

=====PROGRAM DOCUMENTATION=====

Leader: Darben Lamonte

Members:

Shanella Cagulang
Miyuki Parocha
Althea Amor J. Asis
Gerard Cuadra

Year & Section: BSIT 1-2

COLOR CODE:

Blue Highlight = Comments

Green Highlight = Functions

Yellow highlight = Explanation

Red highlight = Preparation

Magenta highlight = Variable

Violet highlight = Process/Formula

Gray highlight = Sample output

RUBRICS:

Rubrics Grading			
A.	Program Design 25 %	Rating	Criteria
		25	Solution well thought out
		15	Solution partially planned out
		5	Ad hoc solution; program "designed at the keyboard"
B.	Program Execution 20%	Rating	Criteria
		20	Program runs correctly
		12	Program produces correct output half of the time
		4	Program runs, but mostly incorrect
C.	Specification Satisfaction 25%	Rating	Criteria
		25	Program satisfies specification completely and correctly
		15	Many parts of the specification not implemented
		5	Program does not satisfy specification
D.	Coding Style 20%	Rating	Criteria
		20	Well-formatted, understandable code; appropriate use of language capabilities
		12	Code hard to follow in one reading; poor use of language capabilities
		4	Incomprehensive code, appropriate language capabilities not used
E.	Comments 5%	Rating	Criteria
		10	Concise, meaningful, well-formatted comments
		6	Partial, poorly written or poorly formatted comments
		4	Wordy, unnecessary, incorrect, or badly formatted comments
F.	Extra Credits 5%	Rating	Criteria
		5	Programs that usefully extend the requirements
		3	Programs that use a particularly good algorithm
		2	Program that are particularly well written or use the capabilities of the language well

Passing is a raw score of 60 points

=====PROBLEM NUMBER 1:=====

Case Study AY 2021-2022
File Handling

Create a file for the defined set of employee records that will have the following fields:

Employee Structure

Employee Number	10 characters
Employee Name	15 characters
Status Code	character (R or C)
Hours Worked	999
Deductions	99999.99

The program will have the following requirements:

1. Created file may be in text or binary file processing modes.
2. Input the required data for the following fields for n employees.
3. Status code R is for Regular and C is for Casual.
4. If status code is R, input the basic salary amount and check if the employee exceeded 160 hours of work for the month. If employee exceeded 160 hours of work for the month, basic rate is basic pay amount divided by 160 hours. Overtime rate is half more than his/her basic rate. Compute for Overtime pay.
Note: Overtime hour/s is hours worked > 160.
5. If status code is C, then input basic rate. Basic salary amount is equivalent to number of hours worked multiplied by basic rate. If hours worked exceeds 160 hours then the employee is entitled to an Overtime pay which is half more than his/her basic rate. Compute for Overtime pay.
Note: Overtime hour/s is hours worked > 160.
6. Compute for Net pay using the formula
$$\text{Net pay} = \text{Basic Pay} + \text{Overtime pay} - \text{Deductions}$$
7. Produce the output layout below:

PRINT OUTPUT:

Sample output layout:

ABC COMPANY Makati City						
Payroll						
Employee Number	Employee Name	Status	Basic Salary	Overtime Pay	Deductions	Net Pay
1234567890	Juan dela Cruz	Regular	30000.00	1500.00	2356.00	29144.00
2345678901	Maria Makiling	Casual	15000.00		1500.00	13500.00
.

Note: If Status code is equal to R then display the word "Regular".
If Status code is equal to C then display the word "Casual".

VARIABLES:

char empname[15];	// employee name
char empnum[10];	// employee number
char statcode[8];	// status code

```

int hrswork;           // hours worked
double bsalary;        // basic salary
double brate;          // basic rate
double halfbrate;      // half of basic rate
double otpay;          // overtime pay
double othours;        // overtime hours
double otrate;         // overtime rate
double netpay;         // net pay
double deduc;          // deduction

```

FUNCTIONS:

```

int ValidName(const char* name); // function to check if the name input is letters only
void input();                   // function to input employees data
void otcompu();                 // function to compute overtime pay
void regular();                 // function for regular employee
void casual();                 // function for casual employee
void create();                  // function to create the "txt" file or write to the file
void display();                 // function to display the data written to the file
void append();                  // function to append the records to the file

```

PROGRAM REQUIREMENTS:

1. Created file may be in text or binary file processing modes.

Solution: We create file in text processing mode. We use "fptr = fopen("employee.txt", "w");" to create or write to the file, and "fptr = fopen("employee.txt", "a");" to append records to the file.

2. Input the required data for the following fields for n employees.

Solution: We utilize the input(); function.

3. Status code R is for Regular and C is for Casual.

Solution: In order to display the word "Regular" and "Casual", we use a predefined function strcpy, which copies the word "Regular" if 'R' is selected, and the word "Casual" if "C" is selected.

```

    if (status code == 'R') {
        strcpy(status code, "Regular");
    }
    else if (status code == 'C') {
        strcpy(status code, "Casual");
    }

```

4. If status code is R, input the basic salary amount and check if the employee exceeded 160 hours of work for the month. If employee exceeded 160 hours of work for the

month, basic rate is basic pay amount divided by 160 hours. Overtime rate is half more than his/her basic rate. Compute for Overtime pay.

Note: Overtime hour/s is hours worked > 160.

Solution:

```
if (status code == R) {
    input basic salary
    if (hours worked > 160) {
        basic rate = basic salary / 160
        half basic rate = basic rate / 2
        overtime rate = basic rate + half basic rate
        overtime hours = hours worked - 160
        overtime pay = overtime rate * overtime hours
    }
    else {
        just display the basic salary
    }
}
```

5. If status code is C, then input basic rate. Basic salary amount is equivalent to number of hours worked multiplied by basic rate. If hours worked exceeds 160 hours then the employee is entitled to an Overtime pay which is half more than his/her basic rate.

Compute for Overtime pay.

Note: Overtime hour/s is hours worked > 160.

Solution:

```
if (status code == C) {
    input basic rate
    basic salary = basic rate * hours worked
    if (hours worked > 160) {
        half basic rate = basic rate / 2
        overtime rate = basic rate + half basic rate
        overtime hours = hours worked - 160
        overtime pay = overtime rate * overtime hours
    }
    else {
        just display the basic salary
    }
}
```

6. Compute for Net pay using the formula

Net pay = Basic Pay + Overtime pay – Deductions

7. Produce the output layout below:

CODE NO1 (*explanation*):

Code Link:

https://drive.google.com/file/d/1UxLPI5jEDPcpsw11DKovg1JzFgnzswwhy/view?usp=drive_link

/* These are the standard header files included in the code. They provide necessary functions for input/output operations, string manipulation, memory allocation, and character handling. */

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
```

/* Here, the variables number and i are declared to store the number of employees and as a loop counter, respectively. fptr is a file pointer used for file operations. */

```
int number, i;
FILE* fptr;
```

/* This code defines a structure called employee that represents the attributes of an employee. It contains fields such as employee name, number, status code, hours worked, basic salary, base rate, half of basic rate and rates for overtime pay, net pay, and deductions. */

```
struct employee {
    char empname[15];
    char empnum[10];
    char statcode[8];
    int hrswork;
    double bsalary;
    double brate;
    double halfbrate;
    double othours;
    double othrate;
    double netpay;
    double deduc;
};
```

```
struct employee emprec[100]; // Array to store employee records
```

An array of type employee is declared with a size of 100 to store employee records.

```
/* This is a function named ValidName that checks if a given name contains only alphabetic characters. It returns 1 if the name is valid and 0 otherwise. */
```

```
int ValidName(const char* name) {  
    for (int i = 0; name[i] != '\0'; i++) {  
        if (isdigit(name[i])) {  
            return 0;  
        }  
    }  
    return 1;  
}
```

```
/* This function input is used to gather employee information from the user. It prompts the user to enter the employee number, name, hours worked, and deductions. It also validates the name using the ValidName function. */
```

```
void input() {  
    printf("\n\tEnter employee number: ");  
    scanf("%s", emprec[i].empnum);  
  
    again:  
    printf("\n\tEnter employee name: ");  
    scanf("\n");  
    scanf("%[^\\n]s", emprec[i].empname);  
  
    int validName = 0;  
    if (!ValidName(emprec[i].empname)) { // checks if the name does not contain numbers  
        printf("\n\tInvalid employee name!\n\tOnly alphabetic characters are allowed.\n");  
        goto again;  
    } else {  
        validName = 1; // allowing the program to move forward  
    }  
  
    printf("\n\tHours Worked: ");  
    scanf("%d", &emprec[i].hrswork);  
  
    printf("\n\tDeductions: ");  
    scanf("%lf", &emprec[i].deduc);  
  
    printf("\n\tStatus Code [R/C]: %s ", emprec[i].statcode);  
}
```

/* The otcompu function calculates overtime pay for an employee. It computes the overtime rate, overtime hours, and overtime pay based on the basic rate and hours worked. */

```
void otcompu() {
    emprec[i].halfbrate = emprec[i].brate / 2;
    emprec[i].otrate = emprec[i].brate + emprec[i].halfbrate;
    emprec[i].othours = emprec[i].hrswork - 160;
    emprec[i].otpay = emprec[i].otrate * emprec[i].othours;
}
```

/* The regular function is used to gather additional information for regular employees. It calls the input function to get common employee details and then prompts the user to enter the basic salary. If the employee worked more than 160 hours, it calculates the overtime pay by dividing the basic salary by 160 and calls otcompu to compute the overtime-related attributes. */

