#include <stdio.h>

#include <string.h>

//node struct

typedef struct node

{

//node variables

int empId;

char empFirst[30];

char empLast[30];

char empDept[30];

float empSalary;

//pointer to next node

struct node\* next;

} node\_t;

//declare functions

void addFront(node\_t\*\* head);

void addEnd(node\_t\* head);

void loadFront(node\_t\*\* head, FILE\* report);

void loadEnd(node\_t\* head, FILE\* report);

void printDept(node\_t\* head);

int verify(node\_t\* head, int num);

void payslip(node\_t\* head);

void printFile(node\_t\* head, FILE\* report);

void main()

{

//vairables

int choice;

int position;

//head of list

node\_t\* head = NULL;

//open employee file

FILE\* report;

report = fopen("Employee.txt", "r");

//load file

//load patient report

if (report == NULL)

{

//No patient records found

printf("\nPatient records not found!\n");

}

else

{

//load patient file

while (!feof(report))

{

//check if head pointer in null

if (head == NULL)

{

//head pointer is empty (not loaded)

loadFront(&head, report);

}

else

{

//list already contains items

loadEnd(head, report);

}

}//while

//close file

fclose(report);

}//else

do

{

//prompt user

printf("\nPlease Enter 1 to Add New Employee");

printf("\nPlease Enter 2 to Display Wages for Department");

printf("\nPlease Enter 3 to Generate Payslip for Employee");

printf("\nPlease Enter -1 to Exit\n");

//take inputs

scanf("%d", &choice);

if (choice == 1)

{

if (head == NULL)

{

//head pointer is empty (not loaded)

addFront(&head);

}

else

{

//list already contains items

addEnd(head);

}

}

else if (choice == 2)

{

//output depts

printDept(head);

}

else if (choice == 3)

{

//output payslip details

payslip(head);

}

} while (choice != -1);

//On exit save list

FILE\* employees;

employees = fopen("Employee.txt", "w");

printFile(head, employees);

//close file

fclose(employees);

}//main

// === My Functions === //

void addFront(node\_t\*\* head)

{

//set new node pointer

node\_t\* newNode;

//allocate memory for new node and set values

newNode = (struct node\*)malloc(sizeof(node\_t));

//set values

printf("\nEnter Employee ID: ");

scanf("%d", &newNode->empId);

printf("\nEnter Employee First Name: ");

scanf("%s", &newNode->empFirst);

printf("\nEnter Employee Last Name: ");

scanf("%s", &newNode->empLast);

printf("\nEnter Employee Department: ");

scanf("%s", &newNode->empDept);

printf("\nEnter Employee Salary: ");

scanf("%f", &newNode->empSalary);

//next pointer is the "old" head of list

newNode->next = \*head;

//new head is the new node

\*head = newNode;

}//addFront

void addEnd(node\_t\* head)

{

//local variable

int searchNum;

int found = 0; //false

//set pointers

node\_t\* temp = head;

node\_t\* newNode;

//find end of list

while (temp->next != NULL)

{

temp = temp->next;

}

//create new node

newNode = (struct node\*)malloc(sizeof(node\_t));

//set values

do

{

//validate employee ID (Not in use)

printf("\nEnter Employee ID: ");

scanf("%d", &newNode->empId);

searchNum = newNode->empId;

//call verify function

found = verify(head, searchNum);

if (found == 1)

{

printf("\nERROR - ID Already in Use!");

}

} while (found == 1);

printf("\nEnter Employee First Name: ");

scanf("%s", &newNode->empFirst);

printf("\nEnter Employee Last Name: ");

scanf("%s", &newNode->empLast);

printf("\nEnter Employee Department: ");

scanf("%s", &newNode->empDept);

printf("\nEnter Employee Salary: ");

scanf("%f", &newNode->empSalary);

//new node is end of list

newNode->next = NULL;

//second last node points to new node

temp->next = newNode;

}//addEnd

void loadFront(node\_t\*\* head, FILE\* report)

{

//file input variables

int numRead;

int id;

char first[30];

char last[30];

char dept [30];

float salary;

//check number of entries

numRead = fscanf(report, "%d %s %s %s %f", &id, first, last, dept, &salary);

if (numRead == 5)

{

if (\*head == NULL)

{

//set new node pointer

node\_t\* newNode;

