Programming
Practice Lab

Tathagata Sur

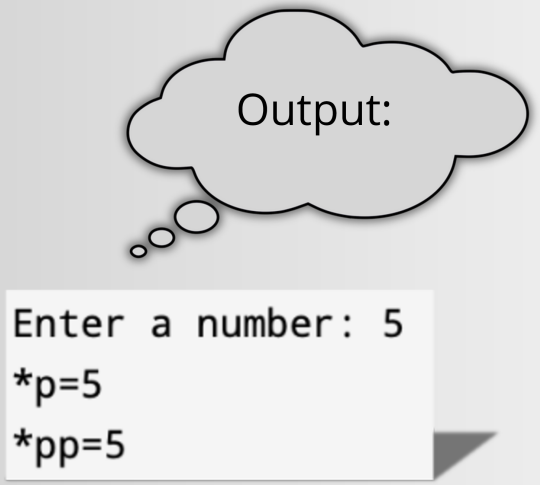
BCSE UG-II
ROLL-002310501030
A1

Assignment 1

8 questions

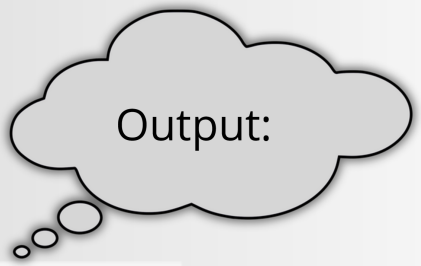
Q1.Write a program that will have an integer variable and a pointer (say, p) pointing to it. Also have a pointer to pointer pointing to p. Take the value for the integer variable and print it using p, and pp.

```
#include <stdio.h>
int main(){
int n;
printf("Enter a number: ");
scanf("%d",&n);
int *p;
p=&n;
int **pp=&p;
printf("*p=%d\n*pp=%d\n",*p,**pp);
return 0;
}
```



Q2. Implement a one dimensional array of integers where array size of the array will be provided during runtime. Accept the value for the elements and print those using pointers.

```
#include <stdio.h>
int main(){
    int n;
printf("Enter an number\n");
scanf("%d",&n);
int a[n];
printf("Enter the %d elements of
array\n",n);
for(int i=0;i<n;i++)
scanf("%d",&a[i]);

printf("The array is  [");
for(int i=0;i<n-1;i++)
printf("%d,",a[i]);
printf("%d]\n",a[n-1]);
return 0;
}
```

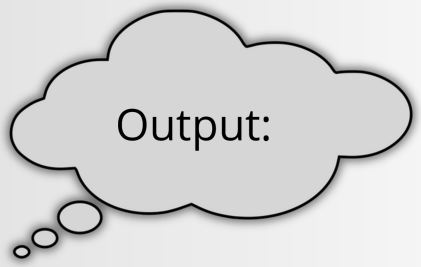



```
Enter an number
5
Enter the 5 elements of array
1 2 3 4 5
The array is  [1,2,3,4,5]
```

Q3.Implement a two dimensional array of integers using:

a) array of pointers

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int r,c;
printf("Enter the number of r\n");
scanf("%d",&r);
printf("Enter the number of columns\n");
scanf("%d",&c);
int **b=(int **)malloc(r*sizeof(int *));
for(int i=0;i<r;i++){
*(b+i)=(int *)malloc(c*sizeof(int));
}
printf("Enter array elements in row major fashion\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
scanf("%d",(*(b+i)+j));
}
}
printf("The array is\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
printf("%d\t",*(*(b+i)+j));
}
printf("\n");
}
free(b);
}
```

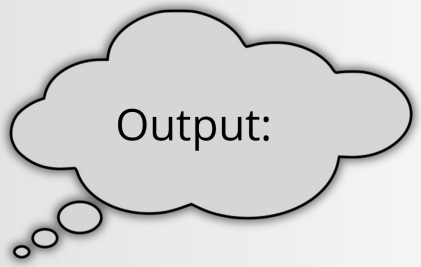



```
Enter the number of r
2
Enter the number of columns
2
Enter array elements in row major fashion
1 2 3 4
The array is
1   2
3   4
```

Q3.Implement a two dimensional array of integers using:

b) pointer to pointer (with two malloc statements and again with one malloc statement

```
#include<stdio.h>
#include<stdlib.h>
void main() {
int r,c;
printf("Enter the number of rows\n");
scanf("%d",&r);
printf("Enter the number of columns\n");
scanf("%d",&c);

int **b = (int **)malloc(r* sizeof(int *));

for (int i = 0; i < r; i++) {
b[i] = (int *)malloc(c* sizeof(int));
}
printf("Enter array elements in row major fashion\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
scanf("%d",(*(b+i)+j));
}
}
printf("The array is\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
printf("%d\t",*(*(b+i)+j));
}printf("\n");
}
for (int i = 0; i<r; i++) {
free(b[i]);
}
free(b);
}
```




```
Enter the number of rows
2
Enter the number of columns
2
Enter array elements in row major fashion
5 6 7 8
The array is
5   6
7   8
```

Q3.Implement a two dimensional array of integers using:

c) pointer to an array. Accept the value for the elements and print those

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int r,c;
printf("Enter the number of rows\n");
scanf("%d",&r);
printf("Enter the number of columns\n");
scanf("%d",&c);
printf("Enter array elements in row major fashion\n");
int* b[r][c];
for(int i=0;i<r;i++){
for (int j=0;j<c;j++){
scanf("%d",&b[i][j]);
}
}
printf("The array is\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
printf("%d\t",b[i][j]);
}
printf("\n");
}

}
```

Output:

```
Enter the number of rows
3
Enter the number of columns
3
Enter array elements in row major fashion
1 2 3 4 5 6 7 8 9
The array is
1   2   3
4   5   6
7   8   9
```

Q4.Implement the programs in Q.2 and 3 breaking it into functions for getting the dimensions from user, dynamic memory allocation, accepting the values and printing the values

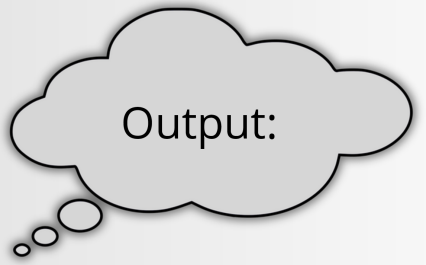
```
#include<stdio.h>
#include<stdlib.h>
int rowacc(){
int r;
printf("Enter the number of rows\n");
scanf("%d",&r);
return r;
}
int colacc(){
int c;
printf("Enter the number of columns\n");
scanf("%d",&c);
return c;
}
void allo(int **b,int r,int c){
for(int i=0;i<r;i++){
*(b+i)=(int *)malloc(c*sizeof(int));
}
}
void accept(int **b,int r,int c){
printf("Enter array elements in row major fashion\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
scanf("%d",(*(b+i)+j));
}
}
}
void print(int **b,int r,int c){
printf("The array is\n");
for(int i=0;i<r;i++){
for(int j=0;j<c;j++){
printf("%d\t",*(*(b+i)+j));
}
printf("\n");
}
}
void main() {
int r,c;
r=rowacc();
c=colacc();
int **b=(int **)malloc(r*sizeof(int *));
allo(b,r,c);
accept(b,r,c);
print(b,r,c);
free(b);
}
```

Output:

```
Enter the number of rows
2
Enter the number of columns
2
Enter array elements in row major fashion
10 20 30 40
The array is
10  20
30  40
```

Q5.Store name and age of number of persons (number provided at run time). Collect the data and display data in the ascending order of age. Implement without using structure. Write functions for memory allocation of the list, sorting and display of data.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void mem(char ***names, int **ages, int n) {
*names = (char **)malloc(n * sizeof(char *));
for (int i = 0; i < n; i++) {
(*names)[i] = (char *)malloc(100 * sizeof(char));
}
*ages = (int *)malloc(n * sizeof(int));
}
void sorting(char **names, int *ages, int n) {
for (int i = 0; i < n - 1; i++) {
for (int j = i + 1; j < n; j++) {
if (ages[i] > ages[j]) {
int tempAge = ages[i];
ages[i] = ages[j];
ages[j] = tempAge;
char tempName[100];
strcpy(tempName, names[i]);
strcpy(names[i], names[j]);
strcpy(names[j], tempName);
} } } }

void displayData(char **names, int *ages, int n) {
printf("Name\tAge\n");
for (int i = 0; i < n; i++) {
printf("%s\t%d\n", names[i], ages[i]);
} }
void main() {
int n;
printf("Enter the number of persons: ");
scanf("%d", &n);
char **names;
int *ages;
mem(&names, &ages, n);
for (int i = 0; i < n; i++) {
printf("Enter name and age of person(Age should be positive) %d: ", i + 1);
scanf("%s %d", names[i], &ages[i]);
}
for (int i = 0; i < n; i++) {
ages[i]=abs(ages[i]);
}
sorting(names, ages, n);
displayData(names, ages, n);

for (int i = 0; i < n; i++) {
free(names[i]);
}
free(names);
free(ages);
}
```




```
Enter the number of persons: 2
Enter name and age of person(Age should be positive) 1: Anup
70
Enter name and age of person(Age should be positive) 2: Rahul
60
Name    Age
Rahul   60
Anup    70
```

Q6. Implement Q.5 using structure

```
#include <stdio.h>
#include <stdlib.h>
struct student{
 char name[100];
int age;
};
void mysort(struct student arr[],  int n) {
int cnt;
for (int i = 0; i < n - 1; i++) {
cnt = 0;
for (int j = 0; j < n - i - 1; j++) {
 if (arr[j].age > arr[j + 1].age) {
 struct student temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
cnt = 1;
}
}
if (cnt == 0) {
break;
}
}
}
int main() {
int m;
printf("Enter total number of students:\t");
scanf("%d",&m);
struct student* finalList;
finalList=(struct student*)malloc(m*sizeof(struct student));
for (int i=0; i<m; i++){
 printf("enter age:\t");
scanf("%d",&finalList[i].age);
getchar();
printf("enter name:\t");
fgets(finalList[i].name, 100, stdin);
}
mysort(finalList, m) ;
for (int i=0; i<m; i++){
printf("name:%sage %d\n", finalList[i].name, finalList[i].age);
}
return 0;
}
```

Output:

```
Enter total number of students: 2
enter age:  40
enter name: Alice
enter age:  50
enter name: Bob
name:Alice
age 40
name:Bob
age 50
```

Q7.Maintain a list to store roll, name and score of students. As and when required student record may be added or deleted. Also, the list has to be displayed. Design suitable functions for different operations

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
int roll;
char name[100];
float score;
} Student;
void addStudent(Student **list, int *size, int roll, const char *name, float
score) {
*list = realloc(*list, (*size + 1) * sizeof(Student));
(*list)[*size].roll = roll;
strcpy((*list)[*size].name, name);
(*list)[*size].score = score;
(*size)++;
}
void deleteStudent(Student *list, int *size, int roll) {
int flag = 0;
for (int i = 0; i < *size; i++) {
if (list[i].roll == roll) {
flag = 1;
for (int j = i; j < *size - 1; j++) {
list[j] = list[j + 1];
}
(*size)--;
list = realloc(list, (*size) * sizeof(Student));
break;
} }
if (!flag)
printf("Student with roll number %d not found.\n", roll);
}
void displayStudents(const Student *list, int size) {
if (size == 0) {
printf("No students in the list.\n");
return; }
printf("Roll\tName\t\tScore\n");
for (int i = 0; i < size; i++) {
printf("%d\t%s\t\t% f\n", list[i].roll, list[i].name, list[i].score);
}
}
```