```
void regular() {
    input();
    printf("\n\n\tEnter Basic Salary: ");
    scanf("%lf", &emprec[i].bsalary);

    if (emprec[i].hrswork > 160) {
        printf("\n\n\tThe employee is entitled to OVERTIME PAY!\n");
        emprec[i].brate = emprec[i].bsalary / 160;
        otcompu();
    }
}
```

/* The casual function is similar to regular but is used for casual employees. It gathers common employee details using the input function and then prompts the user to enter the basic rate. If the employee worked more than 160 hours, it calculates the overtime pay by calling otcompu. */

```
void casual() {
    input();
    printf("\n\n\tEnter Basic Rate: ");
    scanf("%lf", &emprec[i].brate);

    emprec[i].bsalary = emprec[i].brate * emprec[i].hrswork;

    if (emprec[i].hrswork > 160) {
        printf("\n\n\tThe employee is entitled to OVERTIME PAY!\n");
        otcompu();
    }
}
```

```
/* The create function is responsible for creating a new file to store employee records. It prompts
the user to enter the number of employees and the status code (regular or casual) for each
employee. Based on the status code, it calls regular or casual to gather the specific employee
information. The employee records are then written to the file "employ.txt" using the fprintf
function. */
```

```

void create() {
    char sc;
    int ex;

    fptr = fopen("employ.txt", "w"); // Open file in write mode

    system("cls");
    if (fptr == NULL) {
        printf("\n\t[File cannot be created!]\n");
        exit(1);
    }

    do {
        printf("\n\t=====[\033[32mYOU ARE CREATING\033[0m]=====\\n");
        printf("\n\tEnter the number of employees you will input: ");
        scanf("%d", &number);

        if (number <= 0) {
            printf("\n\tInvalid input! Please enter a positive number.\\n");
            fflush(stdin);
        }
    } while (number <= 0);

    printf("\n\t=====[STATUS CODE]=====\\n");
    printf("\n\t|");
    printf("\n\t| [ R ] - Regular |");
    printf("\n\t| [ C ] - Casual |");
    printf("\n\t|");
    printf("\n\t=====\\n");

    fprintf(fptr, "\\n\\t\\tABC COMPANY\\n");
    fprintf(fptr, "\\t\\tMakati City\\n");
    fprintf(fptr, "\\n\\t\\tPayroll\\n\\n");

    fprintf(fptr, "\\n\t %15s\\t%15s\\t%15s\\t%15s\\t%15s\\t%15s\\t%15s\\n", "Employee
Number", "Employee Name", "Status Code", "Basic Salary", "Overtime Pay", "Deductions", "Net
Pay" );
}

```



```

for (i = 0; i < number; i++) {
    ex = 0;
    while (ex == 0) {
        again:
        printf("\n\t[Enter status code of employee \033[32m %d \033[0m (R/C)]: ", i + 1);
        scanf(" %c", &sc);

        switch (sc) {
            case 'R':
            case 'r':
                strcpy(emprec[i].statcode, "Regular");
                regular();
                ex = 1;
                break;
            case 'C':
            case 'c':
                strcpy(emprec[i].statcode, "Casual");
                casual();
                ex = 1;
                break;
            default:
                printf("\n\t[Invalid Input!]\n\tEnter R or C only! Press any key to continue...");
                getch();
                printf("\n");
                goto again;
                break;
        }
    }
}

```

```

emprec[i].netpay = emprec[i].bsalary + emprec[i].otpay - emprec[i].deduc;

```

```

fprintf(fp, "\n\t%13s\t%17s\t%12s\t%12.2lf\t%10.2lf\t%13.2lf\t%15.2lf", emprec[i].empnum,
emprec[i].empname, emprec[i].statcode, emprec[i].bsalary, emprec[i].otpay, emprec[i].deduc,
emprec[i].netpay);
}

```

```

fclose(fp); // Close the file

```

```

printf("\n\n\t=====[You have successfully created employees' record!]======");
printf("\n\tPress any key to return to the main menu...");
getch();

```

```
/* The display function is used to read and display employee records from the file "employ.txt". It
opens the file in read mode and reads the content line by line using fgets. The employee
records are printed on the console after skipping some initial lines. */
```

```
/* The display function is used to read and display employee records from the file "employ.txt". It
opens the file in read mode and reads the content line by line using fgets. The employee
records are printed on the console after skipping some initial lines. */
```

```

size_t len = strlen(trimmedLine);
if (len > 0 && trimmedLine[len - 1] == '\n') {
    trimmedLine[len - 1] = '\0';
}

// Print employee records
printf("%s\n", trimmedLine);
}

fclose(fptr); // Close the file

getch();
return;
}

```

/* The append function is similar to the create function but is used for appending employee records to an existing file. It opens the file in append mode instead of write mode and appends the employee records to the file using fprintf. */

```

void append() {
    char sc;
    int ex;

    fptr = fopen("employ.txt", "a"); // Open file in append mode

    system("cls");
    if (fptr == NULL) {
        printf("\n\t[File cannot be created!];");
        exit(1);
    }

    printf("\n\t=====[\033[32mYOU ARE APPENDING\033[0m]=====\\n");

    do {
        printf("\n\tEnter the number of employees you will input: ");
        scanf("%d", &number);

        if (number <= 0) {
            printf("\n\tInvalid input! Please enter a positive number.\\n");
            fflush(stdin);
        }
    } while (number <= 0);

    printf("\n\t=====[STATUS CODE]=====");

```

```

printf("\n\t|");
printf("\n\t| [ R ] - Regular |");
printf("\n\t| [ C ] - Casual |");
printf("\n\t|");
printf("\n\t=====\\n");

```

```

for (i = 0; i < number; i++) {
    ex = 0;
    while (ex == 0) {
        again:
        printf("\n\t[Enter status code of employee \033[32m %d \033[0m (R/C)]: ", i + 1);
        scanf(" %c", &sc);

```

```

        switch (sc) {
            case 'R':
            case 'r':
                strcpy(emprec[i].statcode, "Regular");
                regular();
                ex = 1;
                break;
            case 'C':
            case 'c':
                strcpy(emprec[i].statcode, "Casual");
                casual();
                ex = 1;
                break;
            default:
                printf("\n\t[Invalid Input!]\n\tEnter R or C only! Press any key to continue...");
                getch();
                goto again;
                break;
        }
    }
}

```

```

emprec[i].netpay = emprec[i].bsalary + emprec[i].otpay - emprec[i].deduc;

```

```

fprintf(fptr, "\n\t%13s\t%17s\t%12s\t%12.2lf\t%10.2lf\t%13.2lf\t%15.2lf", emprec[i].empnum,
emprec[i].empname, emprec[i].statcode, emprec[i].bsalary, emprec[i].otpay, emprec[i].deduc,
emprec[i].netpay);
}

```

```

fclose(fptr); // Close the file

```

```

printf("\n\n\t=====[You have successfully appended employees' record!]=======");

```

```

printf("\n\tPress any key to return to the main menu...");
getch();
return;
}

```

/* The main function is the entry point of the program. It displays a menu to the user and repeatedly prompts for their choice until they choose to exit. Based on the user's choice, it calls the corresponding functions (create, display, append) or exits the program */

```

int main() {
    int choice;
    int close;

    close = 0;
    while (close == 0) {
        system("cls");

printf("\n\t\033[36m=====[\033[37mWELCOME!\033[36m]=====\\
\033[0m\n");

printf("\n\t\033[36m=====\\033[0m");
        printf("\n\t\033[36m|                               |\033[0m");
        printf("\n\t\033[36m| \033[37m [ 1 ] Create                |\033[36m |\033[0m");
        printf("\n\t\033[36m| \033[37m [ 2 ] Display                |\033[36m |\033[0m");
        printf("\n\t\033[36m| \033[37m [ 3 ] Append                |\033[36m |\033[0m");
        printf("\n\t\033[36m| \033[37m [ 4 ] Exit                |\033[36m |\033[0m");
        printf("\n\t\033[36m|                               |\033[0m");

printf("\n\t\033[36m=====\\033[0m\n");

        ulitpar:
        printf("\n\tEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                create(); // Create employee records
                break;
            case 2:
                display(); // Display employee records
                break;
            case 3:

