CAPE UNIT 1 COMPUTER SCIENCE INTERNAL ASSESSMENT 2022

School / Centre: Holy Name Convent, Port of Spain

Center Number: 160027

Candidate Name and Number: Joseann Boneo - 160027xxxx

Paris xxxxxx - 160027xxxx

Shurbrandy xxxxxxx- 160027xxxx

Anya-xxxxxxxxxxx-160027xxxx

Completion Date:

Title: John and Brothers Ltd Payroll System

Purpose: To create a manageable and well-functioning Payroll System for a clerk of an up-and-coming business.

TABLE OF CONTENTS

Identification of Specifications

Problem Definition

John and Brothers Ltd.
Founded in 2017, John and Brothers Ltd. is a telecommunications company that has steadily been gaining popularity. However, the company faces a challenge in the accounting department. There is difficulty producing payrolls for the employees as there is no proper system that has been put in place. Due to the unreliable system, the calculation of overtime worked, income tax, national insurance, pension and union subscription is erroneous.

The accounts department is understaffed with payroll clerks and utilizes a manual payroll system. Having to manually calculate pay slips and vacation leaves, the payroll clerk is overworked and is inclined to become frustrated with the tedious task as it is difficult to keep track of piles of documents comprising of manual recording of attendance and time tracking of employees. Miscalculation can lead to employees getting less or more than they are entitled to. The complicated salary structures require more time and attention to ensure accuracy. Mass producing hand-written reports by the payroll clerk results in fatigue which dissuades them from caring about the physical allure and comprehensiveness of the payroll documents when shared within an organization. Apart from preparing pay slips, the payroll clerks also prepare credit control, prepare audits, treat credit and debit notes and maintain the collection of accounts which may often lead to a decrease in productivity.

Manual payroll processing also needs extra attention to make important legislative updates. Faltering on abiding by any of the statutory laws can make the payroll clerk face severe fines or penalties for non-compliance. Staying up-to-date on all the rules and regulations is time consuming but critical. Manual processing of payroll makes the scope of information getting into the hands of the wrong person high. Such leakage of information can end up hurting the business. Since the manual payroll system is prone to human errors or duplications by the payroll clerk, this can overburden the human resource and finance department with mundane activities like documentation and repetitive cross checking, spending their valuable time in carrying out mundane tasks over and over again which can lower their productivity and morale.

By implementing an efficient payroll program, this would enable the human resource department or the payroll administrator to manage payroll operations smoothly and without errors, thereby, eliminating risks related to payroll processing. Not utilizing any software means that employee data cannot be stored, calculations and deductions cannot be computed automatically, making all steps redundant every pay run.

1) **Functional Requirements** The program should be able to calculate the correct salary for each member of the organization. This then means that overtime, either earned or not earned should be accounted for as well as any other necessary information that would either be deducted or added to each employee's pay check i.e. medical tax, insurance tax, gloss pay
2) Furthermore, the program should be able to calculate the hours worked by each individual in the organization. The clerk must be able to source the adequate documents in order to input the information needed for the calculation. However, inputting the information should be all that they are expected to do and the program should easily produce an output.
3) The system(files) should also be sorted by ID number. This identification dumber will not only make the system a bit more organized but is unique to each individual making it easier in the search sequence. This is also done in the tabulated outputs where the employee ID is displayed first.
4) However, for the bubble sort which would only display information such as employee ID, names and job title. The system should be sorted by employee job title in order to keep account of the employees with the same position in the organization.
5) The program should be able to **search** the name of an employee and have their basic work and personal information should appear in an organized and tabulated form. Sort
6) The program should be able to prompt user(clerk) to enter an employee ID and remove their work and personal information from the system (files)
7) Additionally, the program should be able to edit an employee work information. This means that rate of pay, hours worked, overtime hours and days worked would be updated and stored to the system (files).
8) Lastly, the program should be able to add and store a new record into the system. Information on a new record would include personal information such as employee ID, first name, last name and job title to name a few and work information i.e. rate of pay, overtime works and days worked.

Non-Functional Requirements

1) To begin the system should have a password that only trusted members can have access to. This is to ensure the safety of the system and stop any persons with mal intentions who may try to alter data for their own benefit.

2) The system should also back up every thirty minutes as a precautionary measure. This is to ensure no data is lost mistakenly and everything will be present when needed. Even if no new information is added, the system should still undergo this back to ensure utmost security.

3) It should have a ROM chip of 4 GBs to ensure that the system functions well.

4) The system should have a user-friendly interface. This means the system formatting should be easy to follow along and not misleading. If they hire new clerks, said clerks should be able to adapt well and the layout of the interface should not be confusing or difficult for them to understand.

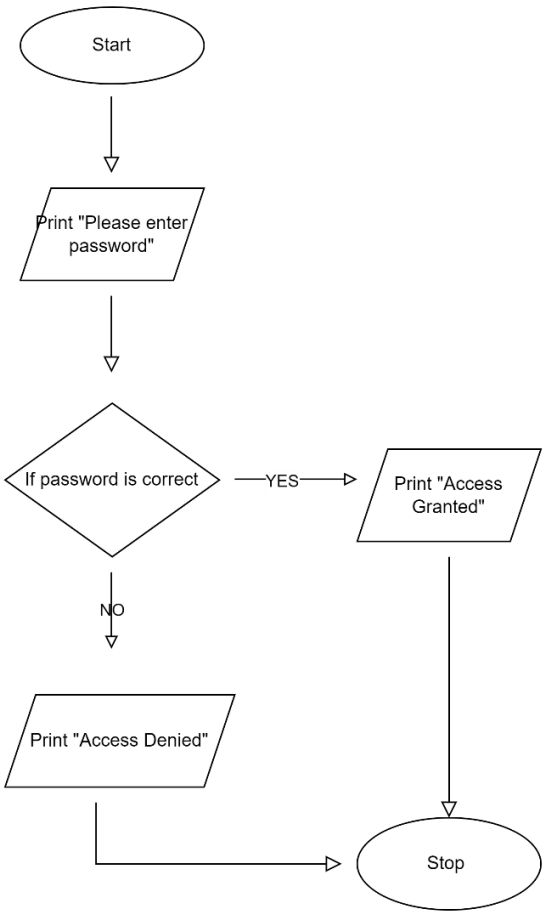
5) The code of the payroll system should contain short notes made by the programmer to assist future users in understanding the inner works of the program. The documentation offers detailed description of each function and purpose of specific complex sections of the code

Design

This algorithm (flowchart) is the graphic representation of the computer program which was made to solve the problem encountered by John and Brothers Ltd. Below, each function within the computer program is represented using a separate flowchart for each of the functions in order to provide a clear and unambiguous flow of the workings of each function.

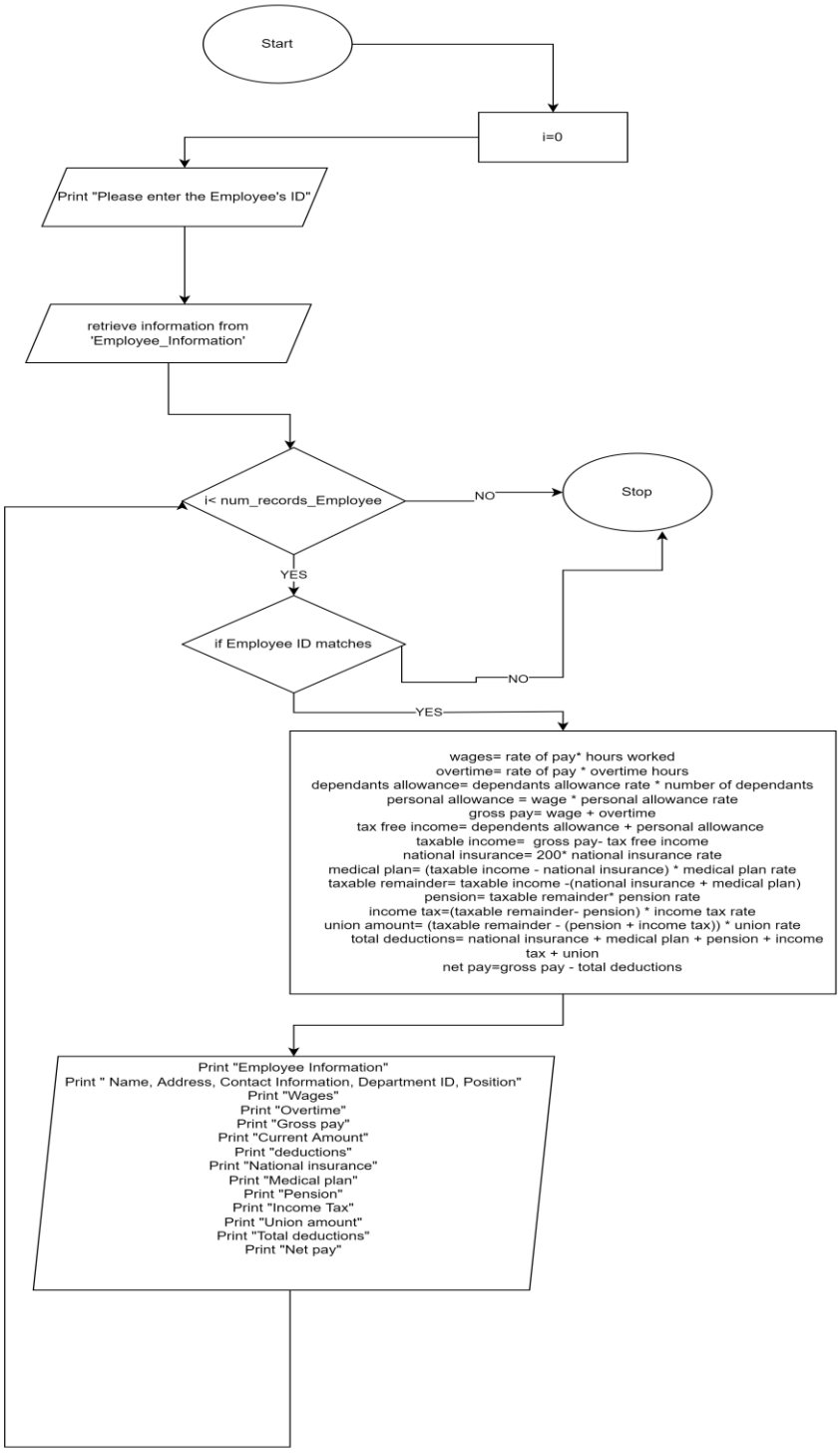
Procedure: Password

This flowchart tracks the input of a password into the computer program.



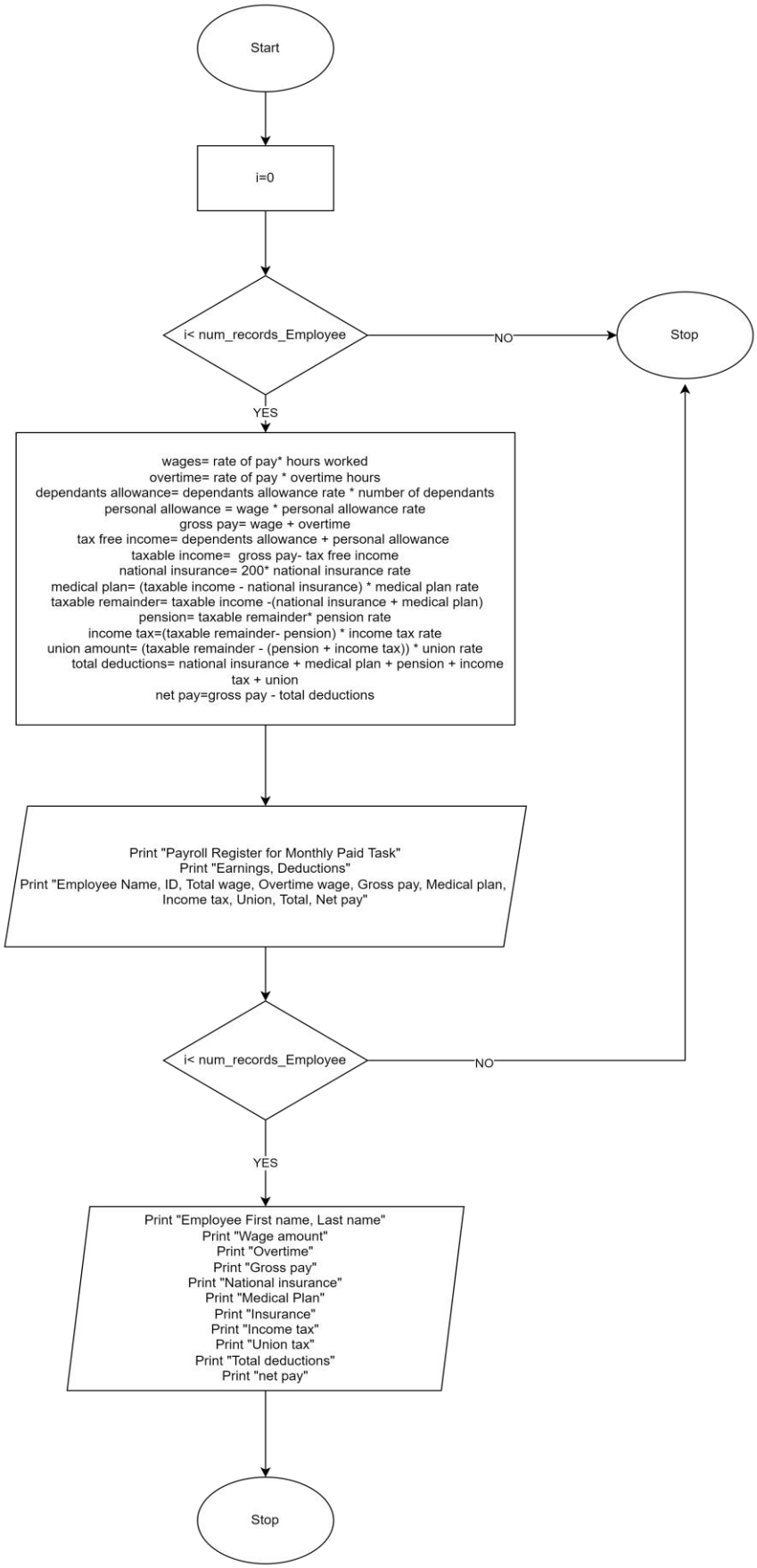
Procedure: Pay slip

This flowchart tracks the input of an employee's ID into the computer program in order to obtain the pay slip of that particular individual