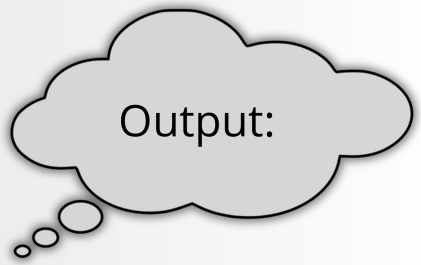
```
void main() {
Student *studentList = NULL;
int size = 0, choice, roll;
float score;
char name[100];
do {
printf("\nMenu:\n");
printf("1. Add Student\n");
printf("2. Delete Student\n");
printf("3. Display Students\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter roll number: ");
scanf("%d", &roll);
printf("Enter name: ");
scanf("%s", name);
printf("Enter score: ");
scanf("%f", &score);
addStudent(&studentList, &size, roll, name, score);
break;
case 2:
printf("Enter roll number of student to delete: ");
scanf("%d", &roll);
deleteStudent(studentList, &size, roll);
break;
case 3:
displayStudents(studentList, size);
break;
case 4:
printf("Exiting...\n");
break;
default:
printf("Invalid choice!\n");
}
} while (choice != 4);
free(studentList);
}
```

Output:

```
Menu:
1. Add Student
2. Delete Student
3. Display Students
4. Exit
Enter your choice: 1
Enter roll number: 20
Enter name: Alice
Enter score: 60

Menu:
1. Add Student
2. Delete Student
3. Display Students
4. Exit
Enter your choice: 3
Roll    Name        Score
20  Alice       60.000000

Menu:
1. Add Student
2. Delete Student
3. Display Students
4. Exit
Enter your choice: 2
Enter roll number of student to

Menu:
1. Add Student
2. Delete Student
3. Display Students
4. Exit
Enter your choice: 3
No students in the list.

Menu:
1. Add Student
2. Delete Student
3. Display Students
4. Exit
Enter your choice:
```

Q8.Consider an array that stores roll, name, and score of number of Students. Develop a function to sort the array.
User of sort() will develop the comparison function for sorting on roll/score and ascending or descending order and reuse the same sort() function.

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

struct Student {
int roll;
char name[50];
int score;
};
void sort(struct Student arr[], int n, int (*cmp)(struct Student, struct Student)) {
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (cmp(arr[j], arr[j + 1]) > 0) {
struct Student temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
} } } }

int compareByRollAsc(struct Student a, struct Student b) {
return a.roll - b.roll;
}
int compareByRollDesc(struct Student a, struct Student b) {
return b.roll - a.roll;
}
int compareByScoreAsc(struct Student a, struct Student b) {
return a.score - b.score;
}
int compareByScoreDesc(struct Student a, struct Student b) {
return b.score - a.score;
}
void main() {
int n;
printf("Enter total number of Students:\t");
scanf("%d",&n);
struct Student* Students;
Students=(struct Student*)malloc(n*sizeof(struct Student));
for (int i=0; i<n; i++){
printf("enter roll number:\t");
scanf("%d",&Students[i].roll);
getchar();
printf("enter name:\t");
fgets(Students[i].name, 100, stdin);
printf("enter total score:\t");
scanf("%d",&Students[i].score);
getchar();
}
printf("Original input:\n");
for (int i = 0; i < n; i++)
printf("Roll: %d, Name: %s Score: %d\n", Students[i].roll, Students[i].name, Students[i].score);

printf("Enter your choice:1 to sort by roll,2 to sort by score\n");
int c1,c2;
scanf("%d",&c1);
if(c1==1) {
printf("Enter your choice:1 to sort roll no. in Ascending order and 2 to sort in descending
order\n");
scanf("%d",&c2);
if(c2==1) {
sort(Students, n, compareByRollAsc);
printf("\nSorted by roll (Ascending):\n");
for (int i = 0; i < n; i++) {
printf("Roll: %d, Name: %sScore: %d\n", Students[i].roll, Students[i].name, Students[i].score);
} }
else {
sort(Students, n, compareByRollDesc);
printf("\nSorted by roll (Descending):\n");
for (int i = 0; i < n; i++) {
printf("Roll: %d, Name: %sScore: %d\n", Students[i].roll, Students[i].name, Students[i].score);
} } }
else if(n==2)
{
printf("Enter your choice:1 to sort score in Ascending order and 2 to sort in descending
order\n");
scanf("%d",&c2);
if(c2==1) {
sort(Students, n, compareByScoreAsc);
printf("\nSorted by score (Ascending):\n");
for (int i = 0; i < n; i++) {
printf("Roll: %d, Name: %sScore: %d\n", Students[i].roll, Students[i].name, Students[i].score);
} }
else {
sort(Students, n, compareByScoreDesc);
printf("\nSorted by score (Descending):\n");
for (int i = 0; i < n; i++)
printf("Roll: %d, Name: %sScore: %d\n", Students[i].roll, Students[i].name, Students[i].score);

else printf("Invalid choice\n");
}
```




```
Enter total number of Students: 2
enter roll number:  30
enter name: Alice
enter total score:  75
enter roll number:  40
enter name: Bob
enter total score:  80
Original input:
Roll: 30, Name: Alice
 Score: 75
Roll: 40, Name: Bob
 Score: 80
Enter your choice:1 to sort by roll,2 to sort by score
2
Enter your choice:1 to sort score in Ascending order and 2 to sort in descending order
2

Sorted by score (Descending):
Roll: 40, Name: Bob
Score: 80
Roll: 30, Name: Alice
Score: 75
```

Assignment 2

1 question

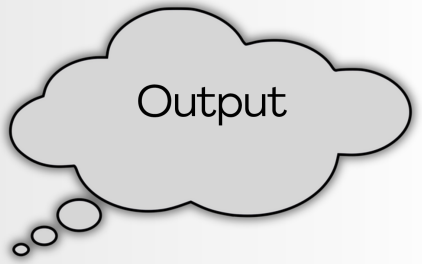
Q1.Write a program to store student information in a file and to do the following operations. Information includes roll, name, and score in five subjects. Use may like to add record (ensure roll number is unique), display all records showing roll, name and total score. User may search for the record against a roll. User may edit the details a record. User may delete record. Deletion may be logical (by some means indicate it is an invalid record and to be avoided in processing) and physical (file will not have the record).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FILENAME "students.dat"

typedef struct {
int roll;
char name[50];
int scores[5];
int is_deleted; // Logical deletion flag (0: active, 1: deleted)
} Student;

void addRecord();
void displayRecords();
void searchRecord();
void editRecord();
void logicalDelete();
void physicalDelete();

int main() {
int choice;
while (1) {
printf("\n1. Add Record\n2. Display All Records\n3. Search Record\n4. Edit Record\n5.
Logical Delete\n6. Physical Delete\n7. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1: addRecord(); break;
case 2: displayRecords(); break;
case 3: searchRecord(); break;
case 4: editRecord(); break;
case 5: logicalDelete(); break;
case 6: physicalDelete(); break;
case 7: exit(0);
default: printf("Invalid choice!\n");
}
}
return 0;
}
```

```
void addRecord() {
FILE *fp = fopen(FILENAME, "ab");
if (fp == NULL) {
printf("Error opening file.\n");
return;
}
Student student;
printf("Enter Roll Number: ");
scanf("%d", &student.roll);

FILE *temp_fp = fopen(FILENAME, "rb");
Student temp;
while (fread(&temp, sizeof(Student), 1, temp_fp)) {
if (temp.roll == student.roll && temp.is_deleted == 0) {
printf("Record with roll number %d already exists.\n", student.roll);
fclose(temp_fp);
fclose(fp);
return;
}
}
fclose(temp_fp);
printf("Enter Name: ");
scanf("%s", student.name);
printf("Enter scores for 5 subjects: ");
for (int i = 0; i < 5; i++) {
scanf("%d", &student.scores[i]);
}
student.is_deleted = 0;
fwrite(&student, sizeof(Student), 1, fp);
fclose(fp);
printf("Record added successfully.\n");
}
void displayRecords() {
FILE *fp = fopen(FILENAME, "rb");
if (fp == NULL) {
printf("Error opening file.\n");
}
```

```
Student student;
printf("Roll\tName\tTotal Score\n");
while (fread(&student, sizeof(Student), 1, fp)) {
if (student.is_deleted == 0) {  // Skip logically deleted records
int total = 0;
for (int i = 0; i < 5; i++) {
total += student.scores[i];
}
printf("%d\t%s\t%d\n", student.roll, student.name, total);
}
}
fclose(fp);
}
void searchRecord() {
FILE *fp = fopen(FILENAME, "rb");
if (fp == NULL) {
printf("Error opening file.\n");
}

int roll;
printf("Enter Roll Number to search: ");
scanf("%d", &roll);

Student student;
int found = 0;
while (fread(&student, sizeof(Student), 1, fp)) {
if (student.roll == roll && student.is_deleted == 0) {
printf("Record found!\n");
printf("Roll: %d\nName: %s\nScores: ", student.roll, student.name);
for (int i = 0; i < 5; i++) {
printf("%d ", student.scores[i]);
}
printf("\n");
found = 1;
break;
}
}
if (!found) {
printf("Record not found.\n");
}
fclose(fp);
}
```

```
void editRecord() {
FILE *fp = fopen(FILENAME, "rb+");
if (fp == NULL) {
printf("Error opening file.\n");
return;
}

int roll;
printf("Enter Roll Number to edit: ");
scanf("%d", &roll);

Student student;
int found = 0;
long pos;
while ((pos = ftell(fp)), fread(&student, sizeof(Student), 1, fp)) {
if (student.roll == roll && student.is_deleted == 0) {
printf("Editing Record with Roll: %d\n", roll);
printf("Enter new Name: ");
scanf("%s", student.name);
printf("Enter new scores for 5 subjects: ");
for (int i = 0; i < 5; i++) {
scanf("%d", &student.scores[i]);
}

fseek(fp, pos, SEEK_SET);
fwrite(&student, sizeof(Student), 1, fp);
printf("Record updated successfully.\n");
found = 1;
break;
}
}
if (!found) {
printf("Record not found.\n");
}
fclose(fp);
}

void logicalDelete() {
FILE *fp = fopen(FILENAME, "rb+");
if (fp == NULL) {
printf("Error opening file.\n");
return;
}

int roll;
printf("Enter Roll Number to logically delete: ");
scanf("%d", &roll);

Student student;
int found = 0;
long pos;
while ((pos = ftell(fp)), fread(&student, sizeof(Student), 1, fp)) {
if (student.roll == roll && student.is_deleted == 0) {
student.is_deleted = 1;
fseek(fp, pos, SEEK_SET);
fwrite(&student, sizeof(Student), 1, fp);
printf("Record logically deleted.\n");
found = 1;
break;
}
}
if (!found) {
printf("Record not found.\n");
}
fclose(fp);
}

void physicalDelete() {
FILE *fp = fopen(FILENAME, "rb");
FILE *temp_fp = fopen("temp.dat", "wb");

if (fp == NULL || temp_fp == NULL) {
printf("Error opening file.\n");
return;
}

int roll;
printf("Enter Roll Number to physically delete: ");
scanf("%d", &roll);

Student student;
int found = 0;
while (fread(&student, sizeof(Student), 1, fp)) {
if (student.roll == roll && student.is_deleted == 0) {
printf("Record physically deleted.\n");
found = 1;
} else {
fwrite(&student, sizeof(Student), 1, temp_fp);
}
}

if (!found) {
printf("Record not found.\n");
}

fclose(fp);
fclose(temp_fp);
remove(FILENAME);
rename("temp.dat", FILENAME);
}
```




```
1. Add Record
2. Display All Records
3. Search Record
4. Edit Record
5. Logical Delete
6. Physical Delete
7. Exit
```