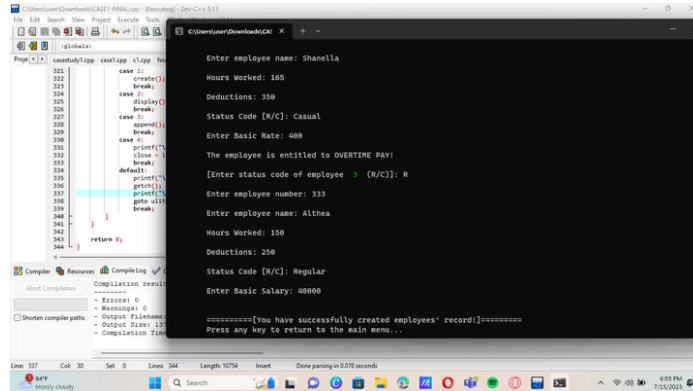
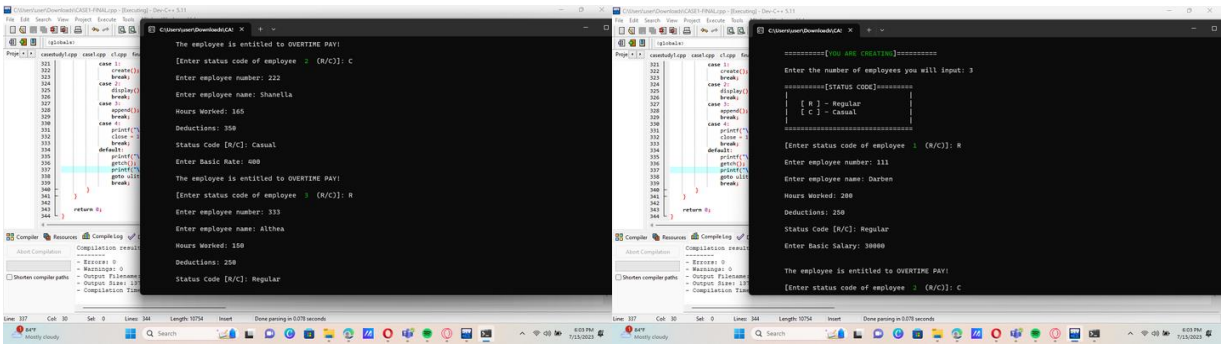
```

```
        append(); // Append employee records
        break;
    case 4:
        printf("\n\tEnd of Program...");
        close = 1; // Exit the program
        break;
    default:
        printf("\n\tPlease enter 1-4 only! Press any key to continue...");
        getch();
        printf("\n");
        goto ulitpar;
        break;
    }
}

return 0;
}
```

SAMPLE OUTPUT:

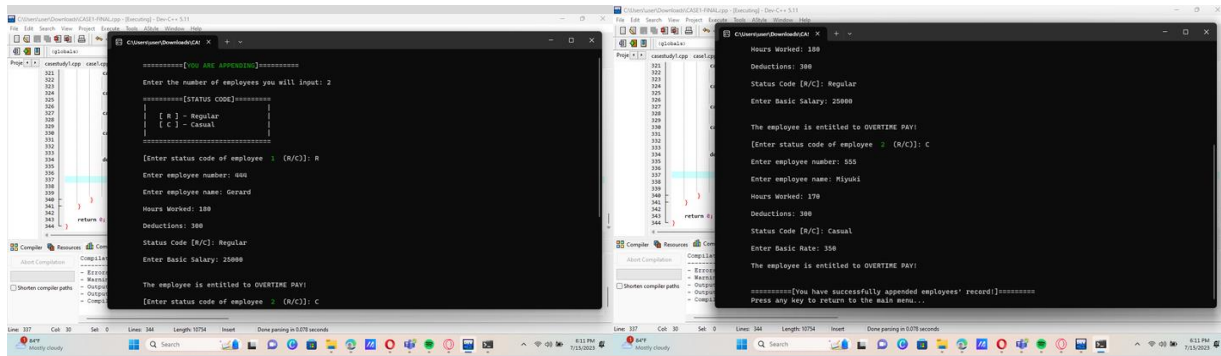
INPUT



DISPLAY

===== [ABC COMPANY] =====						
Makati City Payroll						
=====						
Employee Number	Employee Name	Status Code	Basic Salary	Overtime Pay	Deductions	Net Pay
111	Darben	Regular	30000.00	11250.00	250.00	41000.00
222	Shanella	Casual	60000.00	3000.00	350.00	68650.00
333	Althea	Regular	40000.00	0.00	250.00	39750.00

APPEND



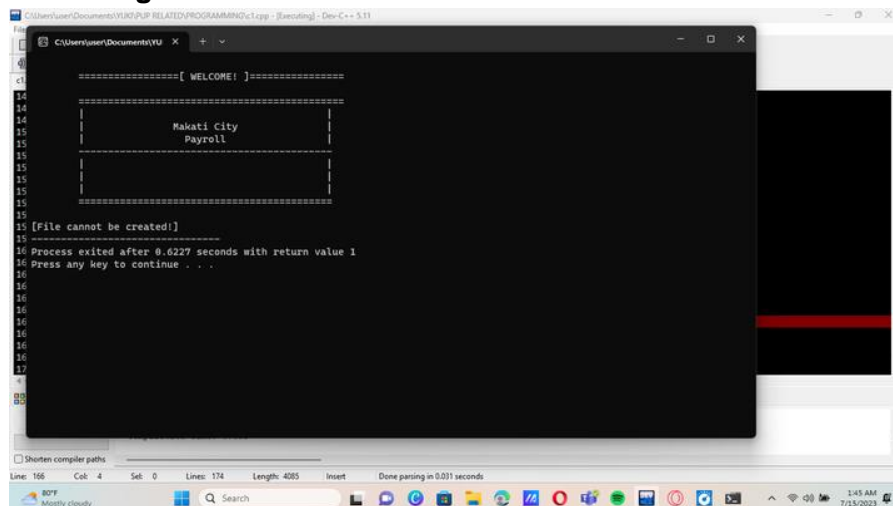
===== [ABC COMPANY] =====

Makati City Payroll

Employee Number	Employee Name	Status Code	Basic Salary	Overtime Pay	Deductions	Net Pay
111	Darben	Regular	30000.00	11250.00	250.00	41000.00
222	Shanella	Casual	60000.00	3000.00	350.00	68650.00
333	Althea	Regular	40000.00	0.00	250.00	39750.00
444	Gerard	Regular	25000.00	4687.50	300.00	29387.50
555	Miyuki	Casual	59500.00	5250.00	300.00	64450.00

[ERRORS ENCOUNTERED]

The program cannot be created, but when we try the same in a different device, the program is functioning.



=====PROBLEM NUMBER 2:=====

Case Study AY 2021-2022
Linked List

Create a menu driven program which will have the following choices:

Example:

MAIN MENU

1. Create a Doubly Linked List
2. Display a Doubly Linked List
 21. Normal order
 22. Reverse order
3. Sort a Doubly Linked List
 31. Ascending
 32. Descending
4. Insert a Node
5. Delete a Node
6. Search a Number in the List
7. Exit

Enter choice: ____

PRINT OUTPUT:

Case Study AY 2021-2022
Linked List

Create a menu driven program which will have the following choices:

Example:

MAIN MENU

1. Create a Doubly Linked List
2. Display a Doubly Linked List
 21. Normal order
 22. Reverse order
3. Sort a Doubly Linked List
 31. Ascending
 32. Descending
4. Insert a Node
5. Delete a Node
6. Search a Number in the List
7. Exit

Enter choice: ____

FUNCTIONS:

void create(); **// function to create doubly linked list**

```

void display();           //function to display the created linked list
void disnormal();        // function to display the linked list data in normal order
void disreverse();       // function to display the linked list data in normal order
void sort();             // function to sort the data in the created linked list
void ascend();           // function to sort the linked list data in ascending order
void descend();          // function to sort the linked list data in descending order
void insert();           // function to insert a new node
void dele();             // function to delete a node
void search();           // function to search a number in the created linked list

```

CODE NO. 2

Code Link:

https://drive.google.com/file/d/1FbSoSoar5JMmz8l6604XOrCC7TKfWC_e/view?usp=drive_link

CODE EXPLANATION:

These three header files are being included in the code. The stdio.h provides the input/output, the stdlib.h provides the memory allocation and the conio.h provides the console input and output.

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

```

Int choice is used to store the user's menu choice. This also has a structure named node; it represents a node in a doubly linked list. The prev is used to point the previous node in the list. Int x is used as an integer variable to store the data of the node. The next pointer is used to point the next node in the list.

```
int choice;
```

```
// structure definition
```

```

struct node {
    struct node *prev;
    int x;
    struct node *next;
};

```

```
struct node *head, *curr, *tail; // structure declaration
```

FUNCTIONS:

```
// function prototyping
```

```
void create();
void display();
void disnormal();
void disreverse();
void sort();
void ascend();
void descend();
void insert();
void dele();
void search();
```

These functions perform various operations on the doubly linked list program. Such as creating, displaying in a normal order or disreverse order, sorting, inserting, deleting and searching.

```
int main () {
```

```
int close = 0; // initialize ng value for close para masatisfy yung condition below
```

```
while(close != 1) { // magrarun continously unless close = 1
```

menu:

```
system("cls");
```

```
printf("\t=====\\n");  
    printf("\t\t\t\t\t|\\n");  
    printf("\t\t\t\t\tMAIN MENU\t\t\t|\\n");  
    printf("\t\t\t\t\t|\\n");  
printf("\t=====\\n");  
    printf("\t\t\t\t\t|\\n");  
    printf("\t\t\t\t\t1. Create a Doubly Linked List\t\t|\\n");  
    printf("\t\t\t\t\t2. Display a Doubly Linked List\t\t|\\n");  
    printf("\t\t\t\t\t3. Sort a Doubly Linked List\t\t\t|\\n");  
    printf("\t\t\t\t\t4. Insert a Node\t\t\t\t\t|\\n");  
    printf("\t\t\t\t\t5. Delete a Node\t\t\t\t\t|\\n");  
    printf("\t\t\t\t\t6. Search a number in the list\t\t|\\n");  
    printf("\t\t\t\t\t7. Exit\t\t\t\t\t\t\t|\\n");  
    printf("\t\t\t\t\t|\\n");  
    printf("\t=====\\n\\n");
```

```
printf("\t [Enter choice]: ");
```

```
scanf("%d", &choice);
```

```
system("cls");
```

```
switch (choice) {
```

```

    case 1:
        create();
    break;
    case 2:
        display();
    break;
    case 3:
        sort();
    break;
    case 4:
        insert();
    break;
    case 5:
        dele();
    break;
    case 6:
        search();
    break;
    case 7:
        printf("\n\t[ Thank you for using our program! ]\n\n");
        close = 1; // the program will exit
    break;
    default: printf("\n\n\t[ Invalid number! Choose 1-7 only. ]");
            printf(" \n\n\tPress any key to continue...\n");

    getch();
    goto menu;
}
}
printf("\n\t=====< End of program >=====");
getch();
return 0;
}

