// Defining the size of bins through constant variable delclaration

#define top\_of\_stack 5

// Header Files For various in built in Function

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<time.h>

// Defining Global Variables

int choice,number\_of\_retailers,number,j,temp,n,i,flag,level,placefound,top[5];

char temp\_number[10],temp\_name[18],temp\_branch[14],temp\_id[15],ch;

// Declaring the Variables Globally (which can be accessed anywhere in the program)

int number\_check();

void wait ( int seconds ) // http://www.cplusplus.com/reference/clibrary/ctime/time/

{

clock\_t endwait;

endwait = clock () + seconds \* CLOCKS\_PER\_SEC ;

while (clock() < endwait) {}

}

void clock\_copy();

// Modules For Stacks

void stack\_start();

void id\_scanning(); // Function to scan all the respective product\_id as well as bin\_id

void stack\_display(); // Function to display the bins

void stack\_menu();

// Modules For Linked List

void linked\_menu(); // Function to print the title of Module "STUDENT MANAGEMENT SYSTEM"

void linked\_list();

void createlist(); // Funtion to initialize the start pointer that would point towards teh firt node in list "Initially it is poinitng to NULL"

void linked\_delete(); // Function to delete a particular record from the list on basis of intake\_id

void linked\_display();

void linked\_update();

void sorting(); // This Function would place the node at its particular location after sorting on the basis of level(1st) and name wise(2nd)

void linked\_insertion(); // For scanning in the values of the appropriate fields

void id\_wise\_display(); // Displaying the record of a particular student on the basis of intake

void entire\_list\_display(); // Displaying the entire list at a time

void level\_wise\_display(); // Displaying all records of a particular level

int unique\_id\_check(); // For checking the uniqueness of the intake\_id entered

void uppercase(char \*p); // Coverting lowercase letters to uppercase

void id\_scan();

// Definition of Strcutures

struct node

{

char student\_name[18], student\_id[15], student\_branch[14];

int level\_number;

struct node \* ptr;

}\*newnode,\*start,\*prev,\*temp\_point,\*freepoint;

struct retailer{ // Strcuture Definition with member product\_id and bin\_id

char product\_id[10];

int bin\_id;

}\*newretail,stack\_temp[5];

struct bin\_temp{ // Structure Definition for Storing the data into bins

char product\_name[10];

}bin[5][5]; // Multidimensional array Bin represents that there are 5 array each of max size 5

void main()

{

system("cls");

// Function Call for the Initial Screen

clock\_copy();

system("cls");

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

printf("\n\t\t \* DATABASE AND DATA STRUCTURES \*");

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

printf("\n\t\t S.NO. PREFERNCES/CHOICES");

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

printf("\n\n\t\t 1 \t STUDENT MANAGEMENT SYSTEM");

printf("\n\n\t\t 2 \t AUTOMATED STOOCKROOM");

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

printf("\n\n ");

flag=0;

// DoWhile Loop for scanning input "if input wrong scan again"

do{

printf("\n\t PRESS S.NO. TO SELECT YOUR PREFERENCE = ");

// Scannning the Choice user will perform

choice = number\_check();

if(choice==1||choice==2)

{

/\* Flag is just to check the accuracy of inputted data

if 1 input is correct\*/

flag = 1;

}

else

{

printf("\a\n\t\t ERROR: ENTER EITHER 1 OR 2 ONLY\n");

flag = 0 ;

}

}while(flag!=1);

switch(choice)

{

// Function calling Linked\_list

case 1: linked\_list(); break;

// Function calling stack\_start

//case 2: stack\_start(); break;

}

}

/\* Function to check

whether the number entered is positive integer or not\*/

int number\_check()

{

int number;

temp = 1; // "temp" will act as flag to store either 0 and 1 for looping purpose

do{

if(temp == 0)

{

printf("\n\n\t ENTER THE VALUE AGAIN\t = ");

}

number=0;

gets(temp\_number); // Scanning the number through gets function defined in <string.h> library

fflush(stdin);

if(stricmp(temp\_number,"")==0)