Assignment 3

Q1. Write a function swap (a, b) to interchange the values of two variables. Do not use pointers.

```
#include <iostream>

using namespace std;

void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
    cout << "After swapping: a = " << a << ", b = " << b << endl;
}

int main() {
    int a = 5, b = 10;
    cout << "Before swapping: a = " << a << ", b = " << b << endl;
    swap(a, b);
    return 0;
}
```

```
Before swapping: a = 5, b = 10
After swapping: a = 10, b = 5


=== Code Execution Successful ===
```

Q2. Write a function max (a, b) that will return the reference of larger value. Store the returned information to x where x is a i) variable of type a or b, ii) variable referring to type of a or b. In both the cases modify x. Check also the values of a and b.

```
#include <iostream>
#include <algorithm>
using namespace std;


void change(int& x){
   cout<< "Enter the values of new value of x1 :   "<<endl;
   cin>>x;
}
int main() {
    int a;
   int b;
   cout<< "Enter the values of a and b :   "<<endl;
   cin>>a>>b;
   cout<<"the values of a and b : "<<a<<","<<b<<endl;
   //part i
   int x1= (a>=b)?a:b;
   cout<< "the value of x1: "<<x1<<endl;
   change (x1);
   cout<<"the value of x1: "<<x1<<endl;
   cout<<"the values of a and b :   "<<a<<" " <<b<<endl;
   //part ii
   int* x2= (a>=b)?&a:&b;
    cout<< "the value referred to by  x2: "<<*x2<<endl;
```

```
    change (*x2);
    cout<< "the value referred to by x2: "<<*x2<<endl;
    cout<<"the values of a and b : "<<a<<","<<b<<endl;
    return 0;
}
```

```
Enter the values of a and b :
5
10
the values of a and b : 5,10
the value of x1: 10
Enter the values of new value of x1 :
15
the value of x1: 15
the values of a and b :   5 10
the value referred to by  x2: 10
Enter the values of new value of x1 :
20
the value referred to by x2: 20
the values of a and b : 5,20


=== Code Execution Successful ===
```

Q3. Write a function that will have income and tax rate as arguments and will return tax amount. In case tax rate is not provided it will be automatically taken as 10%. Call it with and without tax rate.

```
#include <iostream>
using namespace std;

double calculateTax(double income, double taxRate = 10.0) {
    return income * (taxRate / 100);
}
```

```
int main() {
    double income = 50000;
    // Calling the function with a tax rate
    double taxWithRate = calculateTax(income, 15.0);
    cout << "Tax with provided rate (15%): " << taxWithRate << endl;
    // Calling the function without providing a tax rate (defaults to 10%)
    double taxWithoutRate = calculateTax(income);
    cout << "Tax with default rate (10%): " << taxWithoutRate << endl;

    return 0;
}
```

```
Tax with provided rate (15%): 7500
Tax with default rate (10%): 5000


=== Code Execution Successful ===
```

Q4. Write a function void f(int) that prints “inside f(int)”. Call the function with actual argument of type: i) int, ii) char, iii) float and iv) double. Add one more function f(float) that prints “inside f(float)”. Repeat the calls again and observe the outcomes.

```
#include <iostream>
using namespace std;
```

```
void f(int) {
cout << "inside f(int)" << endl;
}

void f(float) {
cout << "inside f(float)" << endl;
}

int main() {
int i = 10;
char c = 'A';
float f1 = 10.5f;
double d = 20.25;
f(i);
f(c);
f(f1);
f(d);
// We have to comment out one of the functions while making these calls else there occurs an error.

return 0;
}
```

```
ERROR!
/tmp/AU3yumZdv0.cpp: In function 'int main()':
/tmp/AU3yumZdv0.cpp:20:2: error: call of overloaded 'f(double&)' is ambiguous
   20 | f(d);
      | ~^~~
/tmp/AU3yumZdv0.cpp:4:6: note: candidate: 'void f(int)'
    4 | void f(int) {
      |      ^
/tmp/AU3yumZdv0.cpp:8:6: note: candidate: 'void f(float)'
    8 | void f(float) {
      |      ^


=== Code Exited With Errors ===
```

Q5. Define functions f(int, int) and f (char, int). Call the functions with arguments of type (int, char), (char,char) and (float, float). Observe and analyze the outcome.

```
#include <iostream>

using namespace std;


void f(int, int) {

cout << "inside f(int, int)" << endl;

}

void f(char, int) {

cout << "inside f(char, int)" << endl;

}


int main() {

int i = 10;

char c = 'A';

float f1 = 10.5f;
```

```
f(i, c);  // Calls f(int, int) due to implicit conversion of char to int
f(c, c);  // Calls f(char, int) due to implicit conversion of the second char to int

/* Calling with (float, float) causes an error because there is no matching function for (float, float)
and explicit type conversion does not occur automatically*/
// f(f1, f1);
return 0;
}
```

```
inside f(int, int)
inside f(char, int)


=== Code Execution Successful ===
```

Q6. Define a structure student with roll and score as attributes and with two member functions to take input and to show the data. Use the member functions to take data for a structure variable and to show. Write global function i) to modify score and ii) to show the data again.

```
#include <iostream>
using namespace std;

struct student {
    int roll;
    float score;

    // Member function to take input
    void takeInput() {
        cout << "Enter roll number: ";
        cin >> roll;
        cout << "Enter score: ";
        cin >> score;
    }

   // Member function to show data
    void showData() const {
        cout << "Roll: " << roll << ", Score: " << score << endl;
    }
};

// Global function to modify the score
void modifyScore(student &s, int newScore) {
    s.score = newScore;
}

// Global function to show data again
```

```
void showDataAgain(const student &s) {
    cout << "Roll: " << s.roll << ", Score (after modification): " << s.score << endl;
}

int main() {
   student s1;
   s1.takeInput();
   s1.showData();
    int newScore;
    cout << "Enter new score for modification: ";
    cin >> newScore;
    modifyScore(s1, newScore);
    showDataAgain(s1);

    return 0;
}
```

```
Enter roll number: 30
Enter score: 95
Roll: 30, Score: 95
Enter new score for modification: 100
Roll: 30, Score (after modification): 100


=== Code Execution Successful ===
```

Q7. Design a class TIME which stores hour, minute and second. The class should have the methods to support the following:

- User may give the time value in 24-hour format.

- User may give the time value in AM/PM format

- Display the time in 24-hour format.

- Display the time in AM/PM format.

- User may like to add minute with a time value.

```
#include <bits/stdc++.h>
using namespace std;
class TIME {
private:
int hour;
int minute;
int second;
int flag;
int d;

void normalize() {
if (second >= 60) {
minute += second / 60;
second %= 60;
}
if (minute >= 60) {
hour += minute / 60;
minute %= 60;
}
hour %= 24;
```

```
}

public:
TIME(int h = 0, int m = 0, int s = 0,int f=0,int ind=0) : hour(h),minute(m),second(s),flag(f),d(ind) {
   if(flag==2)
   hour+=((d==2)?12:0);
   normalize();
}
void display24HourFormat() const {
cout << setfill('0') << setw(2) << hour << ":"
<< setw(2) << minute << ":"
<< setw(2) << second << endl;
}

void displayAMPMFormat() const {
int displayHour = hour % 12;
if (displayHour == 0) displayHour = 12;
string period = (hour >= 12) ? "PM" : "AM";
cout << displayHour << ":"
<< setfill('0') << setw(2) << minute << ":"
<< setw(2) << second << " " << period << endl;
}
void addMinutes(int m) {
minute += m;
normalize();
}
};
```

```
int main() {
int ch,flag=2,d=0;
int h,m,s;
cout << "Enter your choice,1 for 24-Hour Format and 2 for AM/PM format:"<<endl;
cin>>ch;
if(ch==1)
flag=1;
cout << "Enter hour->";
cin>>h;
cout << "Enter minutes->";
cin>>m;
cout << "Enter seconds->";
cin>>s;
if (flag==2)
{
   cout << "1 for AM,2 for PM ->";
   cin>>d;
}
TIME t1(h, m, s, flag, d);
cout << "24-Hour Format: ";
t1.display24HourFormat();
cout << "AM/PM Format: ";
t1.displayAMPMFormat();
cout << "Enter minutes to be added: ";
cin>>m;
t1.addMinutes(m);
```

```
cout << "After adding " << m << " minutes:" << endl;
cout << "24-Hour Format: ";
t1.display24HourFormat();
cout << "AM/PM Format: ";
t1.displayAMPMFormat();
return 0;
}
```

```
Enter your choice,1 for 24-Hour Format and 2 for AM/PM format:
1
Enter hour->19
Enter minutes->50
Enter seconds->56
24-Hour Format: 19:50:56
AM/PM Format: 7:50:56 PM
Enter minutes to be added: 10
After adding 10 minutes:
24-Hour Format: 20:00:56
AM/PM Format: 8:00:56 PM


=== Code Execution Successful ===
```