```

This main function is used to execute the program. It displays the menu based on the choice of the user. It calls the different function to perform specific operations on the program. The program will keep running unless the user chooses to exit the program.

```

void create() {

    head = curr = tail = NULL; // initialize the pointer to null

    curr = (struct node *)malloc(sizeof(struct node)); // allocate the node

    printf("\n\t=====[ OUTPUT ]=====\n\n");
}

```

```

printf("\n\t-----\n");
printf("\tEnter a value for x [ 0 to terminate ]: ");
scanf("%d", &curr->x);

while(curr->x != 0) {
if(head == NULL) {
head = curr;
head->prev = NULL;
head->next = NULL;
tail = curr;
} else {
tail->next = curr;
curr->prev = tail;
curr->next = NULL;
tail = curr;
}

curr = (struct node *)malloc(sizeof(struct node));

printf("\tEnter a value for x [ 0 to terminate ]: ");
scanf("%d", &curr->x);
}
printf("\n\t-----\n");

printf("\n\t=====< End of program >=====");
printf("\n\tPress enter to return to main menu...");
getch();
}

void disnormal() {

if (head == NULL) {
return;
}
else {
printf("\n");
curr = head;
while (curr != NULL) {
printf("\t[ %d ] ", curr->x);
curr = curr->next;
}
}
}
}

```

```

void disreverse() {

    if (head == NULL) {
        return;
    } else {
        printf("\n");
        curr = tail;
        while (curr != NULL) {
            printf("\t[ %d ] ", curr->x);
            curr = curr->prev;
        }
    }
}

void display() {
    int c;
    int exit = 0;

    while(exit == 0) {

        system("cls");
        printf("\n\t===== [ OUTPUT ]\n\t=====");
        choice:

        printf("\n\t=====");
        printf("\t| \n");
        printf("\t| How do you want your linked list data to be display? | \n");
        printf("\t|-----\n");
        printf("\t| \n");
        printf("\t| [ 1 ] Normal Order | \n");
        printf("\t| [ 2 ] Reverse Order | \n");
        printf("\t| [ 3 ] Exit | \n");
        printf("\t| \n");

        printf("\n\t=====");
        printf("\n");

        printf("\n\t[ Enter your choice ]: ");
        scanf("%d", &c);

        system("cls");
    }
}

```

```

switch (c) {
    case 1:
        printf("\n\t=====");
        printf("\n\t|");
        printf("\n\t| Linked List Data in Normal Order: |");
        printf("\n\t-----\n");
        if (head == NULL) {
            printf("\t| \n");
            printf("\t| [ Empty! ] \n");
            printf("\t| \n");
        }

        printf("\n\t===== \n");
        printf("\n\t<Create a doubly linked list first>");
        printf("\n\tPress enter to continue...");
    } else {
        disnormal();
        printf("\n\n\tPress enter to continue...");
        getch();
    }
    break;

    case 2:
        printf("\n\t=====");
        printf("\n\t|");
        printf("\n\t| Linked List Data in Reverse Order: |");
        printf("\n\t-----\n");
        if (head == NULL) {
            printf("\t| \n");
            printf("\t| [ Empty! ] \n");
            printf("\t| \n");
        }

        printf("\n\t===== \n");
        printf("\n\t<Create a doubly linked list first>");
        printf("\n\tPress enter to continue...");
    } else {
        disreverse();
        printf("\n\n\tPress enter to continue...");
        getch();
    }
    break;

    case 3:
        printf("\n\t=====< End of program >====");
        printf("\n\tPress enter to return to main menu...");
        exit = 1;
        break;
}

```

```

        default:
            printf("\n\n\t[ Invalid Input! ]\n");
            printf("\n\tPlease enter 1-3 only! Press any key to continue...");
            getch();
            system("cls");
            goto choice;
    }
    getch();
}
}

```

```

void ascend() {

    int temp;
    struct node *curr;
    struct node *tail;

    //Check whether list is empty
    if(head == NULL) {
        return;
    }
    else {
        //Current will point to head
        curr = head;
        while(curr->next != NULL) {
            //tail will point to node next to current
            tail = curr->next;
            while(tail != NULL) {
                //If current's data is greater than tail's data, swap the data of current and tail
                if(curr->x > tail->x) {
                    temp = curr->x;
                    curr->x = tail->x;
                    tail->x = temp;
                }
                tail = tail->next;
            }
            curr = curr->next;
        }
        disnormal();
    }
}

```

```

void descend() {

```



```

int temp;
struct node *curr;
struct node *tail;

//Check whether list is empty
if(head == NULL) {
    return;
}
else {
    //Current will point to head
    curr = head;
    while(curr->next != NULL) {
        // tail will point to node next to current
        tail = curr->next;
        while(tail != NULL) {
            //If current's data is less than tail's data, swap the data of current and tail
            if(curr->x < tail->x) {
                temp = curr->x;
                curr->x = tail->x;
                tail->x = temp;
            }
            tail = tail->next;
        }
        curr = curr->next;
    }
}
disnormal();
}

```

This display function is responsible for displaying the doubly linked list. It allows the user to choose what they want to display whether in normal or disreverse order.

```

void sort() {
    int c;
    int exit = 0;

    while(exit == 0) {

        system("cls");
        printf("\n\t===== [ OUTPUT
]===== \n");
        choice:
    }
}

```

```

printf("\n\t=====");
==\n");

printf("\t|                               |\n");
printf("\t| How do you want your linked list data to be sorted?           |\n");
printf("\t|-----\n");
printf("\t|                               |\n");
printf("\t| [ 1 ] Ascending Order                               |\n");
printf("\t| [ 2 ] Descending Order                               |\n");
printf("\t| [ 3 ] Exit                                           |\n");
printf("\t|                               |\n");

printf("\t=====");

\n");

printf("\n\t[ Enter choice ]: ");
scanf("%d", &c);

system("cls");
switch (c) {
    case 1:
        printf("\n\t=====");
        printf("\t|                               |");
        printf("\n\t| Linked List Data in Ascending Order: |");
        printf("\n\t|-----\n");
        if (head == NULL) {
            printf("\t|                               |\n");
            printf("\t| [ Empty! ]                               |\n");
            printf("\t|                               |\n");
        }

        printf("\n\t=====");
        printf("\n\t|--< Create a doubly linked list first >--");
        }
        else {
            ascend();
            printf("\n\n\tPress enter to continue...");
            getch();
        }
        break;
    case 2:

        printf("\n\t=====");
        printf("\n\t|                               |");
        printf("\n\t| Linked List Data in Descending Order: |");

```



```

printf("\n\t|");
printf("\n\t| Linked List Data:");
printf("\n\t-----\n");

ascend();
// check if the list is empty
if(head == NULL) {
    printf("\t| \n");
    printf("\t| [ Empty! ] \n");
    printf("\t| \n");
    printf("\t|===== \n");
    printf("\n\t--< Create a doubly linked list first >--");
    printf("\n\tPress any key to continue...");
    getch();
    return;
}

```

// create a new node and assign the value

```

struct node *newnode = (struct node *)malloc(sizeof(struct node));
printf("\n\n\tEnter a value for new node: ");
scanf("%d", &newnode->x);

```

```

newnode->prev = NULL;
newnode->next = NULL;

```

beginning

```

if(newnode->x < head->x) { //check if the new node should be inserted at the

```

```

    newnode->next = head;
    head->prev = newnode;
    head = newnode;

```

```

}

```

end

```

else if(newnode->x > tail->x) { //check if the new node should be inserted at the

```

```

    newnode->prev = tail;
    tail->next = newnode;
    tail = newnode;

```

```

}

```

else { // insert the new node in the middle of the list

```

    struct node *temp = head; // temp will point to head

```

pointer will move to the next node if true

```

    while(temp->next != NULL && temp->next->x < newnode->x) { // temp

```

```

        temp = temp->next;

```

```

    }

```

```

    newnode->prev = temp; // newnode->prev will point to temp

```

```

newnode->next = temp->next; //newnode->next will point to the node next
to temp

if(temp->next != NULL) {
    temp->next->prev = newnode; // the node next to temp will point to
new node
}

temp->next = newnode; // the new node will become the node next to
temp
}

//printing of new linked list data after insertion
again:
system("cls");

printf("\n\t===== [ After Insertion ] =====\n");
printf("\n\t=====");
printf("\n\t|");
printf("\n\t| New Linked List Data: |");
printf("\n\t-----\n");

disnormal();

printf("\n\n\tDo you want to insert another node? [ Y/N ]: ");
scanf(" %c", &deci);

switch (deci) {
    case 'Y':
    case 'y':
        goto input;
        break;
    case 'N':
    case 'n':
        printf("\n\n\tPress enter to return to main menu...");
        break;
    default:
        printf("\n\n\t[ Invalid Input ]\n\n\tPlease enter Y or N. Press enter to
continue");
        getch();
        goto again;
}
getch();
}