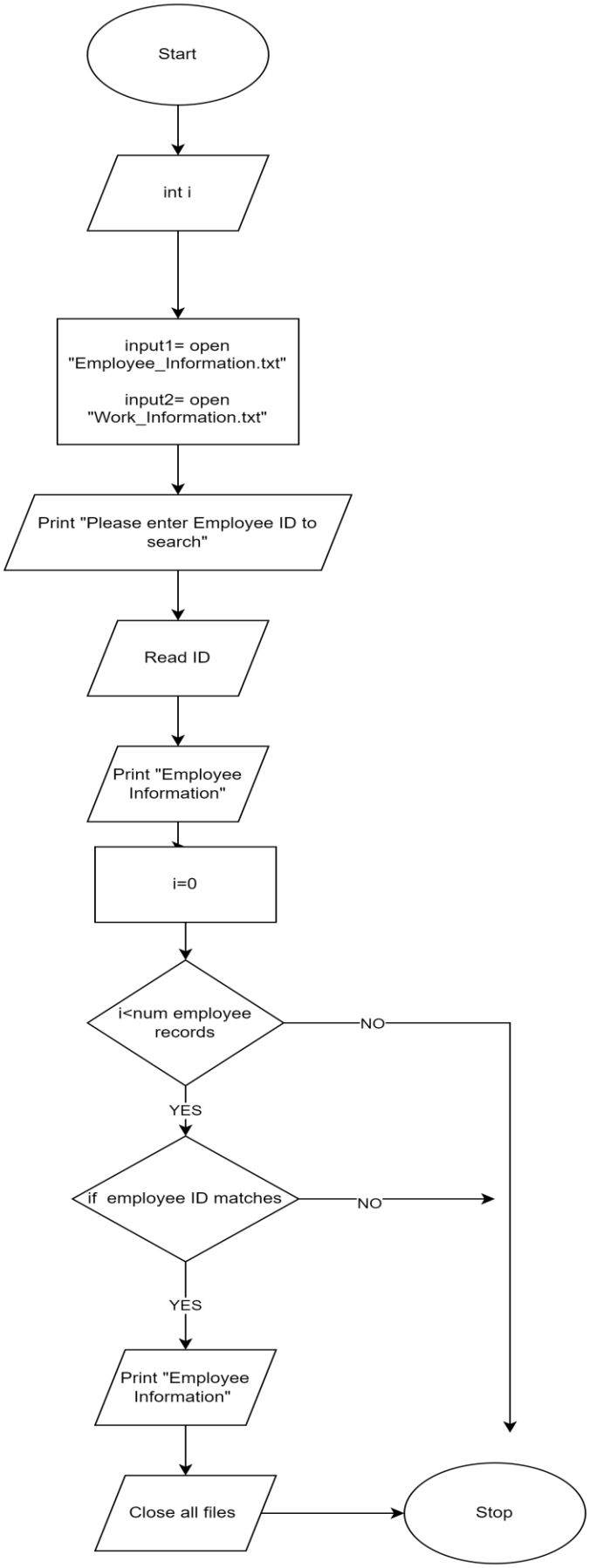
Procedure: Payroll Register

This flowchart tracks the preparation and output of the payroll register.


Start
i=0
i< num_records_Employee
NO
Stop
YES
wages= rate of pay* hours worked
overtime= rate of pay * overtime hours
dependants allowance= dependants allowance rate * number of dependants
personal allowance = wage * personal allowance rate
gross pay= wage + overtime
tax free income= dependents allowance + personal allowance
taxable income= gross pay- tax free income
national insurance= 200* national insurance rate
medical plan= (taxable income - national insurance) * medical plan rate
taxable remainder= taxable income -(national insurance + medical plan)
pension= taxable remainder* pension rate
income tax=(taxable remainder- pension) * income tax rate
union amount= (taxable remainder - (pension + income tax)) * union rate
total deductions= national insurance + medical plan + pension + income tax + union
net pay=gross pay - total deductions
Print "Payroll Register for Monthly Paid Task"
Print "Earnings, Deductions"
Print "Employee Name, ID, Total wage, Overtime wage, Gross pay, Medical plan, Income tax, Union, Total, Net pay"
i< num_records_Employee
NO
YES
Print "Employee First name, Last name"
Print "Wage amount"
Print "Overtime"
Print "Gross pay"
Print "National insurance"
Print "Medical Plan"
Print "Insurance"
Print "Income tax"
Print "Union tax"
Print "Total deductions"
Print "net pay"
Stop

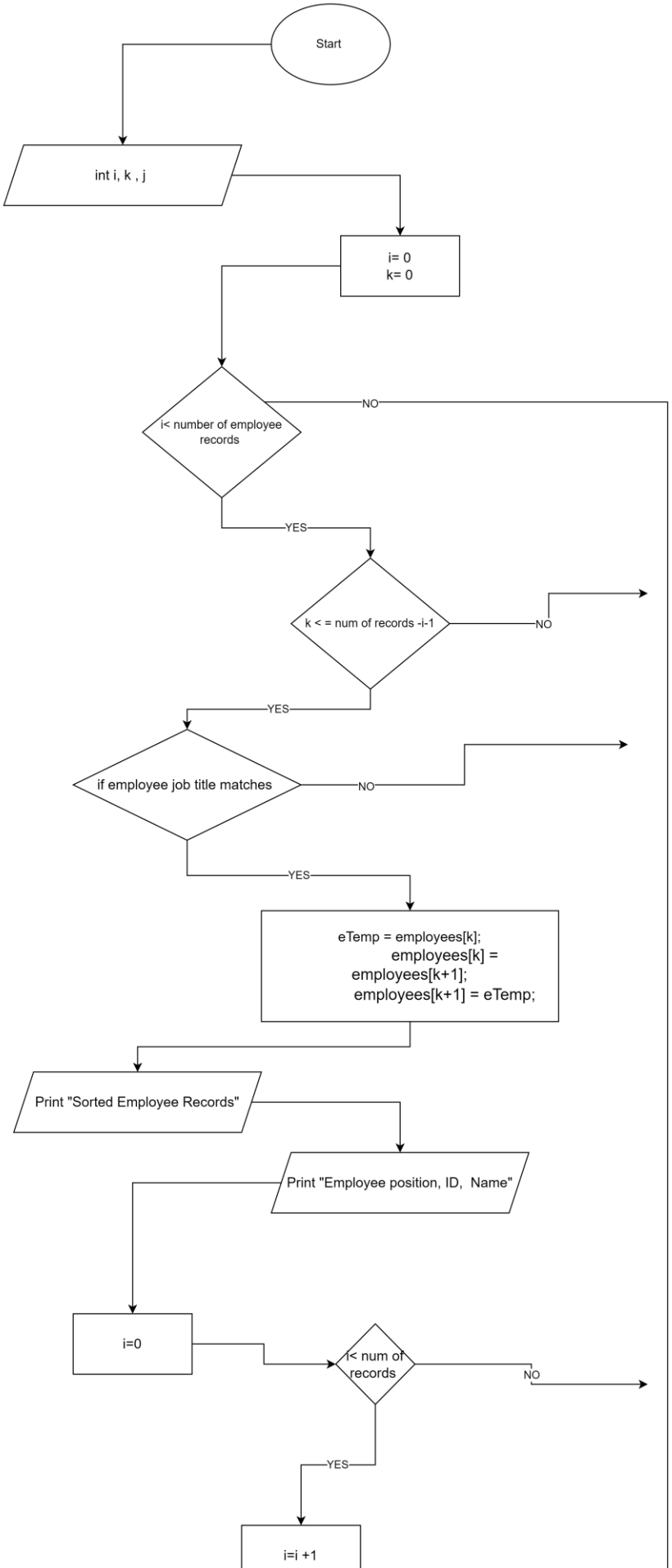
Procedure: Search

This flowchart tracks the searching of a specific employee by entering the employee's ID.


Start
int i
input1= open "Employee_Information.txt"
input2= open "Work_Information.txt"
Print "Please enter Employee ID to search"
Read ID
Print "Employee Information"
i=0
i<num employee records
NO
YES
if employee ID matches
NO
YES
Print "Employee Information"
Close all files
Stop

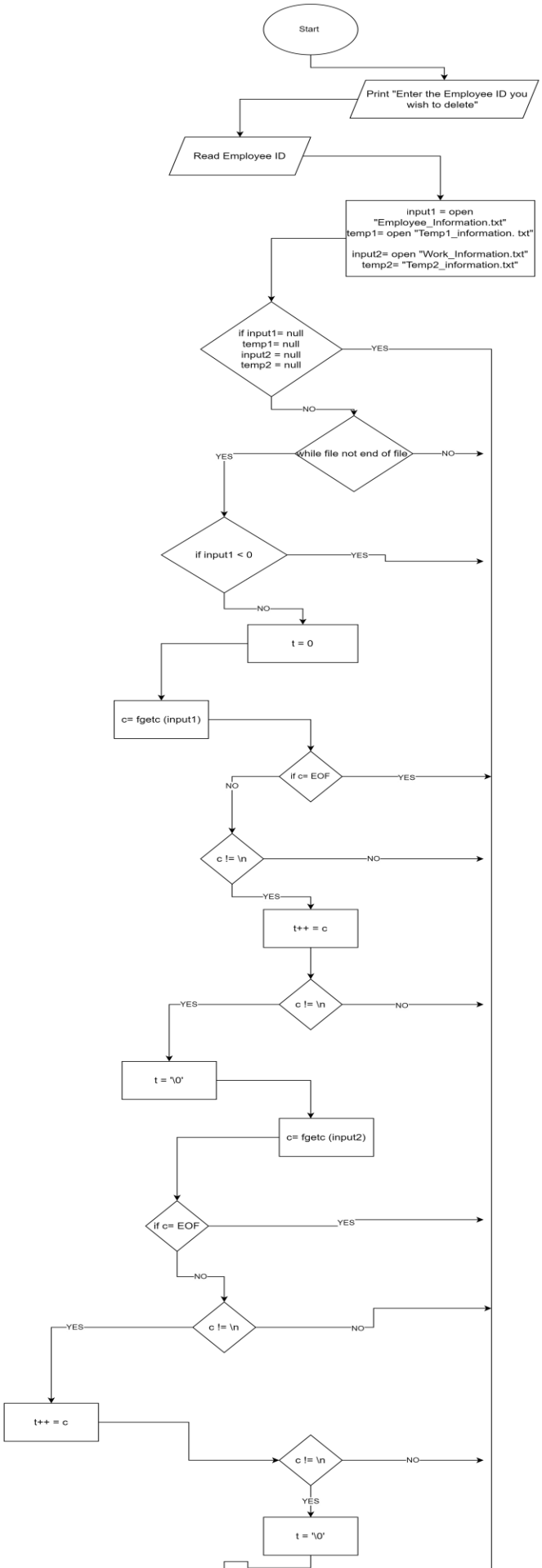
Procedure: Bubble Sort

This flowchart tracks the sorting and printing of the sorted records of employees