Q8. Create a STACK class with operation for initialization, push and pop. Support for checking underflow and overflow condition are also to be provided.

```
#include <iostream>
using namespace std;

class STACK {
private:
```

```
int* arr;
int top;
int capacity;

public:
STACK(int size) {           // Constructor to initialize the stack
capacity = size;
arr = new int[capacity];
top = -1;
}

~STACK() {                  // Destructor to clean up allocated memory
delete[] arr;
}
bool isFull() const {       // Checking if the stack is full
return top == capacity - 1;
}
bool isEmpty() const {
return top == -1;
}
void push(int value) {
if (isFull()) {
cout << "Stack overflow: Cannot push " << value << ". Stack is full." << endl;
} else {
arr[++top] = value;
cout << "Pushed " << value << " onto the stack." << endl;
}
```

```
}
void pop() {
if (isEmpty()) {
cout << "Stack underflow: Cannot pop. Stack is empty." << endl;
} else {
cout << "Popped " << arr[top--] << " from the stack." << endl;
}
}
void display() const {
if (isEmpty()) {
cout << "Stack is empty." << endl;
} else {
cout << "Stack contents: [ ";
for (int i = top; i > 0; i--) {
cout << arr[i] << " ";
}
cout << arr[0]<<" ]"<<endl;
}
}
};

int main() {
   int n;
   cout<<"Enter max capacity of stack: ";
   cin>>n;
   STACK s(n);
   int ch=1;
```

```
   while(true)
   {
  cout<<"Enter your choice:”<<endl<<”1 for pushing element into the array”<<endl<<”2 for popping from the array””<<endl<<”3 for displaying the array”<<endl<<”4 for exiting"<endl;
      cin>>ch;
      if(ch==1){
         cout<<"Enter element to be pushed into stack: ";
         cin>>n;
         s.push(n);
      }
      else if(ch==2)    s.pop();
      else if(ch==3)    s.display();
      else break;
   }
return 0;
}
```

```
Enter max capacity of stack: 2
Enter your choice:
1 for pushing element into the array
2 for popping from the array
3 for displaying the array
4 for exiting
2
Stack underflow: Cannot pop. Stack is empty.
Enter your choice:
1 for pushing element into the array
2 for popping from the array
3 for displaying the array
4 for exiting
1
Enter element to be pushed into stack: 4
Pushed 4 onto the stack.
Enter your choice:
1 for pushing element into the array
2 for popping from the array
3 for displaying the array
4 for exiting
1
Enter element to be pushed into stack: 5
Pushed 5 onto the stack.
Enter your choice:
1 for pushing element into the array
2 for popping from the array
3 for displaying the array
4 for exiting
3
Stack contents: [ 5 4 ]
Enter your choice:
1 for pushing element into the array
2 for popping from the array
3 for displaying the array
4 for exiting
1
Enter element to be pushed into stack: 3
Stack overflow: Cannot push 3. Stack is full.
Enter your choice:
1 for pushing element into the array
2 for popping from the array
3 for displaying the array
4 for exiting
4


=== Code Execution Successful ===
```

Q9. Create an APPLICANT class with application id (auto generated as last id +1), name and score. Support must be there to receive applicant data, show applicant

details and to find out number of applicants.

```
#include <iostream>
#include <string>
using namespace std;
class APPLICANT {
private:
static int lastId;     // Static variable to keep track of the last ID
int applicationId;
string name;
float score;

public:
APPLICANT() {
applicationId = ++lastId;
}

void receiveData() {
cout << "Enter name: ";
cin.ignore();          // To clear the newline character from the input buffer
getline(cin, name);
cout << "Enter score: ";
cin >> score;
}

void showDetails() const {
cout << "Application ID: " << applicationId << endl;
```

```
cout << "Name: " << name << endl;
cout << "Score: " << score << endl;
}

static int getNumberOfApplicants() {
return lastId;
    }  // Static method to find out the number of applicants
};

int APPLICANT::lastId = 0;        // Initialize the static variable

int main() {
   cout << "Enter choice: 1 for adding applicants and 2 for exiting:-> " << endl;
   int ch,count=0;
   cin>>ch;
   while(ch!=2)
   {
APPLICANT applicant;
count++;
cout << "Enter details for applicant "<<count<<": "<< endl;
applicant.receiveData();
cout << "Details of applicant "<<count<<": "<<endl;
applicant.showDetails();
 cout << "Enter choice: 1 for adding applicants and 2 for exiting:-> " << endl;
 cin>>ch;
}
cout << "Total number of applicants: " << count<< endl;
```

return 0;

}

```
Enter choice: 1 for adding applicants and 2 for exiting:->
1
Enter details for applicant 1:
Enter name: Alice
Enter score: 90
Details of applicant 1:
Application ID: 1
Name: Alice
Score: 90
Enter choice: 1 for adding applicants and 2 for exiting:->
1
Enter details for applicant 2:
Enter name: Bob
Enter score: 100
Details of applicant 2:
Application ID: 2
Name: Bob
Score: 100
Enter choice: 1 for adding applicants and 2 for exiting:->
2
Total number of applicants: 2


=== Code Execution Successful ===
```

Q10. Design a STUDENT class to store roll, name, course, admission date and marks in 5 subjects. Provide methods corresponding to admission (marks are not available then), receiving marks and preparing mark sheet. Support must be there to show the number of students who have taken admission.

```
#include <bits/stdc++.h>
using namespace std;

class STUDENT {
private:
static int totalStudents;        //Static variable to keep track of the number of students
int roll;
string name;
string course;
string admissionDate;
float marks[5];                  // Array to store marks for 5 subjects
bool marksReceived;          // Flag to check if marks have been received

public:
// Constructor to initialize the student with an auto-generated roll number
STUDENT() : roll(++totalStudents), marksReceived(false) {
for (int i = 0; i < 5; ++i) {
marks[i] = 0.0;
}
}

// Method to admit a student
void admit(const string& studentName, const string& studentCourse, const string& date) {
name = studentName;
course = studentCourse;
admissionDate = date;
marksReceived = false;               // Marks are not available at admission
}
```

```
// Method to receive marks for the student
void receiveMarks(const float subjectMarks[5]) {
for (int i = 0; i < 5; ++i) {
marks[i] = subjectMarks[i];
}
marksReceived = true;
}

void prepareMarkSheet() const {          // Method to prepare and display the mark sheet
if (!marksReceived) {
cout << "Marks have not been received yet." << endl;
return;
}

cout << "Mark Sheet for Roll Number: " << roll << endl;
cout << "Name: " << name << endl;
cout << "Course: " << course << endl;
cout << "Admission Date: " << admissionDate << endl;
cout << "Marks:" << endl;
for (int i = 0; i < 5; ++i) {
cout << "Subject " << (i + 1) << ": " << fixed << setprecision(2) << marks[i] << endl;
}
}

// Static method to get the number of students admitted
static int getNumberOfStudents() {
```

```
return totalStudents;
}
};

int STUDENT::totalStudents = 0;    // Initialize the static variable

int main() {
STUDENT student1, student2;

// Admitting students
student1.admit("Alice", "Computer Science", "2024-08-15");
student2.admit("Bob", "Mechanical Engineering", "2024-08-16");

// Receiving marks
float marks1[5] = {85.5, 90.0, 78.0, 92.5, 88.0};
float marks2[5] = {82.0, 79.5, 88.0, 91.0, 84.5};
student1.receiveMarks(marks1);
student2.receiveMarks(marks2);

// Preparing mark sheets
cout << "Mark Sheet for Student 1:" << endl;
student1.prepareMarkSheet();
cout << endl;

cout << "Mark Sheet for Student 2:" << endl;
student2.prepareMarkSheet();
cout << endl;

// Displaying the number of students
cout << "Total number of students admitted: " << STUDENT::getNumberOfStudents() << endl;
```

return 0;

}

```
Mark Sheet for Student 1:
Mark Sheet for Roll Number: 1
Name: Alice
Course: Computer Science
Admission Date: 2024-08-15
Marks:
Subject 1: 85.50
Subject 2: 90.00
Subject 3: 78.00
Subject 4: 92.50
Subject 5: 88.00

Mark Sheet for Student 2:
Mark Sheet for Roll Number: 2
Name: Bob
Course: Mechanical Engineering
Admission Date: 2024-08-16
Marks:
Subject 1: 82.00
Subject 2: 79.50
Subject 3: 88.00
Subject 4: 91.00
Subject 5: 84.50

Total number of students admitted: 2


=== Code Execution Successful ===
```

Q11. Create a class for linked list. Consider a separate class NODE for basic node activities and use it in class for linked list.

#include <iostream>

using namespace std;

```
class Node {            // Node class
public:
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {      // LinkedList class
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void append(int value) {        // Function to add a node at the end
        Node* newNode = new Node(value);
        if (!head) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
        cout << "Node with value " << value << " appended.\n";
    }
```

```
void insertAtBeginning(int value) {       // Function to insert a node at the beginning
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
    cout << "Node with value " << value << " inserted at the beginning.\n";
}

void deleteByValue(int value) {.       // Function to delete a node by value
    if (!head) {
        cout << "List is empty. Cannot delete.\n";
        return;
    }
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Node with value " << value << " deleted.\n";
        return;
    }
    Node* temp = head;
    while (temp->next && temp->next->data != value) {
        temp = temp->next;
    }
    if (temp->next) {
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
```

```
            delete nodeToDelete;
            cout << "Node with value " << value << " deleted.\n";
        } else {
            cout << "Value " << value << " not found in the list.\n";
        }
    }

    void display() {        // Function to display the linked list
        if (!head) {
            cout << "List is empty.\n";
            return;
        }
        Node* temp = head;
        while (temp) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL\n";
    }
    ~LinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    }
};
```

```
int main() {
    LinkedList list;
    int choice, value;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Append Node\n";
        cout << "2. Insert at Beginning\n";
        cout << "3. Delete by Value\n";
        cout << "4. Display List\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value to append: ";
            cin >> value;
            list.append(value);
            break;
        case 2:
            cout << "Enter value to insert at beginning: ";
            cin >> value;
            list.insertAtBeginning(value);
            break;
        case 3:
```

```
            cout << "Enter value to delete: ";
            cin >> value;
            list.deleteByValue(value);
            break;
        case 4:
            cout << "Linked List: ";
            list.display();
            break;
        case 5:
            cout << "Exiting program.\n";
            return 0;
        default:
            cout << "Invalid choice. Please try again.\n";
        }
    }
}
```

```
Menu:
1. Append Node
2. Insert at Beginning
3. Delete by Value
4. Display List
5. Exit
Enter your choice: 1
Enter value to append: 45
Node with value 45 appended.

Menu:
1. Append Node
2. Insert at Beginning
3. Delete by Value
4. Display List
5. Exit
Enter your choice: 3
Enter value to delete: 50
Value 50 not found in the list.

Menu:
1. Append Node
2. Insert at Beginning
3. Delete by Value
4. Display List
5. Exit
Enter your choice: 4
Linked List: 45 -> NULL

Menu:
1. Append Node
2. Insert at Beginning
3. Delete by Value
4. Display List
5. Exit
Enter your choice: 2
Enter value to insert at beginning: 40
Node with value 40 inserted at the beginning.
```

Q12. Design the class(es) for the following scenario:

- An item list contains item code, name, rate, and quantity for several items.

- Whenever a new item is added in the list uniqueness of item code is to be checked.

- Time to time rate of the items may change.

- Whenever an item is issued or received existence of the item is checked and quantity is updated.

- In case of issue, availability of quantity is also to be checked.