```

A new node is inserted into the doubly linked list using the insert function. It asks the user to provide a value for the new node and, based on that value, determines whether the node should be added at the start, middle, or end of the list. In order to properly link the new node, it updates the appropriate pointers.

```
void dele() {
    int numdel; // number to be deleted
    char d; // decision
    int exit = 0;

    system("cls");
    printf("\n\t===== [ OUTPUT ] =====\n\n");
    printf("\n\t=====");
    printf("\n\t|");
    printf("\n\t| Linked List Data:");
    printf("\n\t-----\n");

    ascend();

    if (head == NULL) {
        printf("\t|");
        printf("\t [Empty!]");
        printf("\t");
        printf("\t=====");
        printf("\n\t--< Create a doubly linked list first >--");
        printf("\n\tPress any key to continue...");
        getch();
    }
    return;
}

do {
    retry:
    printf("\n\n\tEnter the number you want to delete: ");
    scanf("%d", &numdel);

    struct node *temp = head;
    struct node *curr = NULL;

    if (temp != NULL && temp->x == numdel) {
        head = temp->next;
        free(temp);
    } else {
```

```

while (temp != NULL && temp->x != numdel) {
    curr = temp;
    temp = temp->next;
}
if (temp == NULL) {
    printf("\n\t[ The number you enter is not in the list! ]");
    goto choose;
}

    curr->next = temp->next;
    free(temp);
}

ulit:
system("cls");

printf("\n\t===== [ After Deletion ] =====\n");
printf("\n\t=====");
    printf("\n\t|");
    printf("\n\t| New Linked List Data: |");
    printf("\n\t-----\n");

disnormal();

if (head == NULL) {
    printf("\n\t|");
    printf("\n\t| [ Empty! ] |");
    printf("\n\t|");
    printf("\n\t=====");
    printf("\n\tPress any key to continue...");
    getch();
return;
}

choose:
printf("\n\n\tDo you want to try again? [ Y/N ]: ");
scanf(" %c", &d);

switch (d) {
    case 'Y':
    case 'y':
        goto retry;
        break;
    case 'N':

```

```

        case 'n':
            printf("\n\n\tPress enter to return to main menu...");
            exit = 1;
            break;
        default:
            printf("\n\n\t[ Invalid Input ]\n\n\tPlease enter Y or N. Press enter to
continue");

            getch();
            goto ulit;
    }
} while(exit != 1);

getch();
}

```

A node in the doubly linked list is removed using the dele function. It asks the user to provide a value that should be destroyed, looks for the node that has that value, and then deletes it. It deals with situations in which the node that needs to be deleted is at the start, middle, or end of the list.

```

void search() {
    struct node *ptr; // a pointer that travel accross the nodes
    int val; //value to be search
    int posi; // position
    char decide;
    int found;
    int exit;
    int occur;

    if(head == NULL) {

        system("cls");
        printf("\n\t===== [ OUTPUT ] =====\n");
        printf("\n\t=====");
        printf("\n\t|");
        printf("\n\t| Linked List Data: |");
        printf("\n\t-----\n");
        printf("\t|");
        printf("\t| [ Empty! ] |");
        printf("\t|");
        printf("\t=====");
        printf("\n\t--< Create a doubly linked list first >--");
        printf("\n\tPress any key to continue...");
        getch();
    }
}

```



```

        return;
    }

    do {
        retry:
        system("cls");

        posi = occur = found = exit = 0;

        printf("\n\n\tEnter a value to be searched: ");
        scanf("%d", &val);

        ptr = head;
        while(ptr != NULL) {
            posi++;
            if(ptr->x == val) {
                printf("\n\t[%d is in the list! It is in the node number: %d]", val,
posi);

                found = 1;
                occur++;
            }
            ptr = ptr->next;
        }
        printf("\n\n\t[There is/are %d occurrences of the number %d.]\n", occur, val);

        if(found == 0) {
            printf("\n\n\t[ %d is not in the list!]", val);
        }

        printf("\n\t===== [ FOR CHECKING ] =====\n");
        printf("\n\t=====");
        printf("\n\t|");
        printf("\n\t| Linked List Data: |");
        printf("\n\t-----\n");

        disnormal();

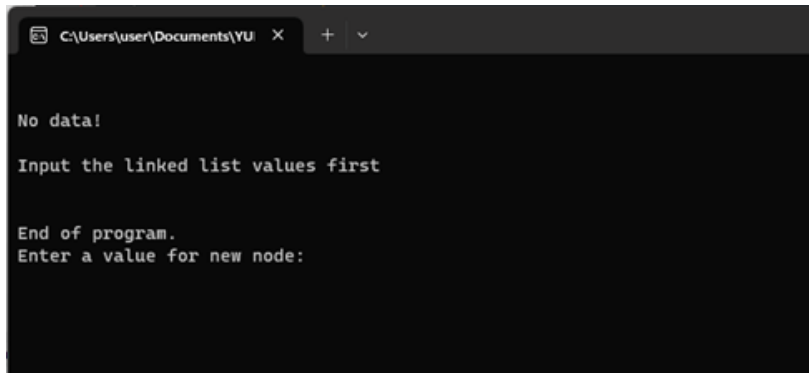
        isapa:
        printf("\n\n\tDo you want to search another value? [ Y/N ]: ");
        scanf(" %c", &decide);

        switch (decide) {
            case 'Y':
            case 'y':

```


[ERRORS ENCOUNTERED]

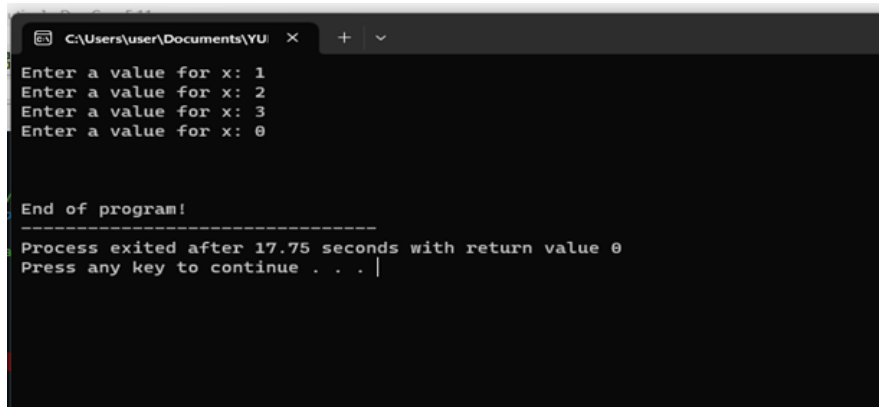
The program is spontaneously running and not going back to main



```
C:\Users\user\Documents\YU >
No data!
Input the linked list values first

End of program.
Enter a value for new node:
```

The program terminates instantly.



```
C:\Users\user\Documents\YU >
Enter a value for x: 1
Enter a value for x: 2
Enter a value for x: 3
Enter a value for x: 0

End of program!
-----
Process exited after 17.75 seconds with return value 0
Press any key to continue . . . |
```

When you are entering a letter in menu number 1 you will get this error.


```

    int hrswork;
    double bsalary;
    double brate;
    double halfbrate;
    double otpay;
    double othours;
    double otrate;
    double netpay;
    double deduc;
};

struct employee emprec[100]; // Array to store employee records

int ValidName(const char* name) {
    for (int i = 0; name[i] != '\0'; i++) {
        if (isdigit(name[i])) {
            return 0;
        }
    }
    return 1;
}

void input() {
    printf("\n\tEnter employee number: ");
    scanf("%s", emprec[i].empnum);

    again:
    printf("\n\tEnter employee name: ");
    scanf("\n");
    scanf("%[^\\n]s", emprec[i].empname);

    int validName = 0;
    if (!ValidName(emprec[i].empname)) {
        printf("\n\tInvalid employee name!\n\tOnly alphabetic characters are allowed.\n");
        goto again;
    } else {
        validName = 1; // allowing the program to move forward
    }

    printf("\n\tHours Worked: ");
    scanf("%d", &emprec[i].hrswork);

    printf("\n\tDeductions: ");

```

```

scanf("%lf", &emprec[i].deduc);

printf("\n\tStatus Code [R/C]: %s ", emprec[i].statcode);
}

void otcompu() {
    emprec[i].halfbrate = emprec[i].brate / 2;
    emprec[i].otrate = emprec[i].brate + emprec[i].halfbrate;
    emprec[i].othours = emprec[i].hrswork - 160;
    emprec[i].otpay = emprec[i].otrate * emprec[i].othours;
}

void regular() {
    input();
    printf("\n\n\tEnter Basic Salary: ");
    scanf("%lf", &emprec[i].bsalary);

    if (emprec[i].hrswork > 160) {
        printf("\n\n\tThe employee is entitled to OVERTIME PAY!\n");
        emprec[i].brate = emprec[i].bsalary / 160;
        otcompu();
    }
}

void casual() {
    input();
    printf("\n\n\tEnter Basic Rate: ");
    scanf("%lf", &emprec[i].brate);

    emprec[i].bsalary = emprec[i].brate * emprec[i].hrswork;

    if (emprec[i].hrswork > 160) {
        printf("\n\n\tThe employee is entitled to OVERTIME PAY!\n");
        otcompu();
    }
}

void create() {
    char sc;
    int ex;

    fptr = fopen("employ.txt", "w"); // Open file in write mode