Start
int i, k , j
i= 0
k= 0
i< number of employee records
NO
YES
k < = num of records -i-1
NO
YES
if employee job title matches
NO
YES
eTemp = employees[k];
employees[k] =
employees[k+1];
employees[k+1] = eTemp;
Print "Sorted Employee Records"
Print "Employee position, ID, Name"
i=0
i< num of records
NO
YES
i=i +1

Procedure: Delete Record

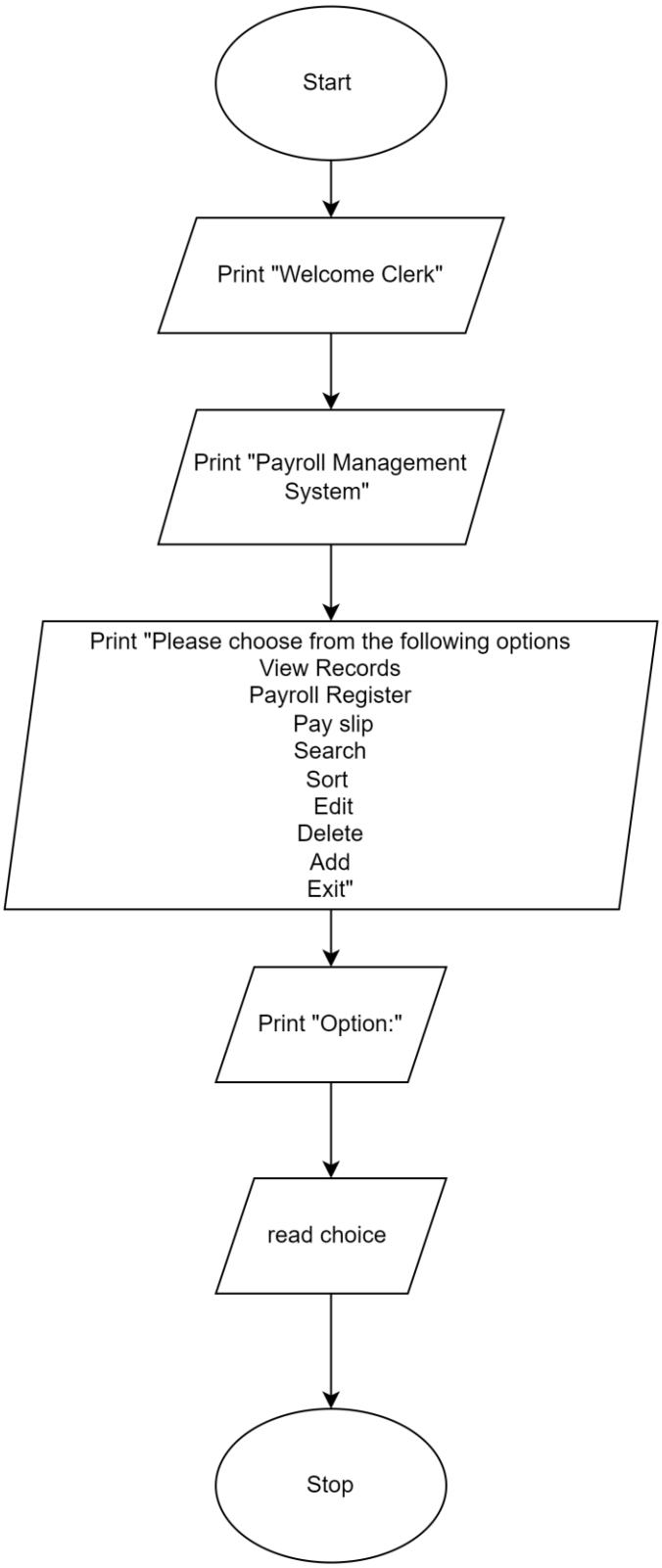
This flowchart tracks the process of the deletion of a particular record


Start
Print "Enter the Employee ID you wish to delete"
Read Employee ID
input1 = open "Employee_Information.txt" temp1= open "Temp1_information. txt"
input2= open "Work_Information.txt" temp2= "Temp2_information.txt"
if input1= null temp1= null input2 = null temp2 = null
YES
NO
while file not end of file
YES
NO
if input1 < 0
YES
NO
t = 0
c= fgetc (input1)
if c= EOF
YES
NO
c != \n
NO
YES
t++ = c
c != \n
YES
NO
t = '\0'
c= fgetc (input2)
if c= EOF
YES
NO
c != \n
YES
NO
t++ = c
c != \n
NO
YES
t = '\0'

Procedure: Menu

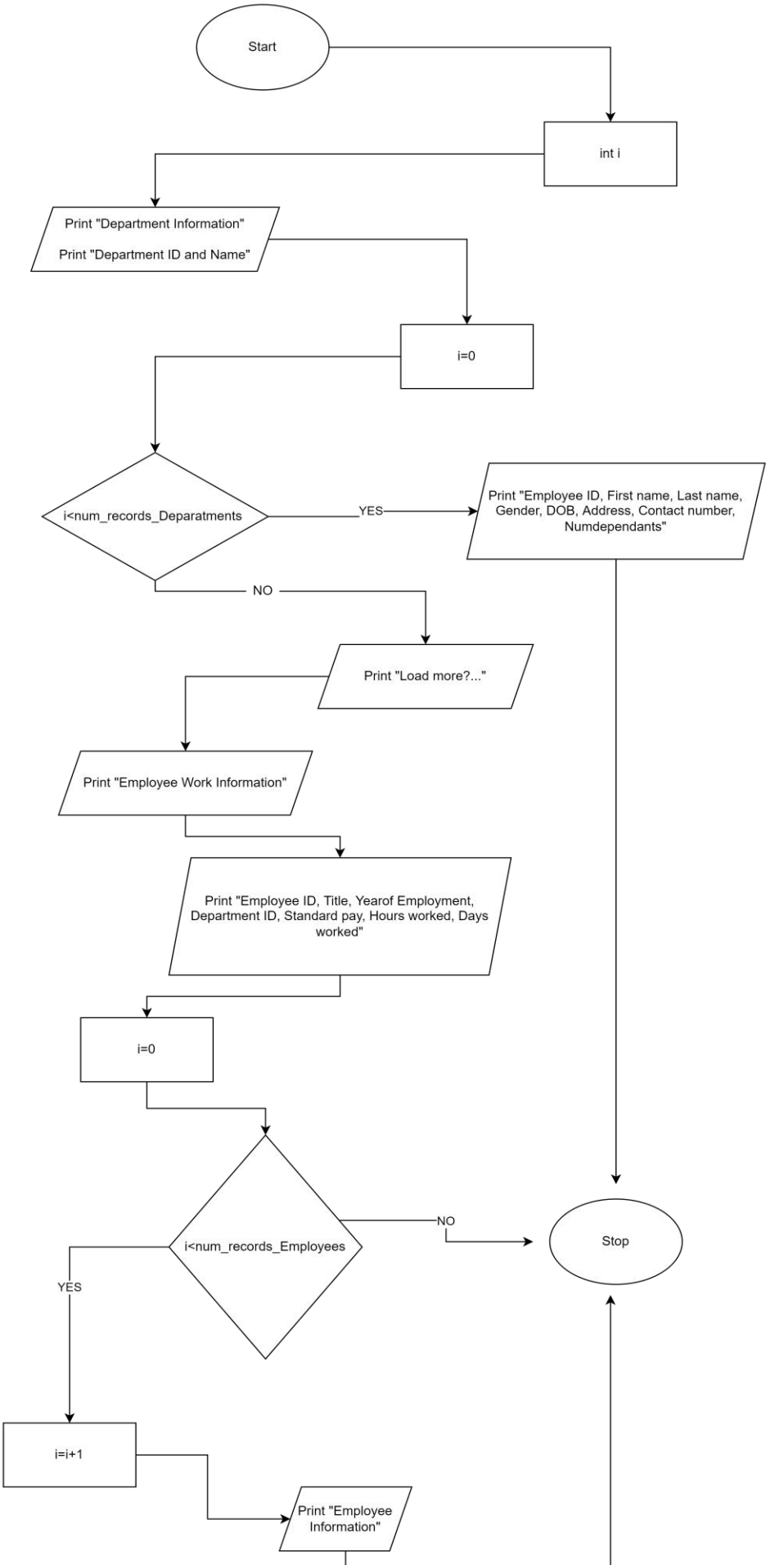
This flowchart tracks the display of the menu to the users of computer program

menu


Start
Print "Welcome Clerk"
Print "Payroll Management System"
Print "Please choose from the following options
View Records
Payroll Register
Pay slip
Search
Sort
Edit
Delete
Add
Exit"
Print "Option:"
read choice
Stop

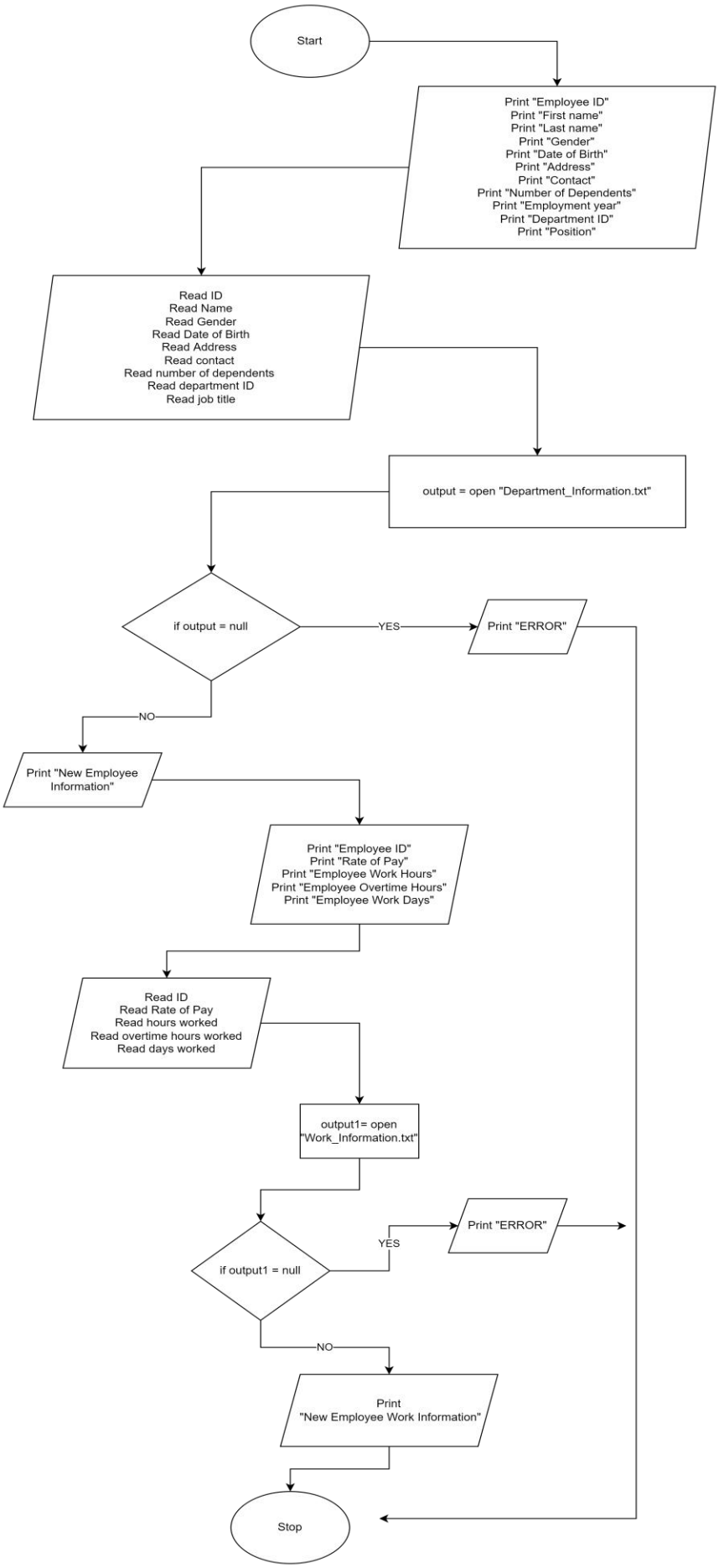
Procedure: View Information

This flowchart tracks the process by which information is displayed to the user of the computer program


Start
int i
Print "Department Information"
Print "Department ID and Name"
i=0
i<num_records_Deparatments
YES
Print "Employee ID, First name, Last name, Gender, DOB, Address, Contact number, Numdependants"
NO
Print "Load more?..."
Print "Employee Work Information"
Print "Employee ID, Title, Yearof Employment, Department ID, Standard pay, Hours worked, Days worked"
i=0
i<num_records_Employees
NO
Stop
YES
i=i+1
Print "Employee Information"