- User may also like to know price/quantity available for an item.

```
#include <bits/stdc++.h>
using namespace std;

class Item {
    string code, name;
    double rate;
    int quantity;

public:
    // Constructor
    Item(string c, string n, double r, int q) : code(c), name(n), rate(r), quantity(q) {}

    // Getter methods
    string getCode() const { return code; }
    string getName() const { return name; }
    double getRate() const { return rate; }
    int getQuantity() const { return quantity; }
    // Update rate
    void updateRate(double newRate) { rate = newRate; }
    // Update quantity (for issuing and receiving)
    void updateQuantity(int qty) { quantity += qty; }
    // Display item details
    void display() const {
```

```
        cout << "Code: " << code << ", Name: " << name
             << ", Rate: " << rate << ", Quantity: " << quantity << endl;
    }
};

class ItemList {
    vector<Item> items;
    // Find item by code
    int findItemIndex(const string &code) const {
        for (size_t i = 0; i < items.size(); i++) {
            if (items[i].getCode() == code)
                return i;
        }
        return -1;
    }
public:
    // Add a new item
    bool addItem(const string &code, const string &name, double rate, int quantity) {
        if (findItemIndex(code) != -1) {
            cout << "Item code must be unique!" << endl;
            return false;
        }
        items.emplace_back(code, name, rate, quantity);
        return true;
    }
    // Update rate of an item
    bool updateRate(const string &code, double newRate) {
```

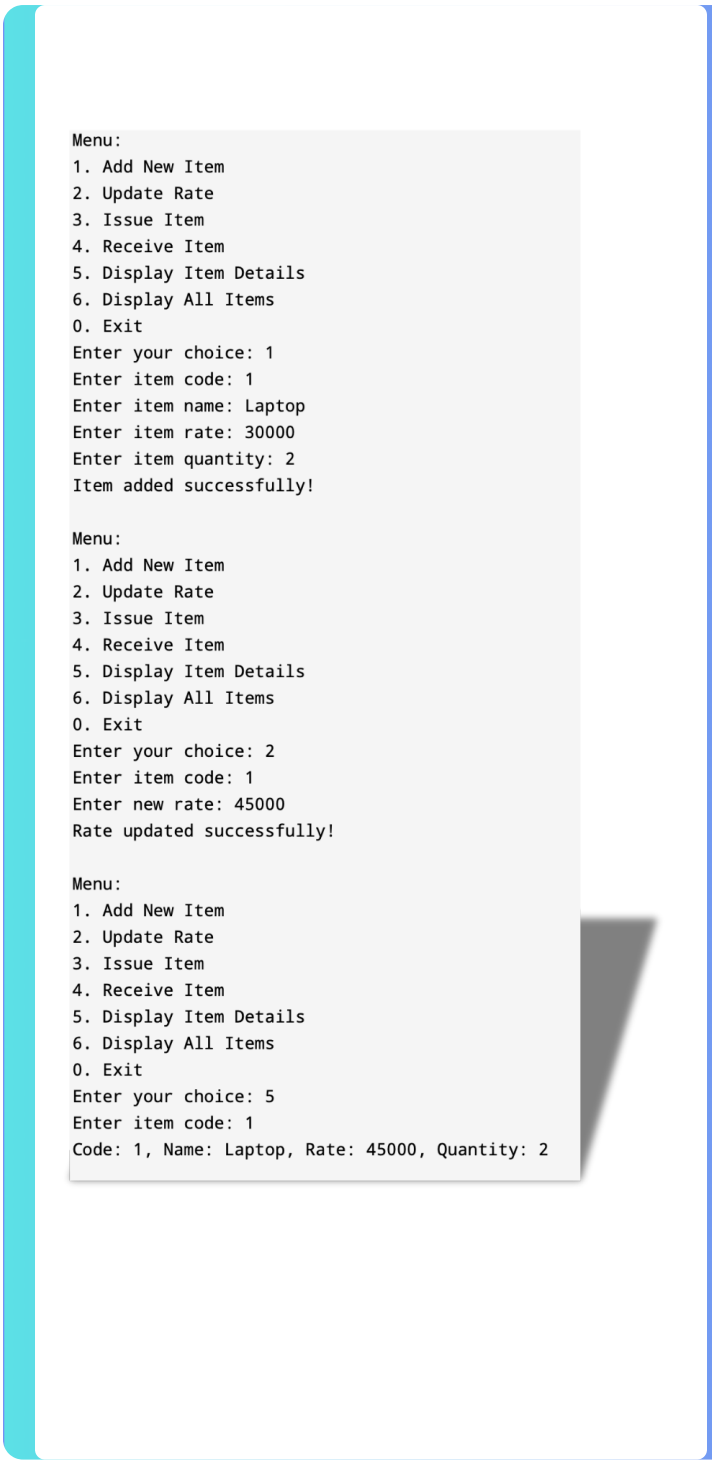
```
    int index = findItemIndex(code);
    if (index == -1) {
        cout << "Item not found!" << endl;
        return false;
    }
    items[index].updateRate(newRate);
    return true;
}
// Issue or receive item
bool updateQuantity(const string &code, int qty) {
    int index = findItemIndex(code);
    if (index == -1) {
        cout << "Item not found!" << endl;
        return false;
    }
    if (qty < 0 && items[index].getQuantity() + qty < 0) {
        cout << "Insufficient quantity available!" << endl;
        return false;
    }
    items[index].updateQuantity(qty);
    return true;
}
// Display details of an item
bool displayItem(const string &code) const {
    int index = findItemIndex(code);
    if (index == -1) {
        cout << "Item not found!" << endl;
```

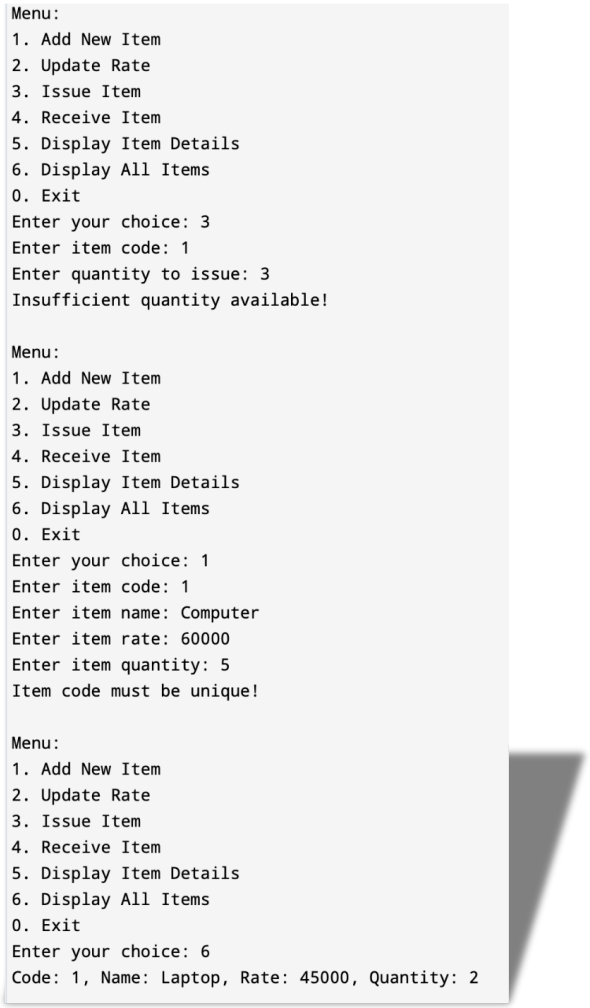
```
        return false;
    }
    items[index].display();
    return true;
  }

void displayAllItems() const {
    if (items.empty()) {
        cout << "No items in the list!" << endl;
        return;
    }
    for (const auto &item : items)
        item.display();
  }
};

int main() {
  ItemList itemList;
  int choice;

  do {
    cout << "\nMenu:\n"
         << "1. Add New Item\n"
         << "2. Update Rate\n"
         << "3. Issue Item\n"
         << "4. Receive Item\n"
         << "5. Display Item Details\n"
```

```
    << "6. Display All Items\n"
    << "0. Exit\n"
    << "Enter your choice: ";
cin >> choice;

switch (choice) {
   case 1: {
      string code, name;
      double rate;
      int quantity;
      cout << "Enter item code: ";
      cin >> code;
      cout << "Enter item name: ";
      cin >> name;
      cout << "Enter item rate: ";
      cin >> rate;
      cout << "Enter item quantity: ";
      cin >> quantity;
      if (itemList.addItem(code, name, rate, quantity))
         cout << "Item added successfully!" << endl;
      break;
   }
   case 2: {
      string code;
      double newRate;
      cout << "Enter item code: ";
      cin >> code;
```

```
    cout << "Enter new rate: ";
    cin >> newRate;
    if (itemList.updateRate(code, newRate))
        cout << "Rate updated successfully!" << endl;
    break;
}
case 3: {
    string code;
    int qty;
    cout << "Enter item code: ";
    cin >> code;
    cout << "Enter quantity to issue: ";
    cin >> qty;
    if (itemList.updateQuantity(code, -qty))
        cout << "Item issued successfully!" << endl;
    break;
}
case 4: {
    string code;
    int qty;
    cout << "Enter item code: ";
    cin >> code;
    cout << "Enter quantity to receive: ";
    cin >> qty;
    if (itemList.updateQuantity(code, qty))
        cout << "Item received successfully!" << endl;
    break;
```

```
        }
        case 5: {
            string code;
            cout << "Enter item code: ";
            cin >> code;
            itemList.displayItem(code);
            break;
        }
        case 6: {
            itemList.displayAllItems();
            break;
        }
        case 0:
            cout << "Exiting program. Goodbye!" << endl;
            break;
        default:
            cout << "Invalid choice! Please try again." << endl;
      }
   } while (choice != 0);

   return 0;
}
```