//allocate memory for new node and set values

newNode = (struct node\*)malloc(sizeof(node\_t));

newNode->empId = id;

strcpy(newNode->empFirst, first);

strcpy(newNode->empLast, last);

strcpy(newNode->empDept, dept);

newNode->empSalary = salary;

//next pointer is the "old" head of list

newNode->next = \*head;

//new head is the new node

\*head = newNode;

}//if

}//if

}//loadFront

void loadEnd(node\_t\* head, FILE\* report)

{

//file input variables

int numRead;

int id;

char first[30];

char last[30];

char dept[30];

float salary;

//check number of entries

numRead = fscanf(report, "%d %s %s %s %f", &id, first, last, dept, &salary);

if (numRead == 5)

{

//set pointers

node\_t\* temp = head;

node\_t\* newNode;

//find end of list

while (temp->next != NULL)

{

temp = temp->next;

}

//create new node

newNode = (struct node\*)malloc(sizeof(node\_t));

//set values

newNode->empId = id;

strcpy(newNode->empFirst, first);

strcpy(newNode->empLast, last);

strcpy(newNode->empDept, dept);

newNode->empSalary = salary;

//new node is end of list

newNode->next = NULL;

//second last node points to new node

temp->next = newNode;

}//if

}//loadEnd

void printDept(node\_t\* head)

{

//local variable

char search[30];

float total = 0;

int found = 0; //false

//get current pointer

node\_t\* current = head;

//prompt user

printf("\nPlease Enter The Department you wish to search: ");

scanf("%s", search);

//while current not empty

while (current != NULL)

{

//search through employee data

if (strcmp(search, current->empDept) == 0)

{

found = 1; //true

//if departments match

printf("\nEmployee: %d %s %s %s %.2f", current->empId, current->empFirst, current->empLast, current->empDept, current->empSalary);

//counter for total salary

total += current->empSalary;

}

//next node pointer

current = current->next;

}//while

if (found != 1)

{

printf("\nNo Employees Match Department!\n");

}

else

{

printf("\nTotal Cost for Sales Dept: %.2f \n", total);

}

}//printList

int verify(node\_t\* head, int num)

{

//local variables

int found = 0; //false

//get current pointer

node\_t\* current = head;

//while current not empty

while (current != NULL)

{

//check PPS number

if (num == current->empId)

{

found = 1; //true

return found; //escape function

}

//next node pointer

current = current->next;

}//while

return found; //false

}//verify

void payslip(node\_t\* head)

{

//local variables

int searchNum;

int found = 0; //false

float paye;

float prsi;

float netPay;

//get current pointer

node\_t\* current = head;

//prompt user

printf("Enter Employee ID: ");

scanf("%d", &searchNum);

//while current not empty

while (current != NULL)

{

//check PPS number

if (searchNum == current->empId)

{

found = 1; //true

printf("\nEmployee: %d %s %s ", current->empId, current->empFirst, current->empLast);

printf("\nGross Pay: %.2f", current->empSalary);

//calculate PAYE

if (current->empSalary <= 1000)

{

paye = 0.00; //0%

}

else if (current->empSalary >= 1001 && current->empSalary <= 2000)

{

paye = 0.20; //20%

}

else

{

paye = 0.40; //40%

}

printf("\nPAYE: %.2f", (current->empSalary \* paye));

//calculate PRSI

prsi = current->empSalary \* 0.05;

if (prsi > 200)

{

prsi = 200;

}

printf("\nPRSI: %.2f", prsi);

//calculate NET pay

netPay = current->empSalary - (current->empSalary \* paye) - prsi;

printf("\nNet Pay: %.2f \n", netPay);

return;

}

//next node pointer

current = current->next;

}//while

if (found != 1)

{

printf("\Employee ID Not Found!\n");

}

}//payslip

void printFile(node\_t\* head, FILE\* report)

{

//get current pointer

node\_t\* current = head;

//while current not empty

while (current != NULL)

{

//output current value

fprintf(report, "%d %s %s %s %.2f\n", current->empId, current->empFirst, current->empLast,

current->empDept, current->empSalary);

//next node pointer

current = current->next;

}//while

}//printFile

// SAMPLE DATA

1 Martin Lyons Sales 5000.00

2 Roger Smith Engineering 6000.00

3 Liam Burke Sales 4000.00

4 Mickey Mouse Management 9000.00

5 Donald Duck Admin 400.00

6 Bob Painter Admin 9000.00

7 Minnie Mouse Sales 8000.00