 0: Start

Step 1: Clear Scr

Step 2: Initialize credits=0

Step 3: Protected Class

Inline Function to calculate Updated Salary using formula,

basic + (basic X credit X 0.02)

Including HRA + PF + DA

Step 4: Declare Public Class -> Implementing Classes and Friend Function

Step 5: Print Basic Pay and Updated Pay

Step 6: Switch Case

a)Add Employee

b)Display Pay of a specific Employee

c)Modify Designation of Employee

d)Exit

Step 7: Add Employee()

Input the details of the Employee

i)Name

ii)Designation

iii)Basic Pay

Step 8: Display() -> Using Run-Time Polymorphism

Print All the Employee Details with their Details.

Step 9: Modify() -> Implementing Friend Function

Input Employee's Name and run through the database and edit the details

Step 10: Exit - exit(0);

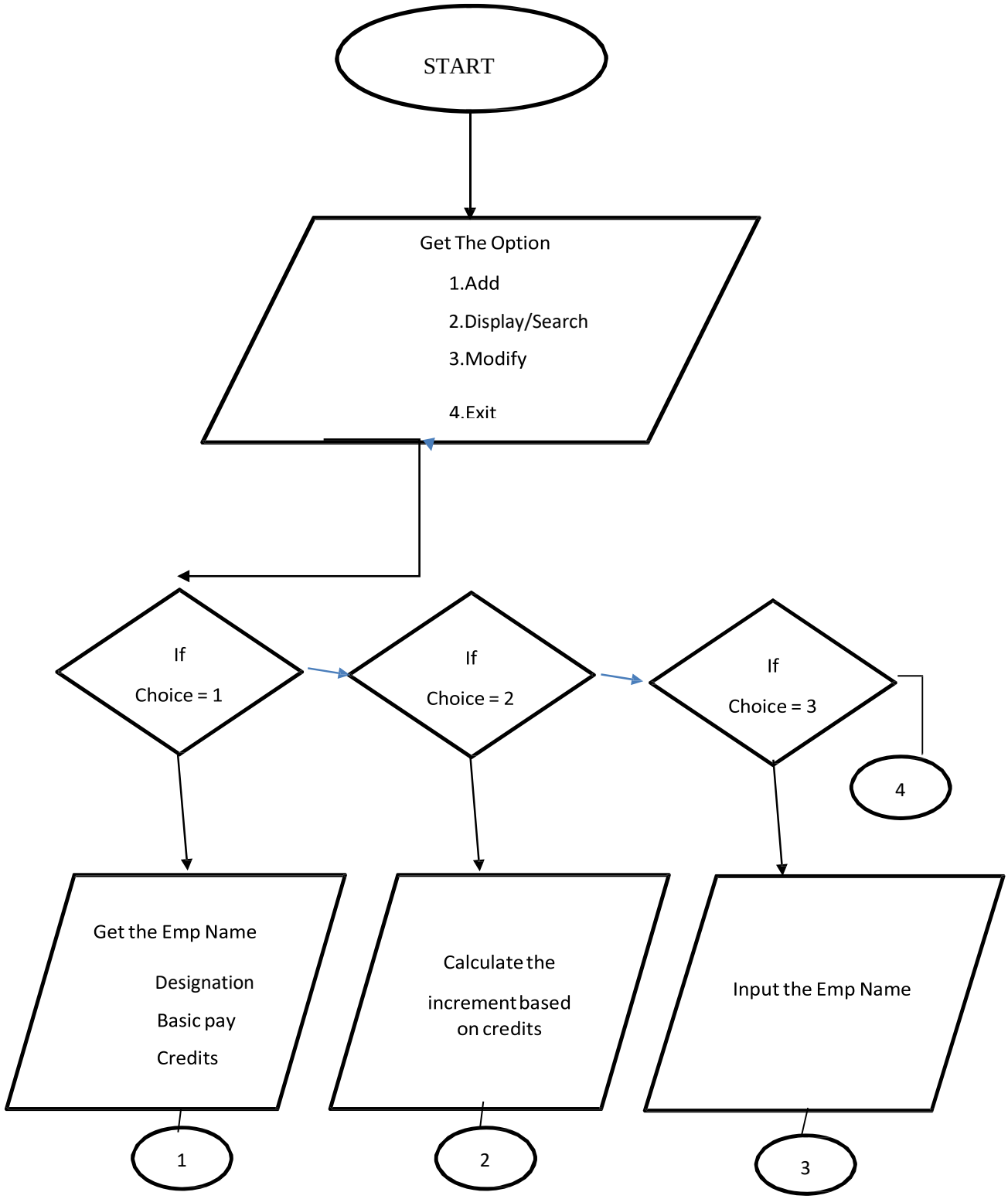
Step 11: Stop

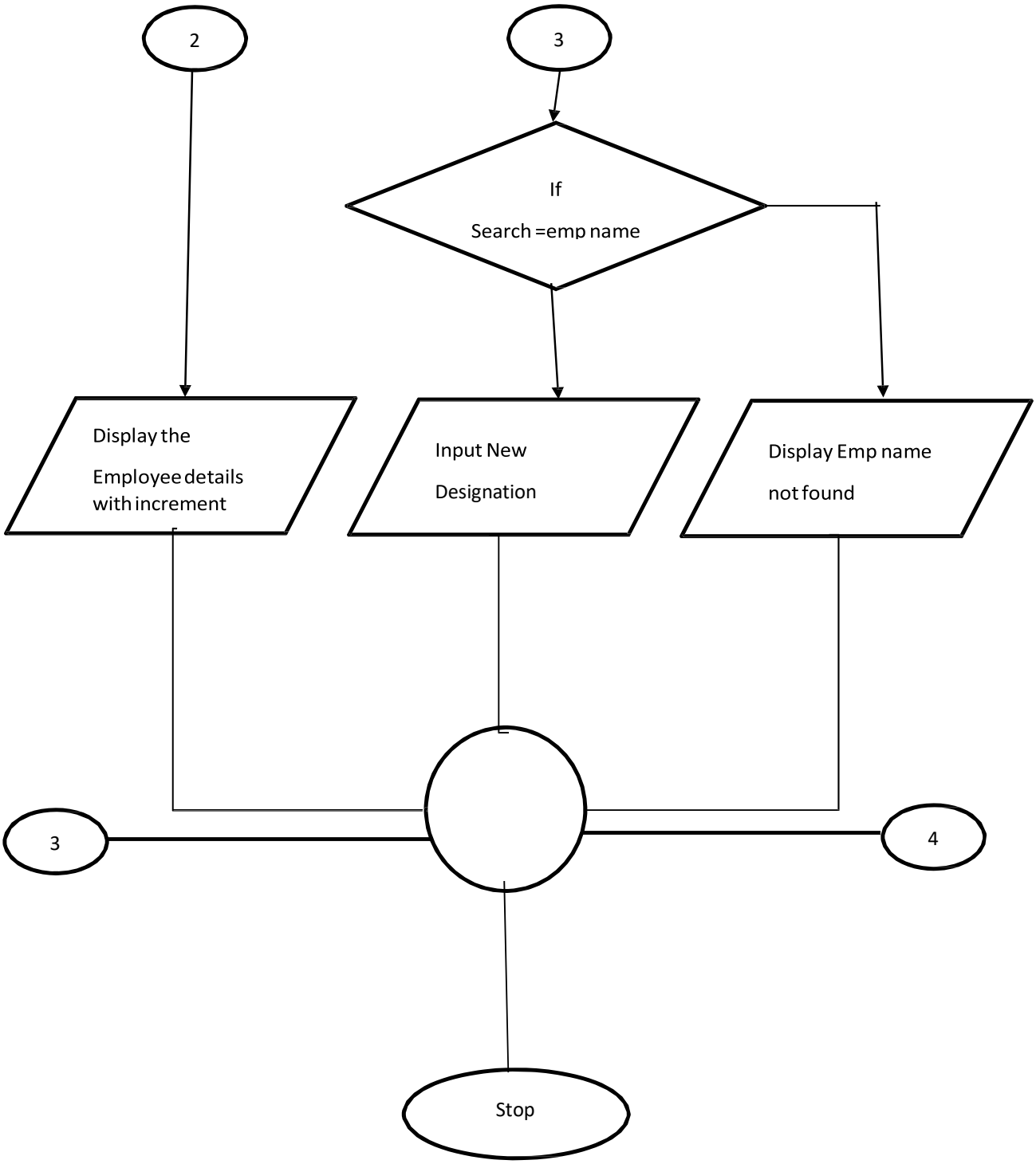
3.2 CLASS DIAGRAM

```
class pay
{

protected:

   double basic;int credit;

   inline double newpay()
            {
      return basic+ basic*credit*0.02;
   }

public:
   pay()
   {
      basic=0.0;credit=0;
   }
   pay(double b,int c)
   {
      basic=b;
      credit=c;
   }
```

```
class employee: public pay
{

public:
      char name[30];char
designation[20];

public:

friend void modify(employee &e);

employee()
{
}

employee(char n[30],char
d[20],double p,int c):pay(p,c)
{
     strcpy(name,n);
       strcpy(designation,d);
}
```

3.2 FLOWCHART




2
3
If
Search =emp name
Display the
Employee details
with increment
Input New
Designation
Display Emp name
not found
3
4
Stop