{

printf("\a\n\t\t ERROR: FIELD CAN'T BE SKIPPED");

temp=0;

}

else

{

n=0;

for(i=0;i<strlen(temp\_number);i++)

{

// 48-57 are the ASCII values of numbers 0-9

if(temp\_number[i]>=48 && temp\_number[i]<=57)

{

n++;

switch (temp\_number[i])

{

case '0': number = number\*10 + 0; break;

case '1': number = number\*10 + 1; break;

case '2': number = number\*10 + 2; break;

case '3': number = number\*10 + 3; break;

case '4': number = number\*10 + 4; break;

case '5': number = number\*10 + 5; break;

case '6': number = number\*10 + 6; break;

case '7': number = number\*10 + 7; break;

case '8': number = number\*10 + 8; break;

case '9': number = number\*10 + 9; break;

}

}

}

if(n!=strlen(temp\_number))

{

printf("\a\n\t\t ERROR: ENTER POSITIVE INTEGERS ONLY");

temp=0;

}

else

{

temp=1;

}

}

}while(temp!=1);

return number;

}

void createlist()

{

start = NULL;

}

void linked\_list()

{

// For clearing screen defined in <stlib.h> class library

system("cls");

createlist();

do{

linked\_menu();

printf("\n\t\t S.NO. PREFERNCES/CHOICES");

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

printf("\n\n\t\t 1 \tINSERT NEW STUDENT RECORD");

printf("\n\n\t\t 2 \tPRINT THE EXISTING RECORD");

printf("\n\n\t\t 3 \tDELETE AN EXISTING RECORD");

printf("\n\n\t\t 4 \tUPDATE AN EXISTING RECORD");

printf("\n\n\t\t 5 \tQUIT TO MAIN MENU");

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

do{

printf("\n\t PRESS S.NO. TO SELECT YOUR PREFERENCE = ");

choice = number\_check();

if(choice>=1 && choice <=5)

{

flag = 1;

}

else

{

printf("\a\n\t\t ERROR: ENTER CHOICE BETWEEN 1 TO 5 ONLY\n");

flag = 0 ;

}

}while(flag!=1);

switch(choice)

{

case 1: linked\_insertion(); break;

case 2: linked\_display(); break;

case 3: linked\_delete(); break;

case 4: linked\_update(); break;

case 5: main(); break;

}

}while(choice!=5);

getch();

}

void linked\_menu()

{

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

printf("\n\t\t \* STUDENT\_MANGEMENT\_SYSTEM \*");

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

}

void linked\_insertion()

{

system("cls");

linked\_menu();

newnode = (struct node \* ) malloc(sizeof(struct node)); // Dynamically allocating memory at runtime which will return a void pointer and typecasting the void pointer to type struct node

// IF memory is not available

if (newnode == NULL)

{

printf("\n\tMEMORY NOT AVAILABLE");

exit(1);

}

else

{

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

printf("\t\t| PLEASE FILL IN THE APPROPRIATE DETAILS |\n");

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

do{

printf("\n\nPROVIDE THE STUDENT NAME = ");

fflush(stdin);

gets(temp\_name); // scan in the student name

fflush(stdin); // Clearing the buffer

flag = character\_check(temp\_name); // Checking the name for special characters as well as numeric values

if(stricmp(temp\_name,"")==0) // Checking whether name entered is empty string or not

{

printf("\a\n\tERROR: NAME FIELD CANNOT BE BLANK"); // Error Message

flag = 0;

}

}while(flag!=1);

uppercase(temp\_name); // Converting all the lowercase letters added to uppercase

strcpy(newnode -> student\_name, temp\_name); // Coping the string scanned to member of the strcuture variable newnode

id\_scan(); // Fucntion will scan and check the input for special characters as well

do {

printf("\n\nPROVIDE THE BRANCH NAME = ");

fflush(stdin);

gets(temp\_branch);

fflush(stdin);

flag = character\_check(temp\_branch); // Checking the string entered for special characters and numeric values

if(stricmp(temp\_branch,"")==0)

{

printf("\a\n\tERROR: BRANCH FIELD CANNOT BE BLANK");

flag = 0;