```

```

system("cls");
if (fptr == NULL) {
    printf("\n\t[File cannot be created!];
    exit(1);
}

do {
    printf("\n\t=====[\033[32mYOU ARE CREATING\033[0m]=====\\n");
    printf("\n\tEnter the number of employees you will input: ");
    scanf("%d", &number);

    if (number <= 0) {
        printf("\n\tInvalid input! Please enter a positive number.\\n");
        fflush(stdin);
    }
} while (number <= 0);

printf("\n\t=====[STATUS CODE]=====");
printf("\n\t|");
printf("\n\t| [ R ] - Regular |");
printf("\n\t| [ C ] - Casual |");
printf("\n\t|");
printf("\n\t=====\\n");

fprintf(fptr, "\\n\\t\\tABC COMPANY\\n");
fprintf(fptr, "\\t\\tMakati City\\n");
fprintf(fptr, "\\n\\t\\tPayroll\\n\\n");

fprintf(fptr, "\\n\\t %15s\\t%15s\\t%15s\\t%15s\\t%15s\\t%15s\\n", "Employee
Number", "Employee Name", "Status Code", "Basic Salary", "Overtime Pay", "Deductions", "Net
Pay" );

for (i = 0; i < number; i++) {
    ex = 0;
    while (ex == 0) {
        again:
        printf("\\n\\t[Enter status code of employee \\033[32m %d \\033[0m (R/C)]: ", i + 1);
        scanf(" %c", &sc);

        switch (sc) {
            case 'R':
            case 'r':
                strcpy(emprec[i].statcode, "Regular");
                regular();

```

```

        ex = 1;
        break;
    case 'C':
    case 'c':
        strcpy(emprec[i].statcode, "Casual");
        casual();
        ex = 1;
        break;
    default:
        printf("\n\t[Invalid Input!]\n\tEnter R or C only! Press any key to continue...");
        getch();
        printf("\n");
        goto again;
        break;
    }
}

```

```

emprec[i].netpay = emprec[i].bsalary + emprec[i].otpay - emprec[i].deduc;

```

```

    fprintf(fp, "\n\t%13s\t%17s\t%12s\t%12.2lf\t%10.2lf\t%13.2lf\t%15.2lf", emprec[i].empnum,
emprec[i].empname, emprec[i].statcode, emprec[i].bsalary, emprec[i].otpay, emprec[i].deduc,
emprec[i].netpay);
}

```

```

fclose(fp); // Close the file

```

```

printf("\n\n\t=====[You have successfully created employees' record!]==");
printf("\n\tPress any key to return to the main menu...");
getch();
return;
}

```

```

void display() {
    fptr = fopen("employ.txt", "r"); // Open file in read mode

    system("cls");
    if (fptr == NULL) {
        printf("\n\t[File does not exist!];
        exit(1);
    }
}

```



```

void append() {
    char sc;
    int ex;

    fptr = fopen("employ.txt", "a"); // Open file in append mode

    system("cls");
    if (fptr == NULL) {
        printf("\n\t[File cannot be created!]\n");
        exit(1);
    }

    printf("\n\t=====[\033[32mYOU ARE APPENDING\033[0m]=====\\n");

    do {
        printf("\n\tEnter the number of employees you will input: ");
        scanf("%d", &number);

        if (number <= 0) {
            printf("\n\tInvalid input! Please enter a positive number.\\n");
            fflush(stdin);
        }
    } while (number <= 0);

    printf("\n\t=====[STATUS CODE]=====\\n");
    printf("\n\t|");
    printf("\n\t| [ R ] - Regular |");
    printf("\n\t| [ C ] - Casual |");
    printf("\n\t|");
    printf("\n\t=====\\n");

    for (i = 0; i < number; i++) {
        ex = 0;
        while (ex == 0) {
            again:
            printf("\n\t[Enter status code of employee \033[32m %d \033[0m (R/C)]: ", i + 1);
            scanf(" %c", &sc);

            switch (sc) {
                case 'R':
                case 'r':
                    strcpy(emprec[i].statcode, "Regular");
                    regular();
                    ex = 1;

```

```

        break;
    case 'C':
    case 'c':
        strcpy(emprec[i].statcode, "Casual");
        casual();
        ex = 1;
        break;
    default:
        printf("\n\t[Invalid Input!]\n\tEnter R or C only! Press any key to continue...");
        getch();
        goto again;
        break;
    }
}

emprec[i].netpay = emprec[i].bsalary + emprec[i].otpay - emprec[i].deduc;

fprintf(fp, "\n\t%13s\t%17s\t%12s\t%12.2lf\t%10.2lf\t%13.2lf\t%15.2lf", emprec[i].empnum,
emprec[i].empname, emprec[i].statcode, emprec[i].bsalary, emprec[i].otpay, emprec[i].deduc,
emprec[i].netpay);
}

fclose(fp); // Close the file

printf("\n\n\t=====[You have successfully appended employees' record!]==");
printf("\n\tPress any key to return to the main menu...");
getch();
return;
}

int main() {
    int choice;
    int close;

    close = 0;
    while (close == 0) {
        system("cls");

        printf("\n\t\033[36m=====[\033[37mWELCOME!\033[36m]=====\n\t\033[0m\n");

        printf("\n\t\033[36m=====\033[0m");
    }
}

```

```

printf("\n\t\033[36m|                                |\033[0m");
printf("\n\t\033[36m| \033[37m [ 1 ] Create          \033[36m |\033[0m");
printf("\n\t\033[36m| \033[37m [ 2 ] Display          \033[36m |\033[0m");
printf("\n\t\033[36m| \033[37m [ 3 ] Append          \033[36m |\033[0m");
printf("\n\t\033[36m| \033[37m [ 4 ] Exit          \033[36m |\033[0m");
printf("\n\t\033[36m|                                |\033[0m");

printf("\n\t\033[36m=====|\033[0m\n");

```

```

ulitpar:
printf("\n\tEnter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        create(); // Create employee records
        break;
    case 2:
        display(); // Display employee records
        break;
    case 3:
        append(); // Append employee records
        break;
    case 4:
        printf("\n\tEnd of Program...");
        close = 1; // Exit the program
        break;
    default:
        printf("\n\tPlease enter 1-4 only! Press any key to continue...");
        getch();
        printf("\n");
        goto ulitpar;
        break;
}
}

return 0;
}

```

CODE NO. 2 (FINAL):

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```

```
int choice;
```

```
// structure definition
struct node {
    struct node *prev;
    int x;
    struct node *next;
};
```

```
struct node *head, *curr, *tail; // structure declaration
```

```
// function prototyping
void create();
void display();
void disnormal();
void disreverse();
void sort();
void ascend();
void descend();
void insert();
void dele();
void search();
```

```
int main () {
```

```
    int close = 0; // initialize ng value for close para masatisfy yung condition below
```

```
    while(close != 1) { // magrarun continuously unless close = 1
```

```
        menu:
```

```
        system("cls");
```

```
        printf("\t\t033[35m=====033[0m\n");
```

```
        printf("\t\t033[35m|                                |033[0m\n");
```

```
        printf("\t\t033[35m|                033[37mMAIN MENU                |033[35m|\n");
```

```
        printf("\t\t033[35m|                                |033[0m\n");
```

```
        printf("\t\t033[35m|=====033[0m\n");
```

```
        printf("\t\t033[35m|                                |033[0m\n");
```

```
        printf("\t\t033[35m|    033[37m1. Create a Doubly Linked List    |033[35m|\n");
```

```

printf("\t\033[35m| \033[37m2. Display a Doubly Linked List \033[35m|\n");
printf("\t\033[35m| \033[37m3. Sort a Doubly Linked List \033[35m|\n");
printf("\t\033[35m| \033[37m4. Insert a Node \033[35m|\n");
printf("\t\033[35m| \033[37m5. Delete a Node \033[35m|\n");
printf("\t\033[35m| \033[37m6. Search a number in the list \033[35m|\n");
printf("\t\033[35m| \033[37m7. Exit \033[35m|\n");
printf("\t\033[35m| \033[0m|\n");
printf("\t\033[35m===== \033[0m|\n");

```

```

printf("\t [Enter choice]: ");
scanf("%d", &choice);

```

```

system("cls");

```

```

switch (choice) {
    case 1:
        create();
        break;
    case 2:
        display();
        break;
    case 3:
        sort();
        break;
    case 4:
        insert();
        break;
    case 5:
        dele();
        break;
    case 6:
        search();
        break;
    case 7:
        printf("\n\t[ Thank you for using our program! ]\n\n");
        close = 1; // the program will exit
        break;
    default: printf("\n\n\t[ \033[31mInvalid number! Choose 1-7 only. \033[0m]");
        printf(" \n\n\tPress any key to continue...");
        getch();
        goto menu;
}

}

printf("\n\t===== < \033[31mEnd of program \033[0m> =====");