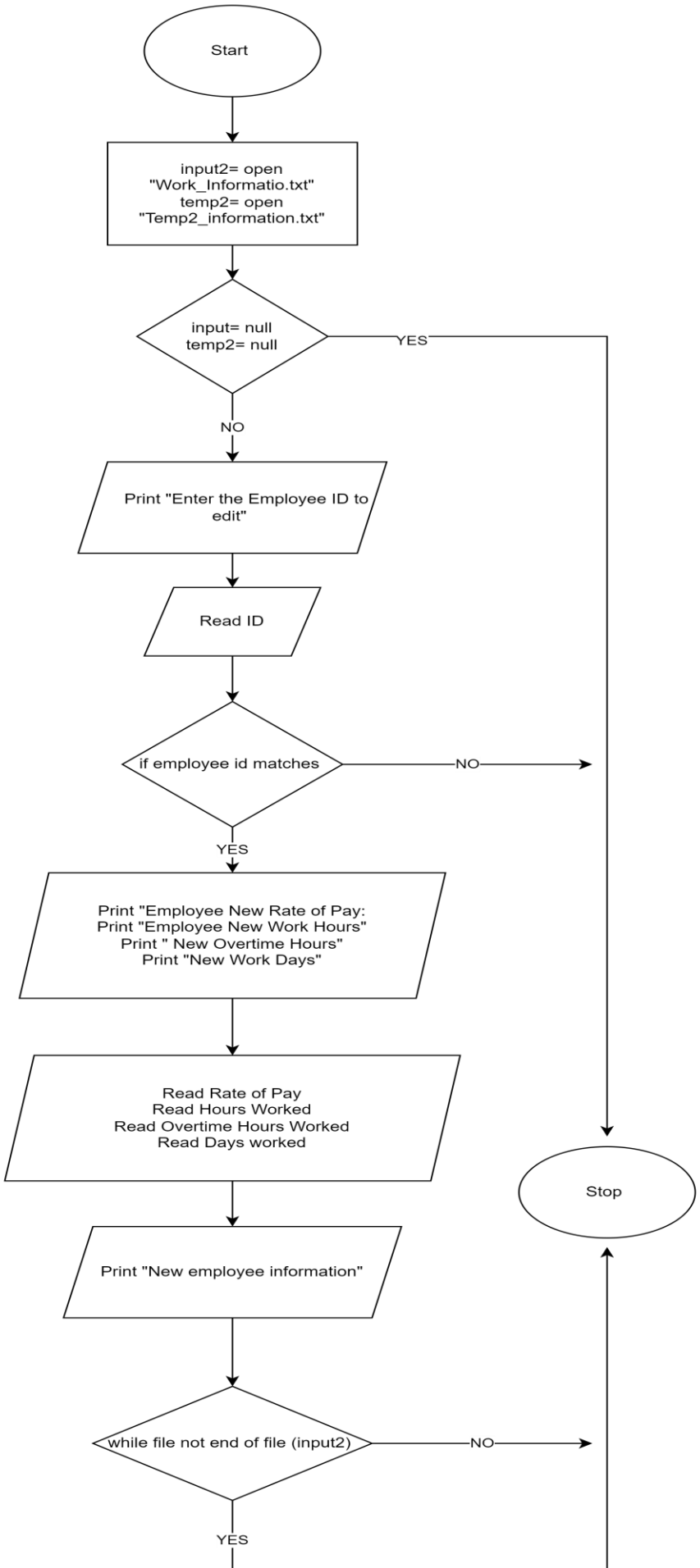
Procedure: Add Record

This flowchart tracks the process by which a recorded is added upon request of the user of the computer program


Start
Print "Employee ID"
Print "First name"
Print "Last name"
Print "Gender"
Print "Date of Birth"
Print "Address"
Print "Contact"
Print "Number of Dependents"
Print "Employment year"
Print "Department ID"
Print "Position"
Read ID
Read Name
Read Gender
Read Date of Birth
Read Address
Read contact
Read number of dependents
Read department ID
Read job title
output = open "Department_Information.txt"
if output = null
YES
Print "ERROR"
NO
Print "New Employee Information"
Print "Employee ID"
Print "Rate of Pay"
Print "Employee Work Hours"
Print "Employee Overtime Hours"
Print "Employee Work Days"
Read ID
Read Rate of Pay
Read hours worked
Read overtime hours worked
Read days worked
output1= open "Work_Information.txt"
if output1 = null
YES
Print "ERROR"
NO
Print "New Employee Work Information"
Stop

Procedure: Edit Information

This flowchart tracks the process by which an employee's information is edited.

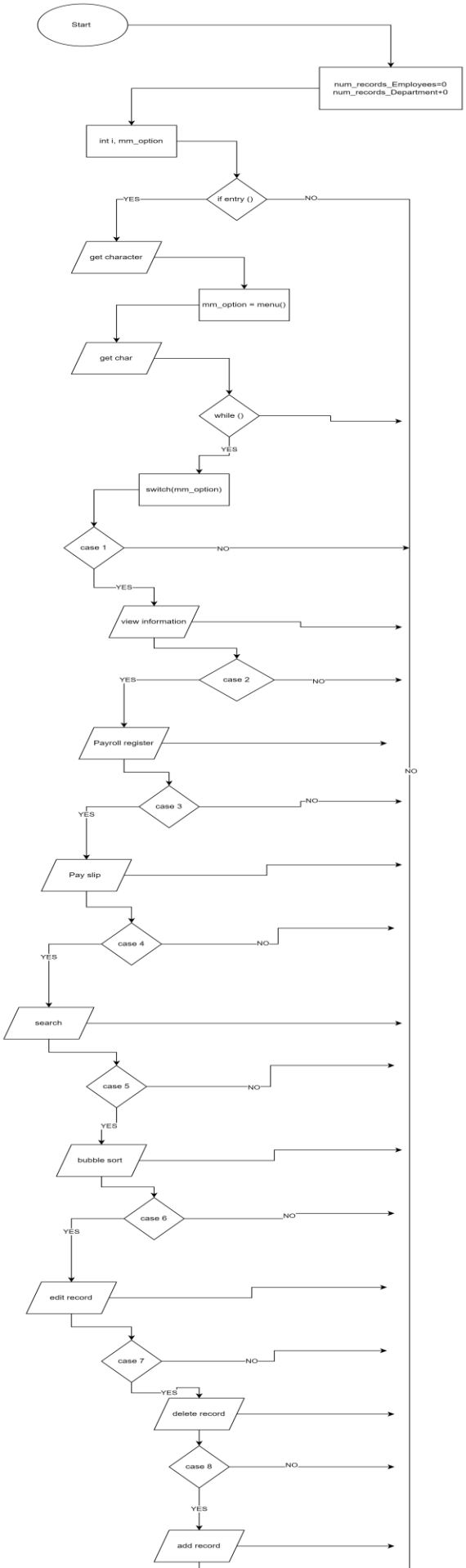

Start
input2= open "Work_Informatio.txt" temp2= open "Temp2_information.txt"
input= null temp2= null
YES
NO
Print "Enter the Employee ID to edit"
Read ID
if employee id matches
NO
YES
Print "Employee New Rate of Pay: Print "Employee New Work Hours" Print " New Overtime Hours" Print "New Work Days"
Read Rate of Pay Read Hours Worked Read Overtime Hours Worked Read Days worked
Print "New employee information"
while file not end of file (input2)
NO
YES
Stop

Procedure: Load Data

This flowchart tracks the process by which data is loaded into the desired text (.txt) files


Start
input= open("Department_Information.txt")
if input= NULL
YES
Print "ERROR"
NO
i=0
i< num_records_dept
YES
Read department ID, department name
num_records_dept ++
close input
i=0
input1= open("Employee_Informationn")
Read Employee ID, First Name, Last Name, Gender, Date of Birth, Address, Contact, Dependants, Year of employment,Department ID, Position
Print Employee ID, First Name, Last Name, Gender, Date of Birth, Address, Contact, Dependants, Year of employment,Department ID, Position
num_records_Employee ++
close input1
1=0
input2= open("Work_Information.txt")
input2= null
YES
Print "Error"
NO
i< number of records in file
NO
YES
i=i+1
Read information in work file
close input2
Stop

Procedure: Main


Start
num_records_Employees=0
num_records_Department+0
int i, mm_option
if entry ()
YES
NO
get character
mm_option = menu()
get char
while ()
YES
switch(mm_option)
case 1
NO
YES
view information
case 2
YES
NO
Payroll register
case 3
NO
YES
Pay slip
case 4
NO
YES
search
case 5
NO
YES
bubble sort
case 6
NO
YES
edit record
case 7
NO
YES
delete record
case 8
NO
YES
add record

Use of Files

FILES

A file structure is a combination of representations for data in files. It is also a collection of operations for accessing the data. It enables applications to read, write, and modify data. File structures may also help to find the data that matches certain criteria. An improvement in file structure has a great role in making applications hundreds of times faster. The main goal of developing file structures is to minimize the number of trips to the disk in order to get desired information. It ideally corresponds to getting what we need in one disk access or getting it with as little disk access as possible.

"Employee_Information.txt" is one of the first files in the system. It will contain information on each employee of differing departments in John and Brothers Ltd. Information such as: name, gender, address, age and any other information that would help the clerk uniquely identify them. This file is needed as it will store personal information about their employees and though some of the information may not directly impact or help the payroll system, it is all information that the company needs. "Temp1_information.txt" works alongside “Employee_Information.txt” closely as it is used when writing to string copy the altered employee’s data such as name or gender, all of which can be found in “Employee_Information.txt”. "Work_Information.txt" is the third file in the system. It contains information that will be used or directly impact the formation of the payroll or an employee’s pay slip. It contains information such as hours worked, pension or the employee’s medical plan. In a similar fashion, "Temp2_information.txt" works closely alongside "Work_Information.txt" as it is used to string copy altered employee’s work data when in write mode.

Our group utilized TXT files because they are used to store notes, step-by-step instructions, manuscripts, and other text-based information. A text file is used to store standard and structured textual data or information that is human readable. Furthermore, plain text documents saved in the TXT format can be created, opened and edited using a wide variety of text editing and word processing programs developed for Linux systems, Microsoft Windows-based computers and Mac platforms. Since the clerk in our scenario would need to constantly update and edit information a TXT file seemed most suitable especially since those files housed no calculations and such

Use of Data Structures

DATA STRUCTS

Structure is a user-defined datatype in C language which allows us to combine data of different types together. More accurately, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. Data structure provides efficiency, reusability and abstraction. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only.

Some simple advantages of structs are as follows:

- Structures gather more than one piece of data about the same subject together in the same place.
- It is helpful when you want to gather the data of similar data types and parameters like first name, last name, etc.
- It is very easy to maintain as we can represent the whole record by using a single name.
- In structure, we can pass complete set of records to any function using a single parameter.
- You can use an array of structure to store more records with similar types.

However, there are some disadvantages such as:

- If the complexity of IT project goes beyond the limit, it becomes hard to manage.
- Change of one data structure in a code necessitates changes at many other places. Therefore, the changes become hard to track.
- Structure is slower because it requires storage space for all the data.

In our program we utilized three structs, one of them being struct department, which contains the department name and ID that will eventually be used by the clerk when she or he wishes to search for something on the payroll system. The main components that make up a struct are the keyword which is first used to declare the struct, the struct tag , struct members/ fields which can be found within the curly brackets and sometimes structure variables.

In the case of struct department:

Keyword: struct

Tag: department

Members/ fields: departmentId and departmentName

Variables: NUM_RECORDS_DEPT

Arrays

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. Instead of declaring individual variables, such as number1, number2, ..., number99, you just declare one array variable number of integer type and use number1[0], number1[1], and ..., number1[99] to represent individual variables. A variable holds a single item of data. There may be a situation where lots of variables are needed to hold similar and related data. In this situation, using an array can simplify a program by storing all related data under one name. This means that a program can be written to search through an array of data much more quickly than having to write a new line of code for every variable. This reduces the complexity and length of the program which makes it easier to find and debug errors.

Struct Department and Struct Employee were created to hold information on the department and employees, including arrays such as char departmentID[25] and char employeeID[25] which were defined in the program to hold the ID of the department or employees.

Demonstration of Structured Programming Concepts

Structured programming languages allow the programmer to divide the whole program into modules or functions. Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Advantages of Structured Programming

1. It is user friendly and easy to understand.
2. Similar to English vocabulary of words and symbols.
3. It is easier to learn.
4. They require less time to write.
5. They are easier to maintain.
6. These are mainly problem oriented rather than machine based.
7. Program written in a higher-level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator.
8. It is independent of machine on which it is used i.e. programs developed in high level languages can be run on any computer.