}

}while(flag!=1);

// Changing the lowercase letters to uppercase

uppercase(temp\_branch);

do{

//SEMESTER=level

printf("\n\nPROVIDE THE SEMESTER NUMBER = ");

level = number\_check();

// Checking whether the level/semester entered is in between 1-12 or not

if(level>=1 && level<=12)

{

flag=1;

}

else{

printf("\a\n\t ERROR: CAN ACCEPT BETWEEN 1-12 ONLY");

flag = 0;

}

}while(flag!=1);

// Copying the values of the variables to the members of structure variable newnode

strcpy(newnode -> student\_id, temp\_id);

strcpy(newnode -> student\_branch,temp\_branch);

newnode -> level\_number = level;

newnode -> ptr = NULL;

// For insertion of the record at its appropriate location

sorting();

printf("\n\n\t\t ^^ RECORD INSERTED SUCCESSFULLY ^^\n");

printf("\n PRESS ENTER TO CONTINUE......");

/\* This while loop will hold the program until

the user will press the "ENTER" key on keyboard\*/

while(getch()!=13){} //13 means enter key

system("cls");

}

}

/\* Function to check the inputted data

for special characters as well as numeric data\*/

int character\_check(char \*p)

{

n = 0;

for(i=0 ;i < strlen(p);i++)

{

/\* Comparing the ASCII values of characters

with the values of the characters entered\*/

if( \*(p+i) >=65 && \*(p+i) <=90 || \*(p+i) >= 95 && \*(p+i) <=122 || \*(p+i) == 32)

{

n++;

}

}

/\* Check whether the length of the input exceeds

the limit of the memory provided\*/

if(strlen(p)>18)

{

printf("\a\n\t\tERROR: LENGTH EXCEEDED BEYOND 18\n");

return 0;

}

if((\*(p+0)==32||\*(p+1)==32) && (strlen(p)==1||strlen(p)==2))

{

printf("\a\n\tERROR:FIELD CANNOT BE LEFT BLANK");

return 0;

}

if(n != strlen(p))

{

printf("\a\n\t\tERROR: CAN ACCEPT ALPHABETS A-Z ONLY");

return 0;

}

else

{

return 1;

}

}

// Function to check the presence of the special characters in the input

int special\_character\_check(char \*p)

{

n =0;

// strlen() will return the length of the string provided with parenthesis

for(i=0;i < strlen(p);i++)

{

if( \*(p+i) >= 48 && \*(p+i)<=57 || \*(p+i)>=65 && \*(p+i)<=90 || \*(p+i)>= 97 && \*(p+i)<=122 || \*(p+i) == 32)

{

n++;

}

}

// Checking for the limit or size of the array

if(strlen(p)>15)

{

printf("\a\n\t\tERROR: LENGTH EXCEEDED BEYOND 15\n");

return 0;

}

if((\*(p+0)==32||\*(p+1)==32) && (strlen(p)==1||strlen(p)==2))

{

printf("\a\n\tERROR:FIELD CANNOT BE LEFT BLANK");

return 0;

}

else if(n!=strlen(p))

{

printf("\a\n\t\tERROR: NO SPECIAL CHARACTERS ALLOWED\n");

return 0;

}

else

{

return 1;

}

}

// Function for Displaying in the records present in the list

void linked\_display()

{

/\* Setting a temporary pointer temp\_point to start

where start represents the first node in the list\*/

temp\_point = start;

// Clearing the Screen

system("cls");

linked\_menu();

// Checking if the list is Empty

if(temp\_point == NULL)

{

printf("\n\n\t\tINSERT SOME RECORD FIRST \"LIST IS EMPTY\"");

}

else

{

/\* Choice for the user how the user wants to display

in the records\*/

printf("\n\n ^^ WHICH WAY YOU WANT TO PRINT THE RECORDS ^^");

printf("\n\n 1. PRINT THE RECORDS (SEMISTER WISE)");

printf("\n\n 2. PRINT THE RECORDS (ID WISE)");

printf("\n\n 3. PRINT THE RECORDS (ENTIRE LIST)");

printf("\n\n");

do{

printf("\tENTER YOUR CHOICE = ");

choice = number\_check();

if(choice == 1)