```

```

    getch();
    return 0;
}

void create() {

    head = curr = tail = NULL; // initialize the pointer to null

    curr = (struct node *)malloc(sizeof(struct node)); // allocate the node

    printf("\n\t\033[36m===== \033[0m\n");
    printf("\n\t\033[36m|          \033[37mOUTPUT          \033[36m|\033[0m");
    printf("\n\t\033[36m-----\033[0m\n\n");
    printf("\tEnter a value for x \033[33m [ 0 to terminate ]\033[0m : ");
    scanf("%d", &curr->x);

    while(curr->x != 0) {
        if(head == NULL) {
            head = curr;
            head->prev = NULL;
            head->next = NULL;
            tail = curr;
        } else {
            tail->next = curr;
            curr->prev = tail;
            curr->next = NULL;
            tail = curr;
        }
    }

    curr = (struct node *)malloc(sizeof(struct node));

    printf("\tEnter a value for x \033[33m [ 0 to terminate ]\033[0m : ");
    scanf("%d", &curr->x);
    }
    printf("\n\t\033[36m-----\033[0m\n");

    printf("\n\t===== <\033[31m End of program\n");
    printf("\n\tPress enter to return to main menu...");
    getch();
}

```

```

void disnormal() {

    if (head == NULL) {
        return;
    }
    else {
        printf("\n");
        curr = head;
        while (curr != NULL) {
            printf("\t[ %d ] ", curr->x);
            curr = curr->next;
        }
    }
}

void disreverse() {

    if (head == NULL) {
        return;
    } else {
        printf("\n");
        curr = tail;
        while (curr != NULL) {
            printf("\t[ %d ] ", curr->x);
            curr = curr->prev;
        }
    }
}

void display() {
    int c;
    int exit = 0;

    while(exit == 0) {

        system("cls");

        printf("\n\t\033[32m=====
=====\\033[0m");
        printf("\n\t\033[32m|                                \033[37mOUTPUT
\033[32m|\033[0m");

        choice:

```



```

printf("\n\t\033[32m|=====
=====|\n");
printf("\t\033[32m| \033[37mHow do you want your linked list data to be display?
\033[32m|\033[0m\n");
printf("\t\033[32m|-----
\033[32m|\033[0m\n");
printf("\t\033[32m|                                     \033[32m|\033[0m\n");
printf("\t\033[32m|\033[0m [ 1 ] Normal Order
\033[32m|\033[0m\n");
printf("\t\033[32m|\033[0m [ 2 ] Reverse Order
\033[32m|\033[0m\n");
printf("\t\033[32m|\033[0m [ 3 ] Exit
\033[32m|\033[0m\n");
printf("\t\033[32m|                                     \033[32m|\033[0m\n");

printf("\t\033[32m=====
=====|\033[0m\n");

printf("\n\t[ Enter your choice ]: ");
scanf("%d", &c);

system("cls");
switch (c) {
    case 1:

printf("\n\t\033[32m=====");
printf("\n\t\033[32m| \033[0mLinked List Data in Normal Order:
\033[32m|");

printf("\n\t\033[32m=====|\033[0m\
n");

        if (head == NULL) {
            printf("\t\033[32m|                                     |\n");
            printf("\t\033[32m| \033[31m[ Empty! ]
\033[32m|\n");
            printf("\t\033[32m|                                     |\n");

printf("\t\033[32m=====|\033[0m\n
");
printf("\n\t=====< \033[31mCreate a doubly linked list first\033[0m
>=====\n");
printf("\n\tPress enter to continue...");
} else {

```

```

        disnormal();
        printf("\n\n\tPress enter to continue...");
        getch();
    }
    break;
case 2:

    printf("\n\t\033[32m=====");
    printf("\n\t\033[32m|  \033[0mLinked List Data in Reverse Order:
\033[32m|");

    printf("\n\t\033[32m===== \033[0m\
n");

        if (head == NULL) {
            printf("\t\033[32m|                                |\n");
            printf("\t\033[32m|                                \033[31m[ Empty! ]
\033[32m|\n");
            printf("\t\033[32m|                                |\n");

            printf("\t\033[32m===== \033[0m\n
");
            printf("\n\t====< \033[31mCreate a doubly linked list first\033[0m
>====\n");

            printf("\n\tPress enter to continue...");
            } else {
                disreverse();
                printf("\n\n\tPress enter to continue...");
                getch();
            }
            break;
case 3:
    printf("\n\t====< \033[31mEnd of program
\033[37m>==== \033[0m\n\n");
    printf("\n\tPress enter to return to main menu...");
    exit = 1;
    break;
default:
    printf("\n\n\t\033[31m Invalid Input! \033[37m]\033[0m\n");
    printf("\n\tPlease enter 1-3 only! Press any key to continue...");
    getch();
    system("cls");
    goto choice;
}
getch();

```

```
    }  
}
```

```
void ascend() {
```

```
    int temp;  
    struct node *curr;  
    struct node *tail;
```

```
    //Check whether list is empty
```

```
    if(head == NULL) {  
        return;  
    }
```

```
    else {
```

```
        //Current will point to head
```

```
        curr = head;
```

```
        while(curr->next != NULL) {
```

```
            //tail will point to node next to current
```

```
            tail = curr->next;
```

```
            while(tail != NULL) {
```

```
                //If current's data is greater than tail's data, swap the data of current and tail
```

```
                if(curr->x > tail->x) {
```

```
                    temp = curr->x;
```

```
                    curr->x = tail->x;
```

```
                    tail->x = temp;
```

```
                }
```

```
                tail = tail->next;
```

```
            }
```

```
            curr = curr->next;
```

```
        }
```

```
    }
```

```
    disnormal();
```

```
}
```

```
void descend() {
```

```
    int temp;  
    struct node *curr;  
    struct node *tail;
```

```
    //Check whether list is empty
```

```
    if(head == NULL) {  
        return;  
    }
```

```
}
```

```

else {
    //Current will point to head
    curr = head;
    while(curr->next != NULL) {
        // tail will point to node next to current
        tail = curr->next;
        while(tail != NULL) {
            //If current's data is less than tail's data, swap the data of current and tail
            if(curr->x < tail->x) {
                temp = curr->x;
                curr->x = tail->x;
                tail->x = temp;
            }
            tail = tail->next;
        }
        curr = curr->next;
    }
    }
    disnormal();
}

void sort() {
    int c;
    int exit = 0;

    while(exit == 0) {

        system("cls");

        printf("\n\t\033[36m=====
=====\\033[0m");
        printf("\n\t\033[36m|                                \033[37mOUTPUT
\033[36m|\033[0m");
        choice:

        printf("\n\t\033[36m|=====
=====|\\033[0m\n");
        printf("\t\033[36m|  \033[37mHow do you want your linked list data to be sorted?
\033[36m|\n");
        printf("\t\033[36m|-----\033[36m|\n");
        printf("\t\033[36m|                                \033[36m|\n");
        printf("\t\033[36m| \033[37m[ 1 ] Ascending Order
\033[36m|\n");

```

```

printf("\t\033[36m\ \033[37m[ 2 ] Descending Order
\033[36m\n");
printf("\t\033[36m\ \033[37m[ 3 ] Exit                                \033[36m\n");
printf("\t\033[36m\                                \033[36m\n");

printf("\033[36m\t=====
=====\\033[0m\n");

printf("\n\t[ Enter choice ]: ");
scanf("%d", &c);

system("cls");
switch (c) {
    case 1:

printf("\n\t\033[35m=====");
printf("\n\t\033[35m\ \033[37mLinked List Data in Ascending
Order:    \033[35m");
printf("\n\t\033[35m|-----
|\033[0m\n");

        if (head == NULL) {
            printf("\t\033[35m\                                |\n");
            printf("\t\033[35m\                                \033[31m[ Empty! ]
\033[35m|\033[0m\n");
            printf("\t\033[35m\                                |\n");

printf("\t\033[35m=====\\03
3[0m\n");

            printf("\n\t===< \033[31mCreate a doubly linked list first\033[0m
>====\n");

            printf("\n\tPress enter to continue...");
        }
        else {
            ascend();
            printf("\n\n\tPress enter to continue...");
            getch();
        }
        break;
    case 2:

printf("\n\t\033[35m=====");
printf("\n\t\033[35m\ \033[37mLinked List Data in Descending
Order:    \033[35m");

```

```

printf("\n\t\033[35m|-----
\033[0m\n");
if (head == NULL) {
    printf("\t\033[35m|                                     |\n");
    printf("\t\033[35m|                                     \033[31m[ Empty! ]
\033[35m|\033[0m\n");
    printf("\t\033[35m|                                     |\n");

    printf("\t\033[35m===== \03
3[0m\n");
    printf("\n\t===< \033[31mCreate a doubly linked list first\033[0m
>====\n");
    printf("\n\tPress enter to continue...");
}
else {
    descend();
    printf("\n\n\tPress enter to continue...");
    getch();
}
break;
case 3:
    printf("\n\t=====< \033[31mEnd of program
\033[37m>===== \033[0m\n\n");
    printf("\n\tPress enter to return to main menu...");
    exit = 1;
    break;
default:
    printf("\n\n\t\033[0m[\033[31m Invalid Input! \033[37m]\033[0m\n");
    printf("\n\tPlease enter 1-3 only! Press any key to continue...");
    getch();
    system("cls");
    goto choice;
}
getch();
}
}

```

```

void insert() {

```

```

    char deci; // decision

```

```

    input:

```

```

    system("cls");

```

```

printf("\n\t\033[32m===== \033[0m");
printf("\n\t\033[32m|          \033[37mOUTPUT          \033[32m|\033[0m");
printf("\n\t\033[32m|=====| \033[0m\n");
printf("\t\033[32m|          \033[37mLinked List Data:          \033[32m|\n");

printf("\t\033[32m|=====| \033[0m\n");

ascend();
// check if the list is empty
if(head == NULL) {
    printf("\t\033[32m|          |\n");
    printf("\t\033[32m|          \033[31m[ Empty! ]          \033[32m|\n");
    printf("\t\033[32m|          |\n");
    printf("\t\033[32m|===== \033[0m\n");
    printf("\n\t===== < \033[31mCreate a doubly linked list first\033[0m >===== \n");
    printf("\n\tPress any key to continue...");
    getch();
    return;
}

// create a new node and assign the value
struct node *newnode = (struct node *)malloc(sizeof(struct node));
printf("\n\n\t\033[33mEnter a value for new node: \033[0m");
scanf("%d", &newnode->x);

newnode->prev = NULL;
newnode->next = NULL;

if(newnode->x < head->x) { //check if the new node should be inserted at the
beginning
    newnode->next = head;
    head->prev = newnode;
    head = newnode;
}
else if(newnode->x > tail->x) { //check if the new node should be inserted at the
end
    newnode->prev = tail;
    tail->next = newnode;
    tail = newnode;
}
else { // insert the new node in the middle of the list
    struct node *temp = head; // temp will point to head
    while(temp->next != NULL && temp->next->x < newnode->x) { // temp
pointer will move to the next node if true

```

```

        temp = temp->next;
    }
    newnode->prev = temp; // newnode->prev will point to temp
    newnode->next = temp->next; //newnode->next will point to the node next
to temp

    if(temp->next != NULL) {
        temp->next->prev = newnode; // the node next to temp will point to
new node
    }

    temp->next = newnode; // the new node will become the node next to
temp
}

//printing of new linked list data after insertion
again:
system("cls");

printf("\n\t\033[35m===== \033[37m[ After Insertion
]\033[35m===== \n");
printf("\n\t\033[35m===== \n");
printf("\n\t\033[35m|          \033[37mNew Linked List Data:      \033[35m|");
printf("\n\t\033[35m----- \033[0m\n");

disnormal();

printf("\n\n\tDo you want to insert another node? [ Y/N ]: ");
scanf(" %c", &deci);

switch (deci) {
    case 'Y':
    case 'y':
        goto input;
        break;
    case 'N':
    case 'n':
        printf("\n\n\tPress enter to return to main menu...");
        break;
    default:
        printf("\n\n\t\033[32m[\033[31m Invalid Input \033[32m]\n\n\t\033[37mPlease
enter Y or N. Press enter to continue\033[0m");
        getch();
        goto again;

```



```

        }
        getch();
    }

void dele() {
    int numdel; // number to be deleted
    char d; // decision
    int exit = 0;

    system("cls");
    printf("\n\t\033[32m===== \033[0m");
    printf("\n\t\033[32m|          \033[37mOUTPUT          \033[32m| \033[0m");
    printf("\n\t\033[32m|===== | \033[0m\n");
    printf("\t\033[32m|          \033[37mLinked List Data:          \033[32m| \n");
    printf("\t\033[32m|===== | \033[0m\n");

    ascend();

    if (head == NULL) {
        printf("\t\033[32m|          | \n");
        printf("\t\033[32m|          \033[31m[ Empty! ]          \033[32m| \n");
        printf("\t\033[32m|          | \n");
        printf("\t\033[32m|===== \033[0m\n");
        printf("\n\t====< \033[31mCreate a doubly linked list first\033[0m >==== \n");
        printf("\n\tPress any key to continue...");
        getch();
        return;
    }

    do {
        retry:
        printf("\n\n\tEnter the number you want to delete: ");
        scanf("%d", &numdel);

        struct node *temp = head;
        struct node *curr = NULL;

        if (temp != NULL && temp->x == numdel) {
            head = temp->next;
            delete(temp);
        } else {
            while (temp != NULL && temp->x != numdel) {
                curr = temp;
                temp = temp->next;
            }
        }
    } while (1);
}

```

```

    }
    if (temp == NULL) {
        printf("\n\t\033[31mThe number you enter is not in the list!\033[0m");
        goto choose;
    }

    curr->next = temp->next;
    delete(temp);
}

ulit:
system("cls");

printf("\n\t\033[32m===== \033[37m[ After Deletion
]\033[32m===== \n");
printf("\n\t\033[32m=====");
printf("\n\t\033[32m|          \033[37mNew Linked List Data:      \033[32m|");
printf("\n\t\033[32m-----\033[0m");

disnormal();

if (head == NULL) {
    printf("\t|          |\n");
    printf("\t\033[31m| [ Empty! ]          \033[31m|\n");
    printf("\t|          |\n");
    printf("\t===== \n");
    printf("\n\tPress any key to continue...");
    getch();
return;
}

choose:
printf("\n\n\tDo you want to try again? [ Y/N ]: ");
scanf(" %c", &d);

switch (d) {
    case 'Y':
    case 'y':
        goto retry;
        break;
    case 'N':
    case 'n':
        printf("\n\n\tPress enter to return to main menu...");
        exit = 1;

```

```

        break;
    default:
        printf("\n\n\t[ \033[31mInvalid Input \033[0m]\n\n\tPlease enter Y or N.
Press enter to continue");
        getch();
        goto ulit;
    }
} while(exit != 1);

getch();
}

void search() {
    struct node *ptr; // a pointer that travel accross the nodes
    int val; //value to be search
    int posi; // position
    char decide;
    int found;
    int exit;
    int occur;

    if(head == NULL) {

        system("cls");

        printf("\n\t\033[34m===== \033[0m");
        printf("\n\t\033[34m|          \033[37mOUTPUT          \033[34m|\033[0m");
        printf("\n\t\033[34m|=====|\033[0m\n");
        printf("\t\033[34m|          \033[37mLinked List Data:          \033[34m|\n");

        printf("\t\033[34m|=====|\033[0m\n");
        printf("\t\033[34m|                               |\n");
        printf("\t\033[34m|          \033[31m[ Empty! ]          \033[34m|\n");
        printf("\t\033[34m|                               |\n");
        printf("\t\033[34m|=====|\033[0m\n");
        printf("\n\t===< \033[31mCreate a doubly linked list first\033[0m >===\n");
        printf("\n\tPress any key to continue...");
        getch();
        return;
    }

    do {
        retry:
        system("cls");

```

```

posi = occur = found = exit = 0;

printf("\n\n\tEnter a value to be searched: \033[0m");
scanf("%d", &val);

ptr = head;
while(ptr != NULL) {
    posi++;
    if(ptr->x == val) {
        printf("\n\t[%d is in the list! It is in the node number: %d]", val,
posi);

        found = 1;
        occur++;
    }
    ptr = ptr->next;
}
printf("\n\n\t\033[32mThere is/are %d occurrences of the number
%d.\033[0m]\n", occur, val);

if(found == 0) {
    printf("\n\n\t\033[31m %d is not in the list!\033[0m]\n", val);
}

printf("\n\t\033[34m===== \033[37m[ FOR CHECKING
]\033[34m===== \033[0m\n");
printf("\n\t\033[34m===== \033[0m\n");
printf("\n\t\033[37mLinked List Data: \033[34m|");
printf("\n\t\033[34m----- \033[0m\n");

disnormal();

isapa:
printf("\n\n\tDo you want to search another value? [ Y/N ]: ");
scanf(" %c", &decide);

switch (decide) {
    case 'Y':
    case 'y':
        goto retry;
        break;
    case 'N':
    case 'n':
        printf("\n\n\tPress enter to return to main menu...");

```

```

        exit = 1;
        break;
    default:

        printf("\n\n\t033[31m[ Invalid Input! ]\033[0m\n\n\tPlease enter Y or N.
Press enter to continue...");
        getch();
        goto isapa;
    }
} while (exit != 1);
getch();
}

```

RUBRICS:

Rubrics Grading			
A.	Program Design 25 %	Rating	Criteria
		25	Solution well thought out
		15	Solution partially planned out
		5	Ad hoc solution; program "designed at the keyboard"
B.	Program Execution 20%	Rating	Criteria
		20	Program runs correctly
		12	Program produces correct output half of the time
		4	Program runs, but mostly incorrect
C.	Specification Satisfaction 25%	Rating	Criteria
		25	Program satisfies specification completely and correctly
		15	Many parts of the specification not implemented
		5	Program does not satisfy specification
D.	Coding Style 20%	Rating	Criteria
		20	Well-formatted, understandable code; appropriate use of language capabilities
		12	Code hard to follow in one reading; poor use of language capabilities
		4	Incomprehensible code, appropriate language capabilities not used
E.	Comments 5%	Rating	Criteria
		10	Concise, meaningful, well-formatted comments
		6	Partial, poorly written or poorly formatted comments
		4	Wordy, unnecessary, incorrect, or badly formatted comments
F.	Extra Credits 5%	Rating	Criteria
		5	Programs that usefully extend the requirements
		3	Programs that use a particularly good algorithm
		2	Program that are particularly well written or use the capabilities of the language well

Passing is a raw score of 60 points