Our program follows a top down approach. A "top-down" approach is where an executive decision maker or other top person makes the decisions of how something should be done. This approach is disseminated under their authority to lower levels in the hierarchy, who are, to a greater or lesser extent, bound by them. One of the most important advantages of top-down planning is that targets can be set quickly for the whole business. There is no time wasted in analyzing each department's performance, and management can rapidly implement the company's goals. The program has one main function with eleven (11) sub functions such as edit, delete, search and bubble sort .

The edit procedure should allow the clerk to edit the files as she or he pleases to add, change or delete information in the files of the employee. The delete procedure will allow you to delete a file in the TXT documents that are included in the program. The search procedure allows the clerk to search for an Employee using the department ID, employeeID and producing the employee information that the clerk was searching for. On the other hand, bubble sort is used to sort the information in a specific order.

These modules divide the functionality of the program such that each one contains all that is necessary to execute only one aspect of the entire functionality. In this program, separate procedures were developed for the purpose maintainability and decreased development time.

Coding of Program

Code

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define EMPLOYEE_INFO_FILE "Employee_Information.txt"

#define TMP1_FILE "Temp1_information.txt"

#define EMPLOYEE_WORK_FILE "Work_Information.txt"

#define TMP2_FILE "Temp2_information.txt"

#define NUM_DEPTS 10

#define MAX_CHARS 50

#define EMPLOYEE_CAPACITY 60

#define NUM_RECORDS_IN_FILE 50

#define dependantsAllowance_rate 0.1

#define personalAllowance_rate 0.5

#define nationalInsurance 0.1

#define medicalPlan 0.05

#define pension 0.02

#define incomeTax 0.1
```

```
#define union_rate 0.03

#define NUM_RECORDS_DEPT 8



struct Department

{

    char departmentID[25];

    char departmentName[25];

};


struct Department department[NUM_DEPTS];


struct Employee

{

    char employeeID[MAX_CHARS];

    char firstName[MAX_CHARS];

    char lastName[MAX_CHARS];

    char gender[MAX_CHARS];

    char dob[MAX_CHARS];

    char address[MAX_CHARS];

    char contactNumber[MAX_CHARS];

    int numDependants;

    int startYear;
```

```
    char departmentID[MAX_CHARS];
    char jobTitle[MAX_CHARS];
};

struct Employee employees[EMPLOYEE_CAPACITY];

struct MonthlyWageData
{
    char employeeID[25];
    float RateOfPay;
    int hoursWorked;
    int overtimeHoursWorked;
    int daysWorked;
};

struct MonthlyWageData mwages[EMPLOYEE_CAPACITY];

//Globals
int num_records_Employees;
int num_records_Department;

/*
 * Function:  approx_pi
```

```
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
int entry()
{
   system("cls");
   char code[25];
   printf("Please enter password:");
   scanf("%s",&code);
   const char password[8]="abc";

   if (strcmp(code,password)==0 )
   {
      printf("\nAccess Granted.....\n");
      return 1;
   }
```

```
    else
    {
        printf("\nAccess Denied.....\n");
        return 0;
    }
}


/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void exit_message()
{
    system ("cls");
    printf("THANK YOU!\n\n");
```

```
    system("pause");
    exit(1);

}



/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void paySlip()
{
    system("pause");
    system("cls");
    char id_employee[25];
```

```
printf("\nPlease Enter the employess's ID: ");
scanf("%s",id_employee);
int i;
int line_num;
int find_result;

float grossPay, dependantsAllowance, personalAllowance, taxFree_income;
float taxable_income, national_insuranceAMT, medicalPlan_AMT;
float taxable_remainder, pension_AMT, incomeTax_AMT, union_AMT;
float total_deductions, overTime_AMT, netPay, wageAMT,overtime_AMT;

FILE *input1= fopen("Employee_Information.txt","r");

for(i=0; i<num_records_Employees; i++)
{
   if((strcmp(id_employee, employees[i].employeeID)) == 0)
   {

      wageAMT=(mwages[i].hoursWorked * mwages[i].RateOfPay) *
mwages[i].daysWorked;
      overTime_AMT= mwages[i].overtimeHoursWorked * (1.5 * mwages[i].RateOfPay);
      dependantsAllowance= employees[i].numDependants * dependantsAllowance_rate;
      personalAllowance= wageAMT * personalAllowance_rate;
```

```
grossPay=wageAMT+overTime_AMT;

taxFree_income= dependantsAllowance + personalAllowance;


taxable_income= grossPay - taxFree_income;


//The first $200 of an employee's taxable income is subjected to national insurance rate

national_insuranceAMT = 200 * nationalInsurance;


medicalPlan_AMT= (taxable_income - national_insuranceAMT) * medicalPlan;


//The remainder of the Taxable_income is subjected to pension rate, income tax rate and union rate

taxable_remainder= taxable_income -(national_insuranceAMT + medicalPlan_AMT);

pension_AMT= taxable_remainder * pension;

incomeTax_AMT= (taxable_remainder - pension_AMT) * incomeTax;

union_AMT= (taxable_remainder - (pension_AMT + incomeTax_AMT)) * union_rate;


total_deductions= national_insuranceAMT + medicalPlan_AMT + pension_AMT + incomeTax_AMT + union_AMT;


netPay= grossPay - total_deductions;


printf("\n-----------------------------------------------------------------------");
```

```
        printf("\nEmployee Information\n");
        printf("\nName of Employee: %s %s",employees[i].firstName,employees[i].lastName);
        printf("\nAddress: %s",employees[i].address);
        printf("\nContact: %s",employees[i].contactNumber);
        printf("\nDepartment ID: %s",employees[i].departmentID);
        printf("\nPosition: %s",employees[i].jobTitle);


        printf("\n\n\t\tHours\t    Days    Rate\tCurrent Amount");
        printf("\nEarnings");
        printf("\nStandard pay\t %-10d   %-d     %-10.2f  $%-
10.2f",mwages[i].hoursWorked,mwages[i].daysWorked,mwages[i].RateOfPay, wageAMT);
        printf("\nOvertime pay\t %-10d         %-0.2f       $%-
10.2f",mwages[i].overtimeHoursWorked,mwages[i].RateOfPay*1.5, overTime_AMT);
        printf("\n---------------------------------------------------------------------------");
        printf("\n\n\t\t\tGross Pay\t\t $%.2f",grossPay);
        printf("\n---------------------------------------------------------------------------");
        printf("\n\n\t\t\t\t\t\tCurrent Amount");
        printf("\n\nDeduction");
        printf("\nNational Insurance\t\t\t\t $%-1.2f",national_insuranceAMT);
        printf("\nMedial Plan\t\t\t\t\t $%-1.2f",medicalPlan_AMT);
        printf("\nPension\t\t\t\t\t\t $%-1.2f",pension_AMT);
        printf("\nIncome Tax\t\t\t\t\t $%-5.2f",incomeTax_AMT);
        printf("\nUnion\t\t\t\t\t\t $%-10.f",union_AMT);
```

```
        printf("\n--------------------------------------------------------------------------");
        printf("\n\n\t\tTotal Deductions \t\t $%.2f",total_deductions);
        printf("\n--------------------------------------------------------------------------");
        printf("\n\n\t\tNet Pay\t\t\t\t $%.2f",netPay);
        printf("\n--------------------------------------------------------------------------");


    }


  }
}


/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
```

```
void payroll_register()

{

   system("cls");

   int i;

   float grossPay[num_records_Employees], dependantsAllowance[num_records_Employees],
personalAllowance[num_records_Employees], taxFree_income[num_records_Employees];

   float taxable_income[num_records_Employees],
national_insuranceAMT[num_records_Employees],
medicalPlan_AMT[num_records_Employees];

   float taxable_remainder[num_records_Employees], pension_AMT[num_records_Employees],
incomeTax_AMT[num_records_Employees], union_AMT[num_records_Employees];

   float total_deductions[num_records_Employees], overTime_AMT[num_records_Employees],
netPay[num_records_Employees],
wageAMT[num_records_Employees],overtime_AMT[num_records_Employees];


   for(i=0; i<num_records_Employees; i++)

   {

      wageAMT[i]=(mwages[i].hoursWorked * mwages[i].RateOfPay) *
mwages[i].daysWorked; //The employee wage was calculated by multiplying their hours worked
daily by hourly rate and the total number of days worked in the given month

      overTime_AMT[i]= mwages[i].overtimeHoursWorked * (1.5 * mwages[i].RateOfPay); //
the employee overtime wage was calculated by multiply their overtime hours by time and a half
of their employee hourly rate

      dependantsAllowance[i]= employees[i].numDependants * dependantsAllowance_rate; //
the dependants allowance is part of the employee taxfree income by mutiplying the number of
household dependants by the rate per
```

```
personalAllowance[i]= wageAMT[i] * personalAllowance_rate; // The personal allowance is part of the employee taxfree income by multiplying their wage by the personal aloowance rate

grossPay[i]=wageAMT[i]+overTime_AMT[i]; // the gross par is the sum of the employee wage and overtime wage

taxFree_income[i]= dependantsAllowance[i] + personalAllowance[i]; // the total taxfree income was the sum of the dependants allowance and personal allowance


taxable_income[i]= grossPay[i] - taxFree_income[i]; // the taxable income was the differnce of the employee gross pay and the total taxfree income


//The first $200 of an employee's taxable income is subjected to national insurance rate

// National insurance is multiplying the national insurance rate by $200

national_insuranceAMT[i] = 200 * nationalInsurance;


// Medcial plan is obtained by first subtracting the taxable_income by the national insurance amount then multiplying the differnce by the medical plan rate

medicalPlan_AMT[i] = (taxable_income[i] - national_insuranceAMT[i]) * medicalPlan;


// Taxable remainder is the sum of the medical plan amount and the  national insurance amount subtacted from the taxable income

taxable_remainder[i]= taxable_income[i] -(national_insuranceAMT[i] + medicalPlan_AMT[i]);


//The remainder of the Taxable_income is subjected to pension rate, income tax rate and union rate
```

```
        // Pension Amount is the taxable reminder multiplied by the pension rate
        pension_AMT[i]= taxable_remainder[i] * pension;

        //Income Tax Amount is the first subtracting the taxable remainder by the pension Amount then multiplying the difference by the income tax rate
        incomeTax_AMT[i]= (taxable_remainder[i] - pension_AMT[i]) * incomeTax;

        // Union Amount is first subtracting the taxable income by the sum of the pension Amount and the income tax then multiplying the differnce by the union rate
        union_AMT[i]= (taxable_remainder[i] - (pension_AMT[i] + incomeTax_AMT[i])) * union_rate;

        // The Totla deductions is the sum of the national in surance amount, medical plan amount, pension amount, income tax amount and the union amount
        total_deductions[i]= national_insuranceAMT[i] + medicalPlan_AMT[i] + pension_AMT[i] + incomeTax_AMT[i] + union_AMT[i];

        // Net Pay is the gross pay subtracted by the total deductions
        netPay[i]= grossPay[i] - total_deductions[i];
    }
    printf("\t\t\t\t__________ Payroll Register for Monthly Paid Staff_____________\n");
    printf("\n\t\t\t\t   Earnings\t\t\t    Deductions");
    printf("\nEmp ID\t\tEmp Name\t  Total Wage  OT Wage   G Pay\t    Ins.    M. Plan    Pension    I. Tax     Union      Total      Net Pay\n");
```

```
   for(i=0; i<num_records_Employees; i++)
   {
      printf("\n%-8s    %-10s %-9s   $%-10.2f $%-9.2f  $%-8.2f  $%-7.2f $%-9.2f $%-10.2f $%-10.2f $%-10.2f $%-10.2f $%-20.2f",employees[i].employeeID,
            employees[i].firstName,
            employees[i].lastName,
            wageAMT[i],
            overTime_AMT[i],
            grossPay[i],
            national_insuranceAMT[i],
            medicalPlan_AMT[i],
            pension_AMT[i],
            incomeTax_AMT[i],
            union_AMT[i],
            total_deductions[i],
            netPay[i]);
   }
}

/*
 * Function:  approx_pi
 * --------------------
```

```
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void search()
{
   system("pause");
   system("cls");
   int i;
   char id_employee[MAX_CHARS];

   FILE *input1= fopen("Employee_Information.txt","r");
   FILE *input2= fopen("Work_Information.txt","r");

   printf("Enter the Employee ID to Search : ");
   scanf("%s",&id_employee);
   system("cls");
```