{

level\_wise\_display();

flag=1;

}

else if(choice == 2)

{

id\_wise\_display();

flag=1;

}

else if(choice == 3)

{

entire\_list\_display();

flag=1;

}

else{

printf("\a\n\t\t ERROR: ENTER BETWEEN 1 TO 3 ONLY\n\n");

flag =0;

}

}while(flag!=1);

}

printf("\n\nPRESS ENTER TO CONTINUE......");

while(getch()!=13){}

system("cls");

}

void linked\_delete()

{

system("cls");

linked\_menu();

// Checking whether the list is empty

if(start == NULL)

{

printf("\a\n\n\t\tNO ELEMENT IN THE RECORD TO BE DELETED");

printf("\n\nPRESS ENTER TO CONTINUE........");

while((int)getch()!=13){}

}

else

{

// For displaying all the records before deleting a particular one

entire\_list\_display();

fflush(stdin);

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");

// Record will be deleted on the basis of the unique id

printf("\n\n\tENTER THE STUDENT ID = ");

do{

fflush(stdin);

gets(temp\_id);

temp = special\_character\_check(temp\_id);

if(strcmp(temp\_id,"")==0)

{

printf("\a\n\tERROR: STUDENT ID FIELD CANNOT BE BLANK");

temp=0;

}

else

temp=1;

fflush(stdin);

}while(temp!=1);

uppercase(temp\_id);

temp\_point=start;

placefound=0;

prev = NULL;

/\* Transversing through the list for checking

whether student\_id entered exist or not\*/

while(temp\_point!=NULL && placefound==0)

{

/\* Comparing the student\_id entered with one

that actually is present in the record\*/

if(stricmp(temp\_point -> student\_id,temp\_id)==0)

{

placefound=1;

}

else

{

// prev represents the previous node that has been checked

prev = temp\_point;

temp\_point = temp\_point ->ptr;

}

}

/\* if the record with that particular id exist

execute the if block\*/

if (placefound==1)

{

system("cls");

linked\_menu();

printf("\n\n | STUDENT NAME | STUDENT\_ID | BRANCH | Semester |");

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n ");

// Displaying that particular record that has been find out

printf("| %-18s", temp\_point -> student\_name);

printf("| %-14s", temp\_point -> student\_id);

printf("| %-14s",temp\_point -> student\_branch);

printf("| %-3d |",temp\_point->level\_number);

printf("\n\n\t=> PRESS Y TO CONFIRM\t\t=> PRESS N TO SKIP\n");

printf("\n ENTER YOUR CHOICE = ");

ch = getche();

if(ch=='Y'||ch=='y')

{

freepoint = temp\_point;

// IF there is only one record in the list

if (prev == NULL && start->ptr==NULL)

{

/\* Set the start pointer to NULL

i.e no record is present in the list\*/

start = NULL;

}

// If the first record need to be deleted

else if (prev==NULL && start->ptr!=NULL)

{

/\* Set the start pointer with

the address of the next node in the list\*/

start = start->ptr;

}

else

{

prev -> ptr = temp\_point -> ptr;

}

/\* Freeing up the memory which was occupied

by that particular node that is deleted now\*/

free(freepoint);

printf("\a\n\n\t\t ^^ RECORD DELETED SUCCESSFULLY ^^");

printf("\n\n\tPRESS ENTER TO CONTINUE.....");

while(getch()!=13){}

system("cls");

}

/\* if user presses n

then no record will be deleted from the list\*/

else if(ch=='N'||ch=='n')

{

printf("\a\n\n\t\t ^^ RECORD IS NOT DELETED ^^");

printf("\n\n\tPRESS ENTER TO CONTINUE.....");

while(getch()!=13){}

system("cls");

return ;

}

}

/\* If the record with that particular id does not exist

in the list then this else block will get executed\*/

else

{

printf("\a\n\n\tMESSAGE : THE RECORD WITH STUDENT\_ID %s DOESNOT EXIST",temp\_id);

printf("\nPRESS ENTER TO CONTINUE....");

while(getch()!=13){}

}

}

system("cls");