Q13. Design a BALANCE class with account number, balance and date of last update. Consider a TRANSACTION class with account number, date of transaction, amount and transaction type (W for withdrawal and D for deposit). If it is withdrawal then check whether the amount is available or not. Transaction object will make necessary update in the BALANCE class.

```
#include <bits/stdc++.h>
```

```
using namespace std;

class BALANCE {
private:
    int accountNumber;
    double balance;
    string lastUpdateDate;

public:
    BALANCE(int accNum, double bal, string date)
        : accountNumber(accNum), balance(bal), lastUpdateDate(date) {}

    void displayBalance() {
        cout << "Account Number: " << accountNumber << endl;
        cout << "Balance: " << balance << endl;
        cout << "Last Update Date: " << lastUpdateDate << endl;
    }

    bool withdraw(double amount) {
        if (amount > balance) {
            cout << "Insufficient balance!" << endl;
            return false;
        }
        balance -= amount;
        return true;
    }

    void deposit(double amount) {
```

```
        balance += amount;
    }

    void updateDate(string date) {
        lastUpdateDate = date;
    }

    double getBalance() const {
        return balance;
    }
};

class TRANSACTION {
private:
    int accountNumber;
    string transactionDate;
    double amount;
    char transactionType;        // 'W' for withdrawal, 'D' for deposit

public:
    TRANSACTION(int accNum, string date, double amt, char type)
        : accountNumber(accNum), transactionDate(date), amount(amt), transactionType(type)
{}

    void processTransaction(BALANCE &balanceObj) {
        if (transactionType == 'W') {
            if (balanceObj.withdraw(amount)) {
                balanceObj.updateDate(transactionDate);
                cout << "Withdrawal successful!" << endl;
```

```
        }
    } else if (transactionType == 'D') {
        balanceObj.deposit(amount);
        balanceObj.updateDate(transactionDate);
        cout << "Deposit successful!" << endl;
    } else {
        cout << "Invalid transaction type!" << endl;
    }
  }

  void displayTransaction() {
    cout << "Transaction Details:" << endl;
    cout << "Account Number: " << accountNumber << endl;
    cout << "Transaction Date: " << transactionDate << endl;
    cout << "Amount: " << amount << endl;
    cout << "Transaction Type: " << (transactionType == 'W' ? "Withdrawal" : "Deposit") <<
endl;
  }
};

int main() {
  int accountNumber;
  double initialBalance;
  string lastUpdateDate;

  cout << "Enter account number: ";
  cin >> accountNumber;
  cout << "Enter initial balance: ";
  cin >> initialBalance;
```

```
cout << "Enter date of last update (YYYY-MM-DD): ";
cin >> lastUpdateDate;

BALANCE balance(accountNumber, initialBalance, lastUpdateDate);

int choice;
do {
    cout << "\nMenu:\n";
    cout << "1. Display Balance\n";
    cout << "2. Make a Transaction\n";
    cout << "3. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
    case 1:
        balance.displayBalance();
        break;

    case 2: {
        char transactionType;
        double amount;
        string transactionDate;

        cout << "Enter transaction type (W for Withdrawal, D for Deposit): ";
        cin >> transactionType;
        cout << "Enter amount: ";
        cin >> amount;
```

```
            cout << "Enter transaction date (YYYY-MM-DD): ";
            cin >> transactionDate;

            TRANSACTION transaction(accountNumber, transactionDate, amount,
transactionType);
            transaction.processTransaction(balance);
            transaction.displayTransaction();
            break;
        }

        case 3:
            cout << "Exiting the program." << endl;
            break;

        default:
            cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 3);

    return 0;
}
```

```
Enter account number: 123
Enter initial balance: 5000
Enter date of last update (YYYY-MM-DD): 2000-12-03

Menu:
1. Display Balance
2. Make a Transaction
3. Exit
Enter your choice: 1
Account Number: 123
Balance: 5000
Last Update Date: 2000-12-03

Menu:
1. Display Balance
2. Make a Transaction
3. Exit
Enter your choice: 2
Enter transaction type (W for Withdrawal, D for Deposit): W
Enter amount: 10000
Enter transaction date (YYYY-MM-DD): 2020-05-04
Insufficient balance!
Transaction Details:
Account Number: 123
Transaction Date: 2020-05-04
Amount: 10000
Transaction Type: Withdrawal

Menu:
1. Display Balance
2. Make a Transaction
3. Exit
Enter your choice: 2
Enter transaction type (W for Withdrawal, D for Deposit): D
Enter amount: 5000
Enter transaction date (YYYY-MM-DD): 2030-04-01
Deposit successful!
Transaction Details:
Account Number: 123
Transaction Date: 2030-04-01
Amount: 5000
Transaction Type: Deposit
```

```
Menu:
1. Display Balance
2. Make a Transaction
3. Exit
Enter your choice: 1
Account Number: 123
Balance: 10000
Last Update Date: 2030-04-01
```

Assignment 4

Q1. Design the class(es) for the following. Each account has account number and balance amount. A list of account is to be maintained where one can add and find account, display information of all accounts. While adding, account number must be unique. Withdraw object has account number (must exist) and amount (will not exceed balance amount of corresponding account). Withdraw object will update the balance of corresponding account in the list. User will be able to search and view account, add account and withdraw money from the account. Implement your design. Use friend function wherever required and again, modify your implementation to avoid friend function.

```
#include <bits/stdc++.h>

using namespace std;


class Account {

   int accountNumber;

   double balance;

public:

   // Constructor

   Account(int accNo, double bal) : accountNumber(accNo), balance(bal) { }

   // Getters

   int getAccountNumber() const { return accountNumber; }

   double getBalance() const { return balance; }

   // Update balance

   void updateBalance(double amount) { balance += amount; }

   // Display account details

   void display() const {
```

```
        cout << "Account Number: " << accountNumber << ", Balance: " << balance << endl;
    }
};

class AccountList {
    vector<Account> accounts;

public:
    // Add a new account
    bool addAccount(int accNo, double initialBalance) {
        if (findAccount(accNo) != nullptr) {
            cout << "Error: Account number must be unique!" << endl;
            return false;
        }
        accounts.emplace_back(accNo, initialBalance);
        cout << "Account added successfully!" << endl;
        return true;
    }

    // Find an account by account number
    Account* findAccount(int accNo) {
        for (auto& account : accounts) {
            if (account.getAccountNumber() == accNo) {
                return &account;
            }
```

```
    }
    return nullptr;
}

// Display all accounts
void displayAllAccounts() const {
    if (accounts.empty()) {
        cout << "No accounts to display." << endl;
        return;
    }
    for (const auto& account : accounts) {
        account.display();
    }
}

// Withdraw money
bool withdraw(int accNo, double amount) {
    Account* account = findAccount(accNo);
    if (!account) {
        cout << "Error: Account not found!" << endl;
        return false;
    }
    if (account->getBalance() < amount) {
        cout << "Error: Insufficient balance!" << endl;
        return false;
```

```
        }
        account->updateBalance(-amount);
        cout << "Withdrawal successful. Updated balance: " << account->getBalance() << endl;
        return true;
    }
};

int main() {
    AccountList accountList;
    int choice;

    do {
        cout << "\nMenu:\n";
        cout << "1. Add Account\n";
        cout << "2. Find Account\n";
        cout << "3. Display All Accounts\n";
        cout << "4. Withdraw Money\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int accNo;
                double initialBalance;
```

```
        cout << "Enter account number: ";
        cin >> accNo;
        cout << "Enter initial balance: ";
        cin >> initialBalance;
        accountList.addAccount(accNo, initialBalance);
        break;
    }
    case 2: {
        int accNo;
        cout << "Enter account number to find: ";
        cin >> accNo;
        Account* account = accountList.findAccount(accNo);
        if (account) {
            account->display();
        } else {
            cout << "Account not found!" << endl;
        }
        break;
    }
    case 3:
        accountList.displayAllAccounts();
        break;
    case 4: {
        int accNo;
        double amount;
```

```
            cout << "Enter account number: ";
            cin >> accNo;
            cout << "Enter amount to withdraw: ";
            cin >> amount;
            accountList.withdraw(accNo, amount);
            break;
        }
        case 5:
            cout << "Exiting....." << endl;
            break;
        default:
            cout << "Invalid choice! Try again." << endl;
      }
   } while (choice != 5);

   return 0;
}
```

```
Menu:
1. Add Account
2. Find Account
3. Display All Accounts
4. Withdraw Money
5. Exit
Enter your choice: 1
Enter account number: 1
Enter initial balance: 5000
Account added successfully!

Menu:
1. Add Account
2. Find Account
3. Display All Accounts
4. Withdraw Money
5. Exit
Enter your choice: 4
Enter account number: 1
Enter amount to withdraw: 10000ERROR!

Error: Insufficient balance!

Menu:
1. Add Account
2. Find Account
3. Display All Accounts
4. Withdraw Money
5. Exit
Enter your choice: 2
Enter account number to find: 1
Account Number: 1, Balance: 5000
```

Q2. Design a COMPLEX class, which will behave like normal integer with respect to

→ addition,

→ subtraction,

→ accepting the value and

→ Displaying the value.

```
#include <iostream>

using namespace std;


class Complex {
   int real;
   int imaginary;


public:
   // Constructor to initialize complex number
   Complex(int r = 0, int i = 0) : real(r), imaginary(i) {}


   // Operator Overloading
   Complex operator+(const Complex &obj) {
      Complex temp;
      temp.real = real + obj.real;
      temp.imaginary = imaginary + obj.imaginary;
      return temp;
   }


   Complex operator-(const Complex &obj) {
      Complex temp;
      temp.real = real - obj.real;
      temp.imaginary = imaginary - obj.imaginary;
```

```
        return temp;
    }

    void accept() {
        cout << "Enter real part: ";
        cin >> real;
        cout << "Enter imaginary part: ";
        cin >> imaginary;
    }

    void display() const {
        if (imaginary >= 0)
            cout << real << " + " << imaginary << "i" << endl;
        else
            cout << real << " - " << -imaginary << "i" << endl;
    }
};

int main() {
    Complex c1, c2, result;
    cout << "Enter first complex number:\n";
    c1.accept();
    cout << "Enter second complex number:\n";
    c2.accept();
    result = c1 + c2;
```

```
    cout << "Sum: ";
    result.display();
    result = c1 - c2;
    cout << "Difference: ";
    result.display();

    return 0;
}
```

```
Enter first complex number:
Enter real part: 1
Enter imaginary part: 2
Enter second complex number:
Enter real part: 3
Enter imaginary part: 4
Sum: 4 + 6i
Difference: -2 - 2i


=== Code Execution Successful ===
```

3. Design an ARRAY class with the following features:

a. Array object may be declared for a specific size and a value for initializing all the elements. If this it is to be assumed as a 0.

b. An array object may be declared and initialized with another object.

c. An array object may be declared and initialized with another array (not the object, standard array as in C language).