```
printf("Employee_ID\tEmp_Name\t Emp_Gender DoB Address Contact Nos_Dependants Emp_year Department_ID Position");


for(i=0; i<num_records_Employees; i++)
{
   if(strcmp(id_employee,employees[i].employeeID)==0)
   {
      system("cls");
      printf("Employee ID: %s \nEmployee Name: %s %s \nEmployee Gender: %s \nDate of Birth: %s \nAddress: %s \nContact: %s \nNumber of Dependants: %d \nEmployment Year: %d \nDepartment ID: %s \nPosition: %s \nRate of Pay: %.2f \nHours Worked: %d \nDays Worked: %d \nOvertime Hours Worked: %d",
            employees[i].employeeID,
            employees[i].firstName,
            employees[i].lastName,
            employees[i].gender,
            employees[i].dob,
            employees[i].address,
            employees[i].contactNumber,
            employees[i].numDependants,
            employees[i].startYear,
            employees[i].departmentID,
            employees[i].jobTitle,
            mwages[i].RateOfPay,
```

```
                mwages[i].hoursWorked,
                mwages[i].daysWorked,
                mwages[i].overtimeHoursWorked);
        }
    }
    fclose(input1);
    fclose(input2);
}

/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void bubble_sort()
{
```

```
system("cls");
int n,k,i;
char temp1[MAX_CHARS];

struct Employee eTemp;

for (i=0; i<=num_records_Employees; i++)
{
   for(k=0; k<=(num_records_Employees-1-i); k++)
   {
      if(strcmp(employees[k].jobTitle,employees[k+1].jobTitle)>0)
      {
         eTemp = employees[k];
         employees[k] = employees[k+1];
         employees[k+1] = eTemp;

         //strcpy(temp1,employees[k].jobTitle);
         //strcpy(employees[k].jobTitle,employees[k+1].jobTitle);
         //strcpy(employees[k+1].jobTitle,temp1);
      }
   }
}
```

```
    printf("___________ Sorted Employee Record_____________\n\n");
    printf("\nEmployee ID\t Employee Position    \t Employee Name");
    for(i=0; i<num_records_Employees; i++)
    {
        printf("\n%-8s        %-23s  %-8s %s",
                employees[i].employeeID,
                employees[i].jobTitle,
                employees[i].firstName,
                employees[i].lastName);
    }
}

/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
```

```
 */
void delete_record()
{
   system("cls");
   char id_employee[25];
   printf("Enter the Employee ID you wish to delete");
   scanf("%s", &id_employee);

   char tempID1[MAX_CHARS];
   int c, t;
   char tempID2[MAX_CHARS];
   char buffer[num_records_Employees];

   FILE *input1= fopen("Employee_Information.txt","r");
   FILE * temp1= fopen("Temp1_information.txt","w");
   FILE *input2= fopen("Work_Information.txt","r");
   FILE * temp2= fopen("Temp2_information.txt","w");

   if (input1 == NULL || temp1 == NULL || input2 == NULL || temp2 == NULL)
   {
      exit(1);
   }
   // Copy the data from the original to the temp file
```

```
while (!feof(input1))
{
   if ( fscanf(input1,"%d\n",&tempID1) < 0) break;
   t = 0;  //reset our count var
   do
   {
      c = fgetc (input1);
      if (c == EOF) break;
      if  (c != '\n' ) buffer[t++] = c;
   }
   while (c != '\n');
   buffer[t] = '\0';

   //Skip the line to omit.
   if (tempID1 != id_employee)
      fprintf(temp1,"%d %s\n",tempID1, buffer);
}

while (!feof(input2))
{
   if ( fscanf(input2,"%d\n",&tempID2) < 0) break;
   t = 0;  //reset our count var
   do
```

```
        {
            c = fgetc (input2);
            if (c == EOF) break;
            if  (c != '\n' ) buffer[t++] = c;
        }
        while (c != '\n');
        buffer[t] = '\0';

        //Skip the line to omit.
        if (tempID2 != id_employee)
            fprintf(temp2,"%d %s\n",tempID2, buffer);
    }
    fclose(input1);
    fclose(temp1);
    fclose(input2);
    fclose(temp2);
    remove(EMPLOYEE_INFO_FILE);                    // remove the original file
    rename(TMP1_FILE, EMPLOYEE_INFO_FILE);
    remove(EMPLOYEE_WORK_FILE);
    rename(TMP2_FILE, EMPLOYEE_WORK_FILE);       // rename the temporary file to
original name
}
```

```
/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
int menu()
{
    int choice;


printf("\n\n/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\
*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\");
    printf("\n");
    printf("\n\t\t\t\t\t  WELCOME CLERK");
    printf("\n\t\t\t             John's and Brothers LTD");
    printf("\n\t\t\t\t\t\t\t\t\t\t\t\t\t");
```

```
   printf("\n\t\t\t\t     PAYROLL MANAGEMENT SYSTEM ");
   printf("\n");
   printf("\n");

printf("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
   printf("\n\t\t\t\tPlease choose from the following menu options:");
   printf("\n\t\t\t\t\t\t\t\t");
   printf("\n\t\t\t\t 1 - View Records");              /* < func. req # */
   printf("\n\t\t\t\t 2 - Payroll Register");          /* < func. req # */
   printf("\n\t\t\t\t 3 - Payslip of an Employee");    /* < func. req # */
   printf("\n\t\t\t\t 4 - Search Records");            /* < func. req # */
   printf("\n\t\t\t\t 5 - Sorted Employee Record");    /* < func. req # */
   printf("\n\t\t\t\t 6 - Edit Records");              /* < func. req # */
   printf("\n\t\t\t\t 7 - Delete Records");            /* < func. req # */
   printf("\n\t\t\t\t 8 - Add Records");               /* < func. req # */
   printf("\n\t\t\t\t 9 - Exit System");
   printf("\n");

printf("\n\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\
\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\*/*\\");


   printf("\n\nOption: ");
   scanf("%d",&choice);
```

```
    return choice;
}


/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void view_information()
{
    int i;


    printf("\t\t__________Deparment Information_____________\n");
    printf("\nDeparment ID\t\tDepartment Name\n");
    for(i=0;i<num_records_Department; i++)
    {
```

```
        printf("\n%-10s            %-
10s",department[i].departmentID,department[i].departmentName);
    }

    printf("\nLoad more?...");
    system("pause");

    printf("\n\n\n\t\t__________ Employee Personal Information_____________\n");

    printf("\nEmployee ID \tEmployee Name\t   Gender    Date Of Birth   Address \t\t Contact \t#
of Dpdts\n");

    for (i=0; i<num_records_Employees; i++)
    {
       printf("\n%-10s    %-10s %-9s %7s     %-10s    %-15s %-14s %13d",
            employees[i].employeeID,
            employees[i].firstName,
            employees[i].lastName,
            employees[i].gender,
            employees[i].dob,
            employees[i].address,
            employees[i].contactNumber,
            employees[i].numDependants);
```

```
    }
    printf("\nLoad more?...");
    system("pause");
    printf("\n\n\t\t\t\t_____ Employee Work Information_____\n");
    printf("\n\nEmp ID\t    Title\t\tYear of Emp\t Dept ID\t Standard Pay\tHours Worked\t
OT\tDays Worked\n\n");
    for (i=0; i<num_records_Employees; i++)
    {
       printf("%-8s  %-23s  %-8d  %12s  %14.2f    %10d  %10d   %10d\n",
            mwages[i].employeeID,
            employees[i].jobTitle,
            employees[i].startYear,
            employees[i].departmentID,
            mwages[i].RateOfPay,
            mwages[i].hoursWorked,
            mwages[i].overtimeHoursWorked,
            mwages[i].daysWorked);
    }
}
/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
```

```
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void add_record()
{
    system("cls");
    int i;
    printf("\t\t\t\t__________ Adding Employee Work Information_____________\n");
    printf("\nEmployee ID:  ");
    scanf("%s",&employees[i].employeeID);
    printf("\nEmployee first name: ");
    scanf("%s",&employees[i].firstName);
    printf("\nEmployee last name:  ");
    scanf("%s",&employees[i].lastName);
    printf("\nEmployee Gender: ");
    scanf("%s",&employees[i].gender);
    printf("\nEmployee date of birth:  ");
    scanf("%s",&employees[i].dob);
```

```
printf("\nEmployee Adress:  ");
scanf("%s",&employees[i].address);
printf("\nEmployee contact:  ");
scanf("%s",employees[i].contactNumber);
printf("/nEmployee Number of Dependants:  ");
scanf("%d",employees[i].numDependants);
printf("\nEmployee Employment Year:  ");
scanf("%d",employees[i].startYear);
printf("\nDepartment ID:  ");
scanf("%s",employees[i].departmentID);
printf("\nEmployee Position:  ");
scanf("%s",employees[i].jobTitle);

FILE *output= fopen("Department_Information.txt","w");
if(output == NULL)
{
   printf("Error");
   exit(1);
}
fprintf(output,"%s %s %s %s %s %s %s %d %d %s %s\n",
       employees[i].employeeID,
       employees[i].firstName,
       employees[i].lastName,
```

```
        employees[i].gender,
        employees[i].dob,
        employees[i].address,
        employees[i].contactNumber,
        employees[i].numDependants,
        employees[i].startYear,
        employees[i].departmentID,
        employees[i].jobTitle);

printf("\n\nEmployee ID:  ");
scanf("%s",&mwages[i].employeeID);
printf("\nEmployee Rate of Pay:  ");
scanf("%f",&mwages[i].RateOfPay);
printf("\nEmployee Work Hours:  ");
scanf("%d",&mwages[i].hoursWorked);
printf("\nEmployee Overtime Hours:  ");
scanf("%d",&mwages[i].overtimeHoursWorked);
printf("\nEmployee Work Days:  ");
scanf("%d",&mwages[i].daysWorked);

FILE *output1= fopen("Work_Information.txt","w");
if(output1 == NULL)
{
```

```
        printf("Error");
        exit(1);
    }
    fprintf(output1,"%s %f %d %d %d\n",
            mwages[i].employeeID,
            mwages[i].RateOfPay,
            mwages[i].hoursWorked,
            mwages[i].overtimeHoursWorked,
            mwages[i].daysWorked);
}
/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
void edit_record()
```

```
{
    char id_employee[MAX_CHARS];
    int i;
    system("cls");

    FILE *input2= fopen("Work_Information.txt","r");
    FILE * temp2= fopen("Temp2_information.txt","w");

    if (input2 == NULL || temp2 == NULL)
    {
        exit(1);
    }
    printf("Enter Employee ID to Edit : ");
    fflush(stdin);
    scanf("%s",id_employee);
    printf("Here\n");
    system("pause");

    for (i = 0; i < num_records_Employees; i++) {
    if((strcmp(id_employee, mwages[i].employeeID)) == 0)
    {
        printf("\nEmployee New Rate of Pay:  ");
        scanf("%f",&mwages[i].RateOfPay);
```

```
    printf("\nEmployee New Work Hours:  ");
    scanf("%d",&mwages[i].hoursWorked);
    printf("\nEmployee New Overtime Hours:  ");
    scanf("%d",&mwages[i].overtimeHoursWorked);
    printf("\nEmployee New Work Days:  ");
    scanf("%d",&mwages[i].daysWorked);

    fprintf(temp2,"%s %f %d %d %d\n",
            &mwages[i].employeeID,
            &mwages[i].RateOfPay,
            &mwages[i].hoursWorked,
            &mwages[i].overtimeHoursWorked,
            &mwages[i].daysWorked);
}
}

printf("Before while \n");
system("pause");
while(!(feof(input2)));
fclose(input2);
fclose(temp2);
remove(EMPLOYEE_WORK_FILE);
rename(TMP2_FILE, EMPLOYEE_WORK_FILE);
```

```
    system ("cls");
}


/*
 * Function:  approx_pi
 * --------------------
 * computes an approximation of pi using:
 *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
 *
 *  n: number of terms in the series to sum
 *
 *  returns: the approximate value of pi obtained by suming the first n terms
 *           in the above series
 *           returns zero on error (if n is non-positive)
 */
int loadDATA()
{
    int i;
    FILE *input= fopen("Department_Information.txt","r");
    if(input == NULL)
    {
        printf("Error");
        exit(1);
```

```
}
for (i=0; i<NUM_RECORDS_DEPT; i++)
{
   fscanf(input, "%s %s",department[i].departmentID,
        department[i].departmentName );
   num_records_Department++;
}
fclose(input);

i=0;
FILE *input1= fopen("Employee_Information.txt","r");
if(input1 == NULL)
{
   printf("Error");
   exit(1);
}
for (i=0; i<NUM_RECORDS_IN_FILE; i++)
{
   fscanf(input1,"%s %s %s %s %s %s %s %d %d %s %s",
        &employees[i].employeeID,
        &employees[i].firstName,
        &employees[i].lastName,
        &employees[i].gender,
```