}

/\* Updating a pre existing record in the list

(First the particular record would be deleted

and the new record will be inserted)\*/

void linked\_update()

{

system("cls");

linked\_menu();

printf(" ^^ SELECT THE RECORD TO UPDATE ^^");

/\* Displaying all the records present in the list

to select a particular one\*/

entire\_list\_display();

// If no record exist

//record are not empty

if(start!=NULL)

{

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n");

do{

printf("\n\tENTER THE STUDENT\_ID NUMBER = ");

gets(temp\_id);

flag = special\_character\_check(temp\_id);

uppercase(temp\_id);

if(strcmp(temp\_id,"")==0)

{

printf("\a\n\tERROR: ENTER THE STUDENT\_ID FROM THE ABOVE RECORDS\n");

flag = 0;

continue;

}

}while(flag!=1);

temp\_point=start;

prev = NULL;

flag = 0;

// Until the temp\_point does not reach the end of the list

while(temp\_point!=NULL)

{

/\* Until the record to be updated is not find out

change the pointers addresses\*/

if(stricmp(temp\_id,temp\_point-> student\_id)!=0)

{

prev = temp\_point;

temp\_point = temp\_point -> ptr;

}

else

{

flag = 1;

break;

}

}

if(temp\_point==NULL || flag==0)

{

printf("\n\t RECORD WITH STUDENT\_ID %s DOESNOT EXIST",temp\_id);

}

else{

/\* If only single record exist then free up that memory,

set the start pointer to NULL and then

insert the updated record at its appropriate location\*/

if(prev==NULL && temp\_point->ptr ==NULL)

{

freepoint = temp\_point;

start = NULL;

}

else if(prev==NULL&&temp\_point->ptr!=NULL)

{

freepoint=temp\_point;

start=temp\_point->ptr;

}

else

{

prev->ptr=temp\_point->ptr;

freepoint=temp\_point;

}

strcpy(temp\_name,temp\_point->student\_name);

strcpy(temp\_branch,temp\_point->student\_branch);

level = temp\_point->level\_number;

free(freepoint);

system("cls");

linked\_menu();

printf("\n\n\t ^^ ENTER THE FIELD YOU WANT TO UPDATE ^^\n");

printf("\n\n 1. UPDATE STUDENT\_ID FOR PARTICULAR RECORD");

printf("\n\n 2. UPDATE NAME FOR PARTICULAR RECORD");

printf("\n\n 3. UPDATE BRANCH FOR PARTICULAR RECORD");

printf("\n\n 4. UPDATE SEMISTER FOR PARTICULAR RECORD");

printf("\n\n\tENTER THE CHOICE = ");

choice =number\_check();

switch(choice)

{

case 1: id\_scan();

break;

case 2: do{

printf("\nENTER THE NEW NAME = ");

fflush(stdin);

// scan in the student name

gets(temp\_name);

// Clearing the buffer

fflush(stdin);

/\* Checking the name for special characters

as well as numeric values\*/

flag = character\_check(temp\_name);

// Checking whether name entered is empty string or not

if(stricmp(temp\_name,"")==0)

{

// Error Message

printf("\a\n\tERROR: NAME FIELD CANNOT BE BLANK");

flag = 0;

}

}while(flag!=1);

break;

case 3: do{

printf("\nENTER THE NEW BRANCH = ");

fflush(stdin);

gets(temp\_branch);

fflush(stdin);

/\* Checking the string entered for

special characters and numeric values\*/

flag = character\_check(temp\_branch);

if(stricmp(temp\_branch,"")==0)

{

printf("\a\n\tERROR: BRANCH FIELD CANNOT BE BLANK");

flag = 0;

}

}while(flag!=1);

break;

case 4:printf("\nENTER THE NEW SEMESTER = ");

level = number\_check();break;