CHAPTER 4

IMPLEMENTATION

INLINE FUNCTION:

```
class pay
{

protected:

double basic;int credit;

inline double newpay()
{
return basic+ basic*credit*0.02;
}

public:
pay()
{
basic=0.0;credit=0;
}
pay(double b,int c)
{
basic=b;
credit=c;
}

void Display()
{
```

```
cout<<"Employee Basic Pay: "<<basic<<endl;
cout<<"Employee Credits: "<<credit<<endl;
cout<<"Employee New Pay: "<<newpay()<<endl;
}
};
```

INHERITENCE :

```
class employee: public pay
{

public:
char name[30];char designation[20];

public:

friend void modify(employee &e);

employee()
{
}

employee(char n[30],char d[20],double p,int c):pay(p,c)
{
strcpy(name,n);
strcpy(designation,d);
}

void Display();
```

```
}emp[10];
```

FRIEND FUNCTION:

```
void modify(employee &e)
{
char d[20];
cout<<"Enter the new designation of the employee: "<<endl;
cin>>d;
strcpy(e.designation,d);
}
```

FUNCTION OVERLOADING or POLYMORPHISM :

```
void employee::Display()
{
cout<<"*****************************************************\n";
cout<<"Employee Name:   ," ,name;
cout<<"Employee Designation:\n",designation;
pay::Display();
cout<<"*****************************************************\n";
cout<<"\n\n";
}

void Add(char n[30],char d[20],double p,int c)
{
int x=no++;
employee e(n,d,p,c);
emp[x]=e;
```

```
}
void Search(char n[30])
{
for(int i=0;i<no;i++)
{
if (stricmp(emp[i].name,n)==0)
{
cout<<"          ENTRY FOUND:-\n\n";
emp[i].Display();
return;
}
}
cout<<"NO SUCH ENTRY WAS FOUND.\n\n\n";
return;
}

int ESearch(char n[30])
{
for(int i=0;i<no;i++)
{
if (stricmp(emp[i].name,n)==0)
{
return i;
}
}
return -1;
}
```

MAIN FUNCTION:

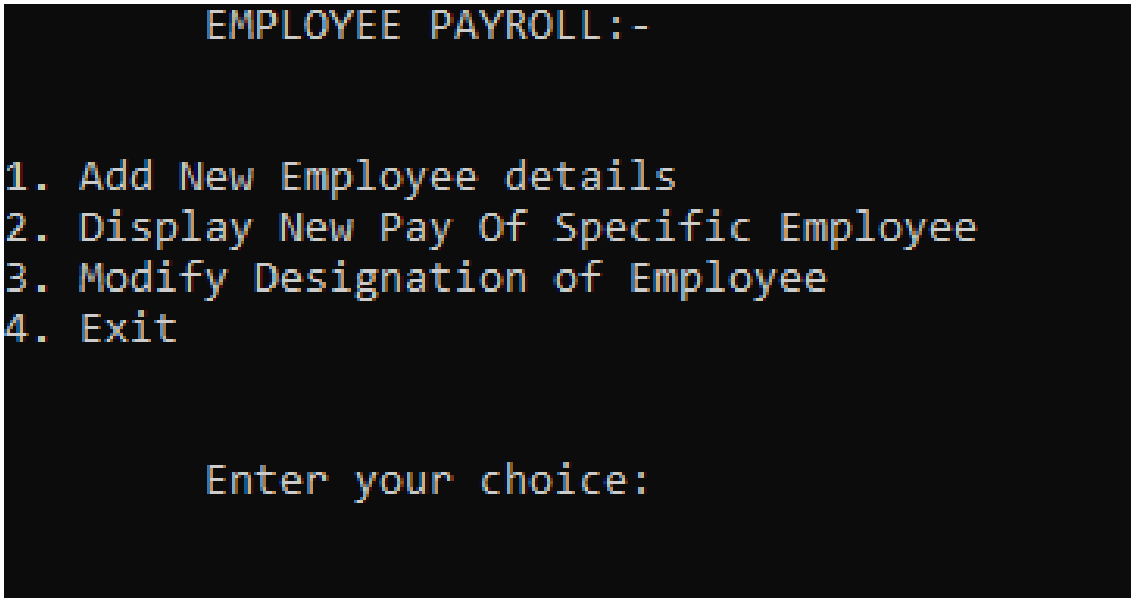
```
void mainscr()
{
```

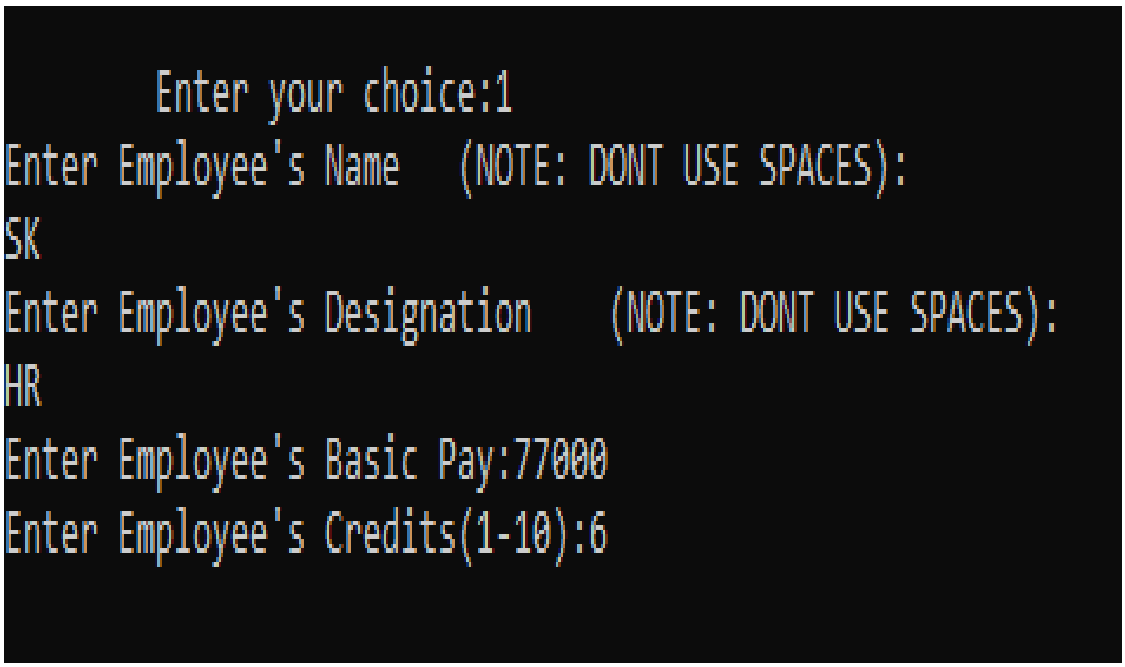
```
int ch=0;char n[30];char d[20];double p=0;int c=0,x;
cout<<"          EMPLOYEE PAYROLL:-\n\n\n";
cout<<"1. Add New Employee details\n";
cout<<"2. Display New Pay Of Specific Employee\n";
cout<<"3. Modify Designation of Employee\n";
cout<<"4. Exit\n\n\n";
cout<<"          Enter your choice:";
cin>>ch;

switch(ch)
{
case 1:
cout<<"Enter Employee's Name
cin>>n;
cout<<"Enter Employee's Designation
cin>>d;
cout<<"Enter Employee's Basic Pay:";
cin>>p;
cout<<"Enter Employee's Credits(1-10):";
cin>>c;
Add(n,d,p,c);
clrscr();
cout<<"\n\n\nENTRY ADDED SUCCESSFULLY.\n\n\n";
mainscr();
break;

case 2:
cout<<"\nEnter Employee's Name
cin>>n;
Search(n);
cout<<"\n\nEnter any number to exit to main screen:";
```

```
cin>>ch;
clrscr();
mainscr();
break;

case 3:
cout<<"\nEnter Employee's Name
cin>>n;
x=ESearch(n);
if(x!=-1){
modify(emp[x]);cout<<"Changes Saved"<<endl;}
cout<<"\n\nEnter any number to exit to main screen:";
cin>>ch;
clrscr();
mainscr();
break;

case 4:exit(0);

default:clrscr();mainscr();;

}

}

int main()
{

clrscr();
mainscr();
```

CHAPTER 5

OUTPUT SNAPSHOTS:





```
        Enter your choice:2

Enter Employee's Name   (NOTE: DONT USE SPACES):
SK
        ENTRY FOUND:-

*****************************************************************
Employee Name: SK  ,Employee Designation: CLERK
Employee Basic Pay: 77000
Employee Credits: 6
Employee New Pay: 86240
*****************************************************************
```

```
        Enter your choice:2

Enter Employee's Name   (NOTE: DONT USE SPACES):
SK
        ENTRY FOUND:-

*****************************************************************
Employee Name: SK  ,Employee Designation: HR
Employee Basic Pay: 77000
Employee Credits: 6
Employee New Pay: 86240
*****************************************************************
```

```
        Enter your choice:3

Enter Employee's Name   (NOTE: DONT USE SPACES):
SK
Enter the new designation of the employee:
CLERK
Changes Saved
```

CHAPTER 6

CONCLUSION

The Projects helps to display the deductions , increments of salary . The project implies the concepts of oops. All the concepts of oops like Dynamic Binding, Polymorphism, Inheritance, Function Overloading, Operator overloading, Virtual function etc… have been implemented in this project. The project is working properly and output is coming as expected