```
            &employees[i].dob,
            &employees[i].address,
            &employees[i].contactNumber,
            &employees[i].numDependants,
            &employees[i].startYear,
            &employees[i].departmentID,
            &employees[i].jobTitle);

    /* printf("%s %s %s %s %s %s %d %d %s %s\n",
            employees[i].employeeID,
            employees[i].firstName,
            employees[i].lastName,
            employees[i].gender,
            employees[i].dob,
            employees[i].address,
            employees[i].contactNumber,
            employees[i].numDependants,
            employees[i].startYear,
            employees[i].departmentID,
            employees[i].jobTitle);*/

    num_records_Employees ++;
}
```

```
    fclose(input1);
    i=0;
    FILE *input2= fopen("Work_Information.txt","r");
    if(input2 == NULL)
    {
       printf("Error");
       exit(1);
    }
    for (i=0; i<NUM_RECORDS_IN_FILE; i++)
    {
       fscanf(input2,"%s %f %d %d %d\n",
            &mwages[i].employeeID,
            &mwages[i].RateOfPay,
            &mwages[i].hoursWorked,
            &mwages[i].overtimeHoursWorked,
            &mwages[i].daysWorked);
    }
    fclose (input2);
}

int main ()
{
    num_records_Employees = 0;
```

```
num_records_Department =0;
loadDATA();
int mm_option;
int i;
if ( entry()  )
{
   getchar();


   system ("cls");
   mm_option = menu();
   getchar();
   while (1)
   {
      switch (mm_option)
      {
      case 1:
         view_information();          /* < func. req # */
         break;
      case 2:
         payroll_register();          /* < func. req # */
         getchar();
         break;
      case 3:
```

```
    paySlip();                  /* < func. req # */
    break;
case 4:
    search();                   /* < func. req # */
    break;
case 5:
    bubble_sort();              /* < func. req # */
    break;
case 6:
    edit_record();              /* < func. req # */
    break;
case 7:
    delete_record();            /* < func. req # */
    break;
case 8:
    add_record();               /* < func. req # */
    break;
case 9:
    exit_message();
    break;
default:
    exit(1);
}//end switch
```

```
            mm_option = menu();
        }//end while
    }//end if @entry
    return 0;
}
```

Testing and Presentation

Test Plan

For this test, the input files used are as follows:

*the highlighted

Department_Information.txt:

FI03456 Financial

SA03859 Sales

MA03184 Marketing

HR03839 Human_Resource_Management

OP03768 Operations

PR03904 Procurement

CS03052 Customer_Service

AC03045 Accounts

Employee_Information.txt:

AB28102 Penelope Santiago Female 04/21/1993 Arouca 1-868-781-0695 0 2021 FI03456 Internal_Auditor
AC04855 Mason Parker Male 02/28/1966 Tunapuna 1-868-650-2751 3 2017 MA03184 Publicity_Asst
AF89345 Paisley Alexander Female 09/10/1999 San_Juan 1-868-636-0209 0 2022 CS03052 Call_Center_Agent
AB87852 Rosie Peyton Female 11/21/1981 Arouca 1-868-671-5178 2 2018 MA03184 Publicity_Asst
AT59893 Megan Piper Female 10/30/1990 Arima 1-868-634-4012 1 2019 PR03904 Assistant_Buyer
AD98455 Jessica Phoenix Female 12/30/1983 Tunapuna 1-868-627-3627 3 2018 FI03456 Fin_Accountant
AY39856 Fiona Patrick Female 06/24/1980 Arima 1-868-652-8282 2 2008 PR03904 Assistant_Buyer
AN08456 Stella Peter Female 03/12/1982 Couva 1-868-672-4508 0 2008 CS03052 Call_Center_Agent
AR40856 Katie Matthew Female 01/25/1978 Port_Of_Spain 1-868-628-2867 0 2000 HR03839 HR_Intern
AR90998 Kyle Hudson Female 08/12/1979 Arima 1-868-660-8874 0 1997 PR03904 Contract_Manager
AJ08735 Kai Francisco Male 2/14/1967 Diego_Martin 1-868-653-0963 1 2001 PR03904 Category_Buyer
AQ98045 Kayla Braxton Female 3/25/1969 Port_Of_Spain 1-868-631-0020 2 1997 HR03839 HR_Intern
AT07875 Keira Hector Female 7/21/1969 Arouca 1-868-645-5573 4 1998 CS03052 Call_Center_Agent
AB00865 Ava Kian Female 7/21/1969 Chaguanas 1-868-660-2905 4 1995 PR03904 Category_Buyer
AW79644 Kaiden Fernando Male 8/20/1971 Port_Of_Spain 1-868-639-8851 3 2004 FI03456 Internal_Auditor
AG08801 Kayden Benjamin Male 11/18/1972 Arouca 1-868-639-8271 0 2002 SA03859 Retail_Sales_Clerk
AV07865 Jack Samuel Male 5/11/1974 Chaguanas 1-868-622-8410 3 2005 MA03184 Comm_Manager
AL07866 Sebastian Braxton Male 7/9/1975 Morvant 1-868-627-9649 2 2000 FI03456 Fin_Accountant
AU76542 Mia Scarlett Female 9/1/1977 Diego_Martin 1-868-629-1094 0 1999 CS03052 Client_Relations_Assoc
AL07542 Freya Easton Female 5/11/1978 Couva 1-868-652-9463 1 1999 SA03859 Retail_Sales_Clerk
AM05326 Faith Hervey Female 12/15/1979 Sangre_Grande 1-868-652-9114 2 2002 PR03904 Buyer
AX02587 Mack Hunter Male 2/4/1980 Arima 1-868-653-6259 4 2005 PR03904 Buyer
AE74184 Ruby Harrison Female 5/9/1980 San_Fernando 1-868-649-2259 1 2008 AC03045 Accounting_Su
AB79420 Hayden James Male 10/6/1982 Morvant 1-868-639-6389 0 2006 OP03768 Operations_Tech
AV84211 Anthoy James Male 12/24/1982 Couva 1-868-657-6034 0 1999 OP03768 Operations_Tech

AL08731 Jamie Bennett Male 9/30/1983 Arima 1-868-639-5191 4 1998 HR03839 HR_Specialist
AL94522 Avery Riley Female 1/23/1984 Diego_Martin 1-868-660-7795 3 1998 CS03052 Customer_Service_Rep
AQ02642 Rebecca Mason Female 9/19/1985 Morvant 1-868-657-2966 1 2002 FI03456 Credit_Controller
AH00128 Millie Barrett Female 12/3/1986 San_Fernando 1-868-671-1350 2 2003 AC03045 Accountant
AV08345 Sofia Ezra Female 1/26/1987 Arouca 1-868-645-4965 0 2014 SA03859 Sales_Engineer
AS32097 Finn Hendrix Male 3/30/1991 Couva 1-868-639-3521 1 2022 OP03768 Operations_Asst
AA20173 Elias Ryan Male 5/5/1991 San_Fernando 1-868-631-0020 0 2020 HR03839 Recruiter
AL30236 Asher Joshua Male 10/28/1991 Diego_Martin 1-868-639-2619 1 2021 PR03904 Purchasing_Manager
AV03281 Reuben Jacobs Male 7/21/1993 Port_Of_Spain 1-868-645-4965 0 2022 PR03904 Procurement_Officer
AZ54087 Maisie Holden Female 1/5/1995 Sangre_Grande 1-868-658-4000 0 2016 AC03045 Staff_Accountant
AJ30857 Eli Finley Female 3/27/1996 Arima 1-868-652-4412 2 2021 MA03184 Comm_Manager
AR24611 Amelia Felix Female 12/29/1968 Diego_Martin 1-868-639-3175 4 1995 AC03045 Accountant
AY39750 Even Henry Male 5/1/1971 Chaguanas 1-868-639-3175 2 1996 OP03768 Operations_Director
AG74308 Sawyer Emmeth Male 9/9/1971 Chaguanas 1-868-663-1981 1 1996 SA03859 Sales_Engineer
AC20852 Max Everett Male 1/1/1972 San_Fernando 1-868-665-7375 0 1998 HR03839 Training_Manager
AA21466 Ethan Rory Male 2/28/1973 Couva 1-868-629-1094 0 2000 MA03184 Marketing_Manager
AL08537 Anthony Hayden Male 1/14/1974 Diego_Martin 1-868-639-7173 0 2002 AC03045 Bookkeeper
AF08211 Abigail Rhys Female 1/13/1975 Morvant 1-868-631-0020 1 1997 HR03839 Recruiter
AB07200 Brandon Hayes Male 1/18/1979 Diego_Martin 1-868-665-9545 3 1997 FI03456 Chief_Financial_Officer
AL09377 Ezekiel Jayden Male 3/11/1980 Morvant 1-868-665-8434 4 2003 SA03859 Sales_Asst
AD07365 Elijah Brayden Male 10/7/1980 Chaguanas 1-868-631-1302 5 2001 CS03052 Receptionist
AAZ9035 Haiden Bentley Male 9/25/1984 Couva 1-868-639-9940 0 2000 OP03768 Operations_Engineer
AO07476 Beau Harrison Female 11/9/1984 Arima 1-868-653-1627 1 1999 HR03839 HR_Manager
AP24871 Jake Mason Male 1/9/1987 San_Fernando 1-868-628-5086 4 2001 MA03184 Creative_Director
AN31751 Jake Hunter Male 9/19/1988 Chaguanas 1-868-637-8467 0 2000 SA03859 Sales_Manager

Work_Information.txt :

AB28102 17.56 10 2 17
AC04855 19.20 8 0 20
AF89345 19.45 9 2 23
AB87852 20.00 7 1 19
AT59893 17.98 10 0 20
AD98455 20.40 8 2 20
AY39856 19.85 7 2 21
AN08456 19.00 6 0 21
AR40856 21.34 9 3 22
AR90998 17.09 9 0 23
AJ08735 17.23 9 3 19
AQ98045 18.30 8 0 20
AT07875 21.34 8 1 20
AB00865 19.37 10 0 23
AW79644 18.39 9 2 22
AG08801 18.38 11 0 21
AV07865 19.02 6 5 23
AL07866 20.38 8 2 21
AU76542 20.71 6 0 21
AL07542 21.90 9 1 21
AM05326 19.28 7 3 20
AX02587 18.92 9 0 21
AE74184 20.93 10 4 20
AB79420 17.29 9 2 19
AV84211 18.20 8 0 20
AL08731 21.39 10 3 21
AL94522 22.34 8 3 19
AQ02642 21.39 7 4 20

AH00128 18.28 8 5 21
AV08345 17.38 7 5 22
AS32097 18.37 10 2 23
AA20173 18.23 10 4 19
AL30236 17.34 8 0 18
AV03281 20.47 9 4 19
AZ54087 20.43 10 2 20
AJ30857 18.39 6 5 20
AR24611 19.30 7 0 21
AY39750 20.39 9 2 19
AG74308 19.90 10 4 18
AC20852 22.34 8 2 17
AA21466 20.39 9 4 20
AL08537 17.92 6 4 21
AF08211 18.20 10 0 19
AB07200 20.48 6 4 18
AL09377 19.30 8 1 17
AD07365 18.92 9 2 20
AAZ9035 20.02 9 0 21
AO07476 18.30 7 2 23
AP24871 18.23 10 0 21
AN31751 19.73 8 2 17

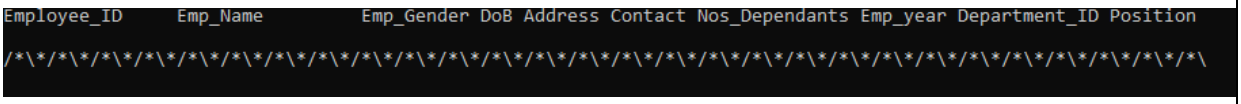
TYPE OF INPUT	DESCRIPTION	EXPECTED RESULTS	ACTUAL RESULTS	SUCCSESS
Normal	After opening the program user would select option 2 from the main menu. They would be directed to enter the employee's ID. When the employee enters the correct ID, the user would then have access to that employee's Pay slip.	From the data entered, the program should display the employee's information, their Gross Pay, Total Deductions as well as their Net Pay	The program accepted the employees ID and printed the following: Please Enter the employees' ID: AB28102 ---------- Employee Information Name of Employee: Penelope Santiago Address: Arouca Contact: 1-868-781-0695 Department ID: FI03456 Position: Internal_Auditor	Test case was successful

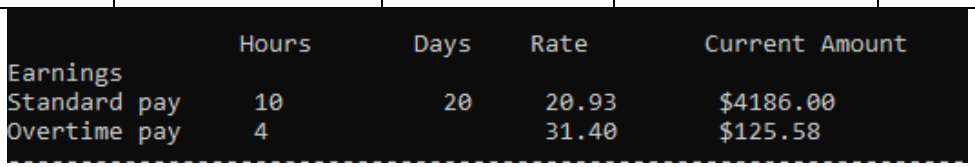
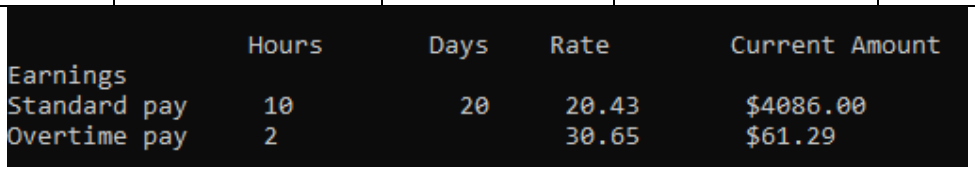
<table>
<tr><td></td><td colspan="4">