}

newnode = (struct node\*)malloc(sizeof(struct node));

uppercase(temp\_name);

uppercase(temp\_branch);

strcpy(newnode->student\_name,temp\_name);

strcpy(newnode->student\_id,temp\_id);

strcpy(newnode->student\_branch,temp\_branch);

newnode->level\_number=level;

newnode->ptr=NULL;

sorting();

printf("\n\n\t\t ^^ RECORD UPDATED SUCCESSFULLY ^^\n");

}

}

printf("\n\nPRESS ENTER TO CONTINUE......");

while(getch()!=13){}

system("cls");

}

void id\_wise\_display()

/\* Function will scan a particular Student\_id, check whether the id exist or not. if the student\_id exist then that particlaur record

will get displayed otherwise message will be displayed that the record will that particular id does not exist\*/

{

system("cls");

linked\_menu();

do{

printf("\n\tENTER THE ID NUMBER = ");

fflush(stdin);

gets(temp\_id);

temp = special\_character\_check(temp\_id);

if(strcmp(temp\_id,"")==0)

{

printf("\a\n\tERROR: STUDENT\_ID FIELD CANNOT BE BLANK");

temp=0;

}

else

temp=1;

fflush(stdin);

}while(temp!=1);

uppercase(temp\_id);

temp\_point = start;

flag = 0;

while(temp\_point!=NULL)

{

if(stricmp(temp\_id,temp\_point-> student\_id)==0)

{

flag = 1;

break;

}

temp\_point = temp\_point -> ptr;

}

if(flag == 1)

{

printf("\n\nRECORDS IN THE LIST ARE AS FOLLOW - ");

printf("\n\n | STUDENT NAME | STUDENT\_ID | BRANCH | SEMESTER |");

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");

printf("\n\n ");

printf("| %-18s", temp\_point -> student\_name);

printf("| %-14s", temp\_point -> student\_id);

printf("| %-14s",temp\_point -> student\_branch);

printf("| %-3d |",temp\_point->level\_number);

}

else if(flag==0)

{

printf("\a\n\t\tSTUDENT\_ID %s DOESNOT EXIST IN THE LIST",temp\_id);

}

}

void entire\_list\_display()

/\*Function will display all the records present in the list,

if no record exist then it will present a message that

the list is empty\*/

{

system("cls");

linked\_menu();

if(start==NULL)

{

printf("\n\t NO RECORD IN THE LIST ADD SOME RECORD FIRST");

}

else

{

printf("\n\nRECORDS IN THE LIST ARE AS FOLLOW - ");

printf("\n\n | STUDENT NAME | STUDENT\_ID | COURSE | SEMESTER |");

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");

temp\_point = start;

// Transversing through the list and display the records

while(temp\_point!=NULL )

{

printf("\n\n ");

printf("| %-18s", temp\_point -> student\_name);

printf("| %-14s", temp\_point -> student\_id);

printf("| %-14s",temp\_point -> student\_branch);

printf("| %-3d |",temp\_point->level\_number);

temp\_point = temp\_point -> ptr;

}

}

}

void level\_wise\_display()

/\* Function will display all the records

for a particular semester \*/

{

printf("\nENTER THE SEMESTER NUMBER TO DISPLAY RECORDS = ");

level = number\_check();

temp\_point = start;

system("cls");

linked\_menu();

temp=0;

printf("\n\nRECORDS IN THE LIST IS AS FOLLOW - ");

printf("\n\n | STUDENT NAME | STUDENT\_ID | COURSE | SEMESTER |");

printf("\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");

while(level >= temp\_point-> level\_number)

{

if(level == temp\_point->level\_number)

{

temp++;

printf("\n\n ");

printf("| %-18s", temp\_point -> student\_name);

printf("| %-14s", temp\_point -> student\_id);

printf("| %-14s",temp\_point -> student\_branch);

printf("| %-3d |",temp\_point->level\_number);

}

temp\_point = temp\_point -> ptr;

if(temp\_point == NULL)

{

break;

}

}

if(temp==0)

{

system("cls");

linked\_menu();

printf("\n\n\tNO RECORD IN THE LIST WITH SEMESTER = %d \nADD SOME RECORD FIRST\n\n",level);