Let a and b are two objects:

i. a+b will add corresponding elements.

ii. a=b will do the assignment.

iii. a[l] will return the ith element of the object.

iv. a*5 or 5*a will multiply the element with 5.

```
#include <bits/stdc++.h>

using namespace std;


class Array {

   int* arr;

   int size;


public:

   // Constructor to initialize array with size and optional initialization value

   Array(int sz, int initVal = 0) : size(sz) {

      arr = new int[size];

      for (int i = 0; i < size; i++) {

         arr[i] = initVal;

      }

   }

   // Copy constructor for initializing with another object

   Array(const Array& other) : size(other.size) {

      arr = new int[size];

      for (int i = 0; i < size; i++) {

         arr[i] = other.arr[i];
```

```
    }
}
// Constructor to initialize with a standard C-style array
Array(int* standardArr, int sz) : size(sz) {
    arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = standardArr[i];
    }
}
~Array() {        // Destructor to clean up dynamically allocated memory
    delete[] arr;
}
// Overloading the + operator for adding two arrays element-wise
Array operator+(const Array& other) {
    assert(size == other.size); // Ensure sizes match
    Array result(size);
    for (int i = 0; i < size; i++) {
        result.arr[i] = arr[i] + other.arr[i];
    }
    return result;
}
// Overloading the = operator for assignment
Array& operator=(const Array& other) {
    if (this != &other) {          // Avoid self-assignment
        if (size != other.size) {
```

```
            delete[] arr;               // Clean up existing memory
            size = other.size;
            arr = new int[size];
        }
        for (int i = 0; i < size; i++) {
            arr[i] = other.arr[i];
        }
    }
    return *this;
}
// Overloading the subscript operator for element access
int& operator[](int index) {
     assert(index >= 0 && index < size);           //Ensure index is within bounds
    return arr[index];
}
// Overloading the * operator for multiplying array elements with a scalar
Array operator*(int scalar) {
    Array result(size);
    for (int i = 0; i < size; i++) {
        result.arr[i] = arr[i] * scalar;
    }
    return result;
}
// Friend function to handle 5*a multiplication (commutative)
friend Array operator*(int scalar, const Array& a) {
```

```
        Array result(a.size);
        for (int i = 0; i < a.size; i++) {
            result.arr[i] = a.arr[i] * scalar;
        }
        return result;
    }
    // Function to display the array elements
    void display() const {
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    // Array object initialized with size 5, all elements set to 0
    Array a(5);
    cout << "Array a initialized with size 5 and default value 0: ";
    a.display();
    // Array object initialized with size 5, all elements set to 3
    Array b(5, 3);
    cout << "Array b initialized with size 5 and all elements set to 3: ";
    b.display();
    // Assigning array b to a
```

```
    a = b;
    cout << "After assigning b to a: ";
    a.display();
    // Adding a and b
    Array c = a + b;
    cout << "Array c = a + b: ";
    c.display();
    // Accessing the third element of c
    cout << "c[2] (third element of c): " << c[2] << endl;
    // Multiply array a with scalar 5
    Array d = a * 5;
    cout << "Array d = a * 5: ";
    d.display();
    // Multiply scalar 5 with array b
    Array e = 5 * b;
    cout << "Array e = 5 * b: ";
    e.display();

    return 0;
}
```

```
Array a initialized with size 5 and default value 0: 0 0 0 0 0
Array b initialized with size 5 and all elements set to 3: 3 3 3 3 3
After assigning b to a: 3 3 3 3 3
Array c = a + b: 6 6 6 6 6
c[2] (third element of c): 6
Array d = a * 5: 15 15 15 15 15
Array e = 5 * b: 15 15 15 15 15


=== Code Execution Successful ===
```

Q4. Design a STRING class, which will have the initialization facility similar to array class. Provide support for:

→ Assigning one object for another,

→ Two string can be concatenated using + operator,

→ Two strings can be compared using the relational operators.

```
#include <iostream>

#include <cstring>

using namespace std;


class String {

char* str;

int length;


public:

// Constructor to initialize with a C-style string

String(const char* s = "") {

length = strlen(s);

str = new char[length + 1];
```

```
strcpy(str, s);
}

// Copy constructor to initialize one object with another
String(const String& other) {
length = other.length;
str = new char[length + 1];
strcpy(str, other.str);
}

~String() {
delete[] str;
}

// Overloading the assignment operator (=) for deep copying
String& operator=(const String& other) {
if (this != &other) {              // Avoiding self-assignment
delete[] str;                      // Free existing memory
length = other.length;
str = new char[length + 1];
strcpy(str, other.str);
}
return *this;
}

// Overloading the + operator to concatenate two strings
String operator+(const String& other) const {
```

```
String temp;
temp.length = length + other.length;
temp.str = new char[temp.length + 1];
strcpy(temp.str, str);
strcat(temp.str, other.str);
return temp;
}

// Overloading the == operator for string comparison
bool operator==(const String& other) const {
return strcmp(str, other.str) == 0;
}

// Overloading the < operator for lexicographical comparison
bool operator<(const String& other) const {
return strcmp(str, other.str) < 0;
}

// Overloading the > operator for lexicographical comparison
bool operator>(const String& other) const {
return strcmp(str, other.str) > 0;
}

void display() const {
cout << str << endl;
}
};
```

```
int main() {
String s1("Hello");
String s2("World");

cout << "String 1: ";
s1.display();
cout << "String 2: ";
s2.display();

String s3 = s1 + s2;
cout << "Concatenated String (s1 + s2): ";
s3.display();

String s4 = s1;
cout << "Assigned String (s4 = s1): ";
s4.display();

if (s1 == s4) {
cout << "s1 and s4 are equal." << endl;
}

if (s1 < s2) {
cout << "s1 is lexicographically less than s2." << endl;
}

if (s2 > s1) {
cout << "s2 is lexicographically greater than s1." << endl;
```

```
}

return 0;

}
```

```
String 1: Hello
String 2: World
Concatenated String (s1 + s2): HelloWorld
Assigned String (s4 = s1): Hello
s1 and s4 are equal.
s1 is lexicographically less than s2.
s2 is lexicographically greater than s1.


=== Code Execution Successful ===
```

Q5. Modify the STRING class so that assigning/initializing a string by another will not copy it physically but will keep a reference count, which will be incremented.

Reference value 0 means the space can be released

```
#include <iostream>

#include <cstring>

using namespace std;


class String {

   char* str;            // Pointer to the string data
```

```
    int* refCount;      // Pointer to the reference count
    int length;         // Length of the string

public:
    // Constructor to initialize the string with a C-style string
    String(const char* s = "") {
        length = strlen(s);
        str = new char[length + 1];
        strcpy(str, s);
        refCount = new int(1);          // Reference count starts at 1
    }

    // Copy constructor (shallow copy, incrementing the reference count)
    String(const String& other) {
        str = other.str;
        length = other.length;
        refCount = other.refCount;
        (*refCount)++;
    }

    // Destructor to decrement the reference count and delete the string if necessary
    ~String() {
        release();
    }
```

```
    // Function to release memory when reference count reaches zero
    void release() {
        (*refCount)--;
        if (*refCount == 0) {
            delete[] str;
            delete refCount;
        }
    }

String& operator=(const String& other) {
        if (this != &other) {
            release(); // Release old memory
            str = other.str;
            length = other.length;
            refCount = other.refCount;
            (*refCount)++;
        }
        return *this;
    }

    // Function to ensure unique ownership of the string (copy-on-write mechanism)
    void ensureUnique() {
        if (*refCount > 1) {
            (*refCount)--;
```

```
        char* newStr = new char[length + 1];
        strcpy(newStr, str);

        str = newStr;
        refCount = new int(1);
    }
}

String operator+(const String& other) const {
    String temp;
    temp.length = length + other.length;
    temp.str = new char[temp.length + 1];
    strcpy(temp.str, str);
    strcat(temp.str, other.str);
    temp.refCount = new int(1);        // New object, so reference count is 1
    return temp;
}
bool operator==(const String& other) const {
    return strcmp(str, other.str) == 0;
}
bool operator<(const String& other) const {
    return strcmp(str, other.str) < 0;
}
bool operator>(const String& other) const {
    return strcmp(str, other.str) > 0;
```

```
    }
    void display() const {
        cout << str << " (refCount: " << *refCount << ")" << endl;
    }

    // Function to modify a specific character (Copy on Write mechanism)
    void modify(int index, char ch) {
        ensureUnique(); // Ensure the string is not shared before modifying
        if (index >= 0 && index < length) {
            str[index] = ch;
        }
    }
};

int main() {
    String s1("Hello");
    String s2(s1); // Copy constructor, no new string copy
    String s3 = s1; // Assignment, no new string copy
    cout << "String s1: "; s1.display();
    cout << "String s2 (copy of s1): "; s2.display();
    cout << "String s3 (copy of s1): "; s3.display();
    s1.modify(0, 'h');
    cout << "\nAfter modifying s1:\n";
    cout << "String s1: "; s1.display();
    cout << "String s2: "; s2.display();
```

```
    cout << "String s3: "; s3.display();

    String s4 = s1 + s2;

    cout << "\nString s4 (s1 + s2): "; s4.display();


    return 0;

}
```

```
String s1: Hello (refCount: 3)
String s2 (copy of s1): Hello (refCount: 3)
String s3 (copy of s1): Hello (refCount: 3)

After modifying s1:
String s1: hello (refCount: 1)
String s2: Hello (refCount: 2)
String s3: Hello (refCount: 2)

String s4 (s1 + s2): helloHello (refCount: 1)


=== Code Execution Successful ===
```

Assignment 5

Q1.There are number of students. For every student roll (unique), name is to be stored. For each subject, subject code and name is to be stored. A student can opt for number of subjects. System should be able to maintain student list, subject list and will be able to answer: i) which student has selected which subjects and ii) for a subjects who are the students. Design the classes and implement. For list consider memory data structure.

```
#include <bits/stdc++.h>
using namespace std;

class Subject;              //forward declaration as it has to be used in Student class

class Student {
private:
int roll;

string name;
vector<shared_ptr<Subject>> subjects;

public:
 Student(int roll, string name) : roll(roll), name(name) {}

 int getRoll() const {
 return roll;
 }

 string getName() const {
 return name;
 }

 void addSubject(shared_ptr<Subject> subject) {
 subjects.push_back(subject);
 }

 const vector<shared_ptr<Subject>>& getSubjects() const {
 return subjects;
```

```
 }
 //const functions cannot change member variables
};

class Subject {
private:
 string code;
 string name;
 vector<shared_ptr<Student>> students;

public:
 Subject(string code, string name) : code(code), name(name) {}

 string getCode() const {
 return code;
 }

 string getName() const {
 return name;
 }

 void addStudent(shared_ptr<Student> student) {
 students.push_back(student);
 }

 const vector<shared_ptr<Student>>& getStudents() const {
 return students;
 }
};
//std::shared_ptr is a smart pointer that retains shared ownership of an object through a pointer.
Several shared_ptrobjects may own the same object.


class SchoolSystem {
private:
 unordered_map<int, shared_ptr<Student>> studentList;
 unordered_map<string, shared_ptr<Subject>> subjectList;

public:
 void addStudent(int roll, string name) {
 if (studentList.find(roll) != studentList.end()) {
 cout << "Student with this roll number already exists" << endl;
 return;
 }
```

```
studentList[roll] = make_shared<Student>(roll, name);
}

void addSubject(string code, string name) {
if (subjectList.find(code) != subjectList.end()) {
cout << "Subject with this code already exists" << endl;
return;
}
subjectList[code] = make_shared<Subject>(code, name);
}

void enrollStudentInSubject(int roll, string subjectCode) {
auto student = studentList.find(roll);
auto subject = subjectList.find(subjectCode);

if (student == studentList.end()) {
cout << "Student with this roll number does not exist" << endl;
} else if (subject == subjectList.end()) {
cout << "Subject with this code does not exist" << endl;
} else {
student->second->addSubject(subject->second);
subject->second->addStudent(student->second);
}
}

vector<shared_ptr<Subject>> getSubjectsByStudent(int roll) const {
auto student = studentList.find(roll);
if (student != studentList.end()) {
return student->second->getSubjects();
}
cout << "Student with this roll number does not exist" << endl;
return {};
}

vector<shared_ptr<Student>> getStudentsBySubject(string subjectCode) const {
auto subject = subjectList.find(subjectCode);
if (subject != subjectList.end()) {
return subject->second->getStudents();
}
cout << "Subject with this code does not exist" << endl;
return {};
}
};
```

```
int main() {
 SchoolSystem schoolSystem;
 int choice;

 while (true) {
 cout << "Enter your choice:\n";
 cout << "1. Add Student\n";
 cout << "2. Add Subject\n";
 cout << "3. Enroll Student in Subject\n";
 cout << "4. Get Subjects by Student\n";
 cout << "5. Get Students by Subject\n";
 cout << "6. Exit\n";
 cin >> choice;
 //menu driven layout according to user's choice
 if (choice == 6)
 break;

 switch (choice) {
 case 1: {
 cout << "Enter roll number: ";
 int roll;
 cin >> roll;
 cout << "Enter name: ";
 string name;
 cin >> name;
 schoolSystem.addStudent(roll, name);
 break;
 }
 case 2: {
 cout << "Enter subject code: ";
 string code;
 cin >> code;
 cout << "Enter subject name: ";
 string name;
 cin >> name;
 schoolSystem.addSubject(code, name);
 break;
 }
 case 3: {
 cout << "Enter roll number: ";
 int roll;
 cin >> roll;
 cout << "Enter subject code: ";
 string code;
```

```
cin >> code;
schoolSystem.enrollStudentInSubject(roll, code);
break;
}
case 4: {
cout << "Enter roll number: ";
int roll;
cin >> roll;
auto subjects = schoolSystem.getSubjectsByStudent(roll);
for (const auto& subject : subjects) {
cout << "Subject code: " << subject->getCode()
<< ", Subject name: " << subject->getName() << endl;
}
break;
}
case 5: {
cout << "Enter subject code: ";
string code;
cin >> code;
auto students = schoolSystem.getStudentsBySubject(code);
for (const auto& student : students) {
cout << "Roll number: " << student->getRoll()
<< ", Name: " << student->getName() << endl;
}
break;
}
default:
cout << "Invalid choice" << endl;
}
}

return 0;
}
```

```
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
1
Enter roll number: 30
Enter name: Tathagata
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
2
Enter subject code: 1
Enter subject name: Physics
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
3
Enter roll number: 30
Enter subject code: 1
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
5
Enter subject code: 1
Roll number: 30, Name: Tathagata
```