```
Please Enter the employess's ID: AB28102

--------------------------------------------------------------------------------
Employee Information

Name of Employee: Penelope Santiago
Address: Arouca
Contact: 1-868-781-0695
Department ID: FI03456
Position: Internal_Auditor

                Hours       Days    Rate        Current Amount
Earnings
Standard pay     8            21     18.28       $3071.04
Overtime pay     5                   27.42       $137.10
--------------------------------------------------------------------------------

                       Gross Pay                 $3208.14
--------------------------------------------------------------------------------

                                                 Current Amount

Deduction
National Insurance                               $20.00
Medial Plan                                      $82.63
Pension                                          $31.40
Income Tax                                       $153.86
Union                                            $42
--------------------------------------------------------------------------------

                Total Deductions                 $329.43
--------------------------------------------------------------------------------

                Net Pay                          $2878.71
--------------------------------------------------------------------------------
```

</td></tr>
<tr><td>Extreme</td><td>When prompted to enter the employee ID, the employee with the with the highest ID, who, in this case, is the individual whose identification code comes last in alphabetical order. When the employee enters their ID, the user would then have</td><td>From the data entered, the program should display the employee's information, their Gross Pay, Total Deductions as well as their Net Pay</td><td></td><td></td></tr>
</table>

	access to that employee's Pay slip			
Wrong	User entered employee ID 'AQ5790'	Program should display no employee records	Employee_ID Emp_Name Emp_Gender DoB Address Contact Nos_Dependants Emp_year Department_ID Position	Test was successful



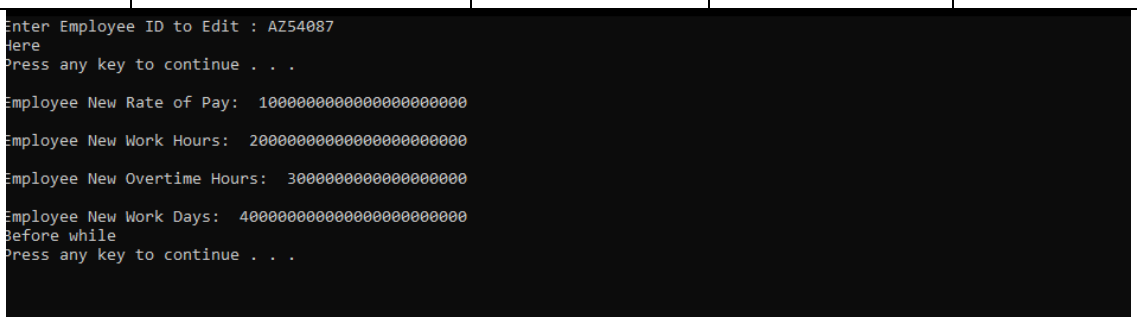
TYPE OF INPUT	DESCRIPTION	EXPECTED RESULTS	ACTUAL RESULTS	SUCCESS?
Normal	Program should calculate the amount of hours worked by employees and their rate	Standard Pay: Overtime Pay: Hours: Rate:	Standard Pay Hours :10 Overtime Pay Hours :4 Standard Rate: $20.93 Overtime Rate: $31.40	This test was successful
				
Extreme	Program should calculate employee AZ54087 hours worked and their rate	Standard Pay: Overtime Pay: Hours: Rate:	Standard Pay Hours :8 Overtime Pay Hours :2 Standard Rate: $20.43 Overtime Rate: $30.65	Test was successful
				
Erroneous	The user enters employee ID “AQ5790” to calculate their	Program should display no records	Program terminated	Test was successful

	hours worked and rates			

TYPE OF INPUT	DESCRIPTION	EXPECTED RESULTS	ACTUAL RESULTS	SUCCESS?
Normal	The employee is prompted to enter their 'employee ID' in order to be given access to edit the desired record. User entered 'AB28102'	Enter Employee ID to Edit : Here Press any key to continue . . . Employee New Rate of Pay: Employee New Work Hours: Employee New Overtime Hours: Employee New Work Days: Before while Press any key to continue . . .	Enter Employee ID to Edit : AB28102 Here Press any key to continue . . . Employee New Rate of Pay: 20 Employee New Work Hours: 40 Employee New Overtime Hours: 8 Employee New Work Days: 5 Before while Press any key to continue . . .	This test was successful

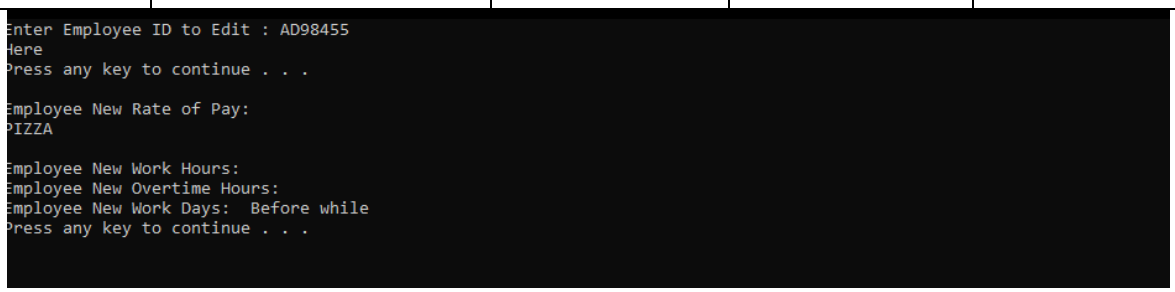
Extreme	The employee is prompted to enter their 'employee ID' in order to be given access to edit the desired record. User entered 'AZ54087'	Enter Employee ID to Edit : Here Press any key to continue . . . Employee New Rate of Pay: Employee New Work Hours: Employee New Overtime Hours: Employee New Work Days:	Enter Employee ID to Edit : AZ54087 Here Press any key to continue . . . Employee New Rate of Pay: 10000000000000000000000 Employee New Work Hours: 20000000000000000000000 Employee New Overtime Hours: 30000000000000000000 Employee New Work Days:	This test was successful

		Before while Press any key to continue . . .	400000000000000000000000 Before while Press any key to continue . . .	



Erroneous	The user is prompted to enter ‘employee ID’ in order to be given access to edit the desired record. However, the user enters an ID that does not exist. User entered ‘AD98455’	Enter Employee ID to Edit : Here Press any key to continue . . . Employee New Rate of Pay: Employee New Work Hours: Employee New Overtime Hours: Employee New Work Days: Before while	Enter Employee ID to Edit : AD98455 Here Press any key to continue . . . Employee New Rate of Pay: PIZZA Employee New Work Hours: Employee New Overtime Hours: Employee New Work Days: Before while	This test was unsuccessful

		Press any key to continue . . .	Press any key to continue . . .	



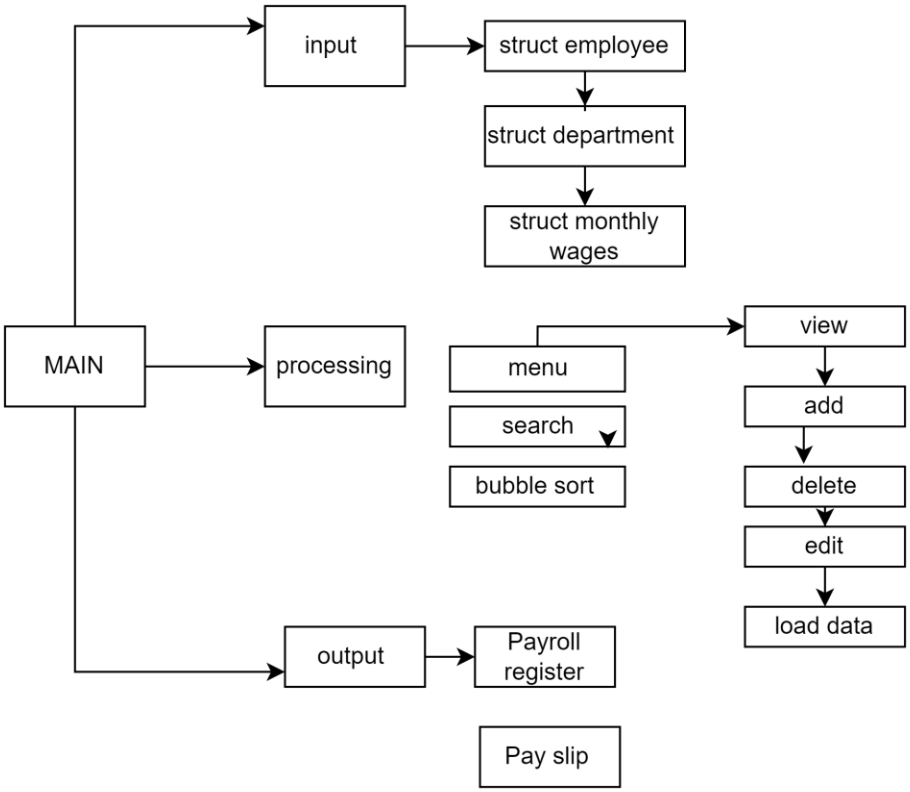
TYPE OF INPUT	DESCRIPTION	EXPECTED RESULTS	ACTUAL RESULTS	SUCCESS?
Normal	With reference to the above table, whenever the user makes changes to a record it is updated in the text file, which, in this case, is 'Work_Information.txt' The user made changes to the record of 'AB28102'	Before changes: AB28102 17.56 10 2 17 After changes: AB28102 25.00 40 8 5	Before changes: AB28102 17.56 10 2 17 After changes: AB28102 25.00 40 8 5	This test was successful
1 AB28102 17.56 10 2 17 1 AB28102 25.00 40 8 5				
Extreme	Whenever the user makes changes to a record it is updated in the text file, which, in this case, is 'Work_Information.txt' The user made changes to the record of 'AZ54087'	Before changes: AZ54087 20.43 10 2 20 After changes: AZ54087 100.00 40 12 7	Before changes: AZ54087 20.43 10 2 20 After changes: AZ54087 100.00 40 12 7	This test was successful

35 AZ54087 20.43 10 2 20 35 AZ54087 100.00 40 12 7				
Erroneous	Whenever the user makes changes to a record it is updated in the text file, which, in this case, is 'Work_Information.txt' The user made changes to the record of 'AD98455'	Before changes: AD98455 20.40 8 2 20 After changes: AD98455 20.40 8 2 20	Before changes: AD98455 20.40 8 2 20 After changes: AD98455 20.40 8 2 20	This test was unsuccessful
6 AD98455 20.40 8 2 20 6 AD98455 20.40 8 2 20				

Structure Chart

Hipo Chart

The following diagram is a Hierarchical Input Processing Output Chart, also known as a HIPO Chart. It is a combination of two organized method to analyze the computer system and provide the means of documentation required for the computer program which is created as a solution to the problem defined by John and Brothers Ltd.



In using the top-down approach in problem solving, a large complex problem is broken down into smaller manageable pieces which will then be combined to form a fully functioning program. Therefore, one can agree that the large problem in this case is the clerk needing a payroll system that will allow them to create pay slips for each employee monthly. The procedures are then broken down into three main categories: Input, output and processing, all of which will be used to solve various tasks such as editing or adding new files.

An input is data that is entered into or received by a computer. This could include a user pressing a key on a keyboard, clicking a mouse to select something on screen or tapping a touch pad. In our program input varies from employee name to department ID, however they all aid the clerk in obtaining the information they will need to carry out the calculations early and successfully.

Furthermore, in computing, a process is the instance of a computer program that is being executed by one or many threads. It contains the program code and its activity. As seen by the Hippo Chart, there are three main processes and menu is further broken down into smaller subdivisions.

Lastly, an output is data that a computer sends. The outputs include Payroll and Pay slip because this were the expected and wanted out turn of the program. The clerk now has a system them helps them efficiently produce billings for each employee at the end of the month.