}

}

void sorting()

{

/\* if list is empty then start will point towards

the record inputted\*/

if(start==NULL)

{

start = newnode;

}

else if(start -> level\_number > level)

/\*

if the level/semester entered is smaller than the level

of the first record in the list

then start will again point to newnode

but pointer of newnode will point towards

the address that was previously stored in pointer

start

\*/

{

newnode -> ptr = start;

start = newnode;

}

else

{

placefound = 0 ; temp\_point = start;

while(placefound==0 && temp\_point!=NULL )

{

/\* if level entered is bigger than the level

in the particular record go to next record\*/

if(temp\_point -> level\_number < level)

{

prev = temp\_point;

temp\_point = temp\_point -> ptr;

}

// If level is same the n check for the sorting of the names

else if(temp\_point -> level\_number == level)

{

/\* stricmp() will compare the two strings

regardless of its case,

if existing name in record is smaller on

the basis of ASCII values again jump to next

record\*/

if(stricmp(temp\_point -> student\_name,temp\_name)<0)

{

prev = temp\_point;

temp\_point = temp\_point -> ptr;

}

else

{

//Else the place if is found when this block will get executed

placefound = 1 ;

}

}

else

{

placefound = 1 ;

}

}

// If only one records exist in the list

if(temp\_point == start)

{

newnode -> ptr = start;

start = newnode;

}

else

{

newnode->ptr=temp\_point;

prev->ptr=newnode;

}

}

}

/\*Function will check the uniqueness of the id entered

i.e whether the student id is already present,

if yes it will display error messages\*/

int unique\_id\_check()

{

temp\_point = start;

while(temp\_point!=NULL )

{

if(stricmp(temp\_point->student\_id,temp\_id)==0)

{

printf("\n\t ERROR: ID SHOULD BE UNIQUE");

return 0;

}

else

temp\_point = temp\_point -> ptr;

}

return 1;

}

void uppercase(char \*p)

{

for(i=0;i<strlen(p);i++)

{

\*(p+i) = toupper( \*(p+i) );

}

}

void id\_scan()

{

do{

printf("\n\nPROVIDE THE STUDENT ID NUMBER = ");

fflush(stdin);

gets(temp\_id);

fflush(stdin);

flag = special\_character\_check(temp\_id);

if(stricmp(temp\_id,"")==0)

{

printf("\a\n\tERROR: STUDENT\_ID FIELD CANNOT BE BLANK");

flag = 0;

continue;

}

if(start!=NULL)

{

while(unique\_id\_check()!=1)

{

printf("\n\nENTER THE STUDENT ID NUMBER = ");

gets(temp\_id);

fflush(stdin);

flag = special\_character\_check(temp\_id);

}

}

}while(flag!=1);

uppercase(temp\_id);

}

// Automated stockroom modules starts from here being implemented suing the concept of stacks

void stack\_start(){

stack\_menu();

do{

printf("\n\t ENTER THE TOTAL NUMBER OF RETAILERS = ");

// Scan the total number of retailers

number = number\_check();

if(number>=100)

{

printf("\a\n\t ERROR: NUBER OF PRODUCTS ARE LIMITED TO 100\n");

temp=0;

}

else

{

temp=1;

}

}while(temp!=1);

id\_scanning();

stack\_display();

getch();

}

void stack\_menu()

{

system("cls");

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

printf("\n\t\t \* AUTOMATED\_STOCKROOM \*");

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

}

void id\_scanning()

{

newretail = (struct retailer \*)malloc (number \* sizeof(struct retailer));

if(newretail == NULL)

{

printf("\a\n\tMEMORY NOT AVAILABLE");

}

else{

for(level=0;level<number;level++)

{

do{

fflush(stdin);

printf("\n PRODUCT [%d] ID = ",level+1);

gets(temp\_name);

temp= special\_character\_check(temp\_name);

if (stricmp(temp\_name,"")==0) // Checking whether the string eneterd is NULL string

{

printf("\a\n\t ERROR: PRODUCT MUST HAVE A ID\n");

temp=0;

}

if(level!=0) // this will get executed when there will be more than one product\_id scanned

{

j = level;

while(j>=0)

{

if(stricmp(newretail[j].product\_id,temp\_name)==0) // Compare the id eneterd with the existing ones

{

printf("\a\n\t ERROR:PRODUCT MUST HAVE A UNIQUE ID\n");

temp=0;

}

j--;

}

}

}while(temp!=1);

uppercase(temp\_name);

strcpy(newretail[level].product\_id,temp\_name);

do{

fflush(stdin);

printf("\n BIN [%d] ID = ",level+1);

// Scanning the bin\_id

choice=number\_check();

if(choice>=1&&choice<=50)

{

temp=1;

}

else{

printf("\tERROR:BIN\_ID SHOULD BE BETWEEN 1-50\n");

temp=0;

}

}while(temp!=1);

newretail[level].bin\_id = choice;

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

for(i=0;i<5;i++)

{

/\*initializing all array elements

of top to -1 that represents the initially

there is not element in any of the bins\*/

top[i]=-1;

}

j=0;

do{

temp=0;

/\* Copying the product id and bin id respectively

in the stacker five at a time or

until the all the retailers being removed\*/

while(temp<5 && j!=number)

{

strcpy(stack\_temp[temp].product\_id,newretail[j].product\_id);

stack\_temp[temp].bin\_id = newretail[j].bin\_id;

j++;

temp++;

}

for(i = temp-1;i>=0;i--)

{

if(top[stack\_temp[i].bin\_id] == top\_of\_stack-1)

{

strcpy(bin[stack\_temp[i].bin\_id][4].product\_name,stack\_temp[i].product\_id);

}

else

{

strcpy(bin[stack\_temp[i].bin\_id][++top[stack\_temp[i].bin\_id]].product\_name,stack\_temp[i].product\_id);

}

}

}while(j!=number);

printf("\n\n PRESS ENTER TO DISPLAY THE STOCKROOM........ ");

while((int)getch()!=13);

}

}

void stack\_display()

{

system("cls");

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

printf("\n\t\t\t\* STOCKROOM INSIDE \*");

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

printf("\n");

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

printf("\t\t BIN\_ID\tPRODUCT\_ID");

for(i=0;i<top\_of\_stack;i++)

{

// Copying the top of particular element

//into temp variable for poping purpose

temp=top[i];

while(temp != -1)

{

printf("\n\t\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|");

printf("\n\t\t%5d",i);

printf(" \t %s ",bin[i][temp].product\_name); // Print the product\_id stored in array bin

temp--;

}

}

printf("\n\t\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|");

printf("\n\n\n=> PRESS 1 TO ADD RETAILERS AGAIN IN STOCKROOM\n\n");

printf("=> PRESS 2 TO GO TO MAIN MENU AGAIN");

do{

printf("\n\n\tENTER YOUR CHOICE = ");

fflush(stdin);

ch = getch();

while((int)ch==13||(int)ch==8||(int)ch==32||(int)ch==9) // if enter, space, backspace or tab will be pressed nothing would get displayed

{

ch=getch();

}

putch(ch);

while((int)getch()!=13){}

if(ch=='1'||ch=='2')

{

temp=1;

}

else {

printf("\a\n\n\t\tERROR:ENTER A VALID CHOICE");

temp=0;

}

}while(temp!=1);

if(ch=='1')

{

stack\_start();

}

else

{

main();

}

}

void clock\_copy()

{

int n;

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n\t\t\tGREEN UNIVERSITY OF BANGLADESH\n");

printf("\n\t\t\t\tBy SOHIDUL ISLAM\n");

printf("\n\t\t\t\t ID: 152002005\n");

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

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

printf("\n\t\t \* \"STUDENT MANAGEMENT SYSTEM\" \*");

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

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

wait(2);

printf("\n\n MODULE LECTURER\n");

printf("\n\_\_\_\_ Ashaduzzaman LECTURERAR OF GREEN UNIVERSITY\_\_\_\_");

wait(2);

printf("\n\n\t\tPRESS ENTER TO CONTINUE....");

while(getch()!=13){}

}