Q2. In a Temp, for each book book-id, serial number (denotes copy number of a book), title, author, publisher and price are stored. Book-id and serial number together will be unique identifier for a book. Members are either student or faculty. Each member has unique member-id. Name, e-mail, address are also to be stored. For any transaction (book issue or return), members are supposed to place transactions slip. User will submit member-id, book-id, and serial number (only for book return). While processing a transaction, check the validity of the member. While issuing, availability

of a copy of the book is to be checked. While returning a book, it is to be checked whether this copy was issued to the member or not. A student member can have 2 books issued at a point of time. For faculty members it is 10. Transaction information is to be stored like date of transaction, member-id, book-id, serial number, returned or not. An entry is made when book is issued and updated when the the book is returned.

Design the classes and implement. For list consider memory data structure.

```
#include <bits/stdc++.h>
using namespace std;

class Book {
public:
 string bookId;
 int serialNumber;
 string title;
 string author;
 string publisher;
 double price;
 bool isIssued;

 Book(string bookId, int serialNumber, string title, string author, string publisher, double price)
 : bookId(bookId), serialNumber(serialNumber), title(title), author(author), publisher(publisher),
price(price), isIssued(false) {}
                                        //default parameterized constructor of book
};

class Member {
public:
 string memberId;
 string name;
 string email;
 string address;
 int maxBooksAllowed;
 vector<Book*> issuedBooks;

 Member(string memberId, string name, string email, string address, int maxBooksAllowed)
 : memberId(memberId), name(name), email(email), address(address),
maxBooksAllowed(maxBooksAllowed) {}
 //default parameterized constructor of member


 bool canIssueMoreBooks() {
```

```
 return issuedBooks.size() < maxBooksAllowed;
 }
};

class Student : public Member {
public:
 Student(string memberId, string name, string email, string address)
 : Member(memberId, name, email, address, 2) {}
};

class Faculty : public Member {
public:
 Faculty(string memberId, string name, string email, string address)
 : Member(memberId, name, email, address, 10) {}
};

class Transaction {
public:
 string memberId;
 string bookId;
 int serialNumber;
 time_t date;
 bool isReturned;

 Transaction(string memberId, string bookId, int serialNumber, time_t date, bool isReturned)
 : memberId(memberId), bookId(bookId), serialNumber(serialNumber), date(date),
isReturned(isReturned) {}
};

class Temp {
 map<string, Book*> books;
 map<string, Member*> members;
 vector<Transaction> transactions;

public:
 void addBook(Book* book) {
 books[book->bookId + "-" + to_string(book->serialNumber)] = book;
 }

 void addMember(Member* member) {
 members[member->memberId] = member;
 }

 void issueBook(const string& memberId, const string& bookId) {
```

```
Member* member = members[memberId];
if (!member) {
cout << "Invalid member ID." << endl;
return;
}

if (!member->canIssueMoreBooks()) {
cout << "Member has reached the maximum limit of issued books." << endl;
return;
}

for (auto& [key, book] : books) {
if (book->bookId == bookId && !book->isIssued) {
book->isIssued = true;
member->issuedBooks.push_back(book);
transactions.emplace_back(memberId, bookId, book->serialNumber, time(0), false);
cout << "Book issued successfully." << endl;
return;
}
}
cout << "Book is not available." << endl;
}

void returnBook(const string& memberId, const string& bookId, int serialNumber) {
Member* member = members[memberId];
if (!member) {
cout << "Invalid member ID." << endl;
return;
}

Book* book = books[bookId + "-" + to_string(serialNumber)];
if (!book || !book->isIssued) {
cout << "Invalid book ID or serial number." << endl;
return;
}

auto it = find(member->issuedBooks.begin(), member->issuedBooks.end(), book);
if (it == member->issuedBooks.end()) {
cout << "This book was not issued to this member." << endl;
return;
}

book->isIssued = false;
member->issuedBooks.erase(it);
```

```
 for (auto& transaction : transactions) {
 if (transaction.memberId == memberId && transaction.bookId == bookId &&
 transaction.serialNumber == serialNumber && !transaction.isReturned) {
 transaction.isReturned = true;
 break;
 }
 }
 cout << "Book returned successfully." << endl;
 }
};

int main() {
 Temp library;
 int choice;
 while (true) {
 cout << "1. Add Book\n2. Add Member\n3. Issue Book\n4. Return Book\n5. Exit\nEnter your choice: ";
 cin >> choice;
 switch (choice) {
 case 1: {
 string bookId, title, author, publisher;
 int serialNumber;
 double price;
 cout << "Enter Book ID: ";
 cin >> bookId;
 cout << "Enter Serial Number: ";
 cin >> serialNumber;
 cout << "Enter Title: ";
 cin >> title;
 cout << "Enter Author: ";
 cin >> author;
 cout << "Enter Publisher: ";
 cin >> publisher;
 cout << "Enter Price: ";
 cin >> price;
 library.addBook(new Book(bookId, serialNumber, title, author, publisher, price));
 break;
 }
 case 2: {
 string memberId, name, email, address;
 int type;
 cout << "Enter Member ID: ";
 cin >> memberId;
 cout << "Enter Name: ";
 cin >> name;
```

```
cout << "Enter Email: ";
cin >> email;
cout << "Enter Address: ";
cin >> address;
cout << "Enter 1 for Student, 2 for Faculty: ";
cin >> type;
if (type == 1) {
library.addMember(new Student(memberId, name, email, address));
} else if (type == 2) {
library.addMember(new Faculty(memberId, name, email, address));
} else {
cout << "Invalid member type." << endl;
}
break;
}
case 3: {
string memberId, bookId;
cout << "Enter Member ID: ";
cin >> memberId;
cout << "Enter Book ID: ";
cin >> bookId;
library.issueBook(memberId, bookId);
break;
}
case 4: {
string memberId, bookId;
int serialNumber;
cout << "Enter Member ID: ";
cin >> memberId;
cout << "Enter Book ID: ";
cin >> bookId;
cout << "Enter Serial Number: ";
cin >> serialNumber;
library.returnBook(memberId, bookId, serialNumber);
break;
}
case 5:
return 0;
default:
cout << "Invalid choice." << endl;
}
}
```

```
 return 0;
}
```

```
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
1
Enter roll number: 30
Enter name: Tathagata
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
2
Enter subject code: 1
Enter subject name: Physics
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
3
Enter roll number: 30
Enter subject code: 1
Enter your choice:
1. Add Student
2. Add Subject
3. Enroll Student in Subject
4. Get Subjects by Student
5. Get Students by Subject
6. Exit
5
Enter subject code: 1
Roll number: 30, Name: Tathagata
```

Q3. Employee has unique emp-id, name, designation and basic pay. An employee is either a permanent one or contractual. For permanent employee salary is computed as basic pay+ hra (30% of basic pay) + da (80% of basic pay). For contractual employee it is basic pay + allowance (it is different for different contractual employee). An employee pointer may point to either of the two categories and accordingly the salary has to be created. Design the classes and implement

```
#include <bits/stdc++.h>
using namespace std;

class Employee {
protected:
 string empId;
 string name;
 string designation;
 double basicPay;

public:
 Employee(string empId, string name, string designation, double basicPay)
 : empId(empId), name(name), designation(designation), basicPay(basicPay) {}

 string getEmpId() const {
 return empId;
 }

 string getName() const {
 return name;
 }

 string getDesignation() const {
 return designation;
 }

 double getBasicPay() const {
 return basicPay;
 }

 virtual double calculateSalary() const = 0;
 virtual ~Employee() = default;
};

class PermanentEmployee : public Employee {
```

```
public:
 PermanentEmployee(string empId, string name, string designation, double basicPay)
 : Employee(empId, name, designation, basicPay) {}

 double calculateSalary() const override {
 return basicPay *2.1;
 }
};

class ContractualEmployee : public Employee {
private:
 double allowance;

public:
 ContractualEmployee(string empId, string name, string designation, double basicPay, double
allowance)
 : Employee(empId, name, designation, basicPay), allowance(allowance) {}

 double calculateSalary() const override {
 return basicPay + allowance;
 }
};

int main() {
 vector<shared_ptr<Employee>> employees;
 int choice;

 while (true) {
 cout << "1. Add Permanent Employee" << endl;
 cout << "2. Add Contractual Employee" << endl;
 cout << "3. Display Salaries" << endl;
 cout << "4. Exit" << endl;
 cout << "Enter your choice: ";
 cin >> choice;

 if (choice == 1) {
 string empId, name, designation;
 double basicPay;
 cout << "Enter empId: ";
 cin >> empId;
 cout << "Enter name: ";
 cin >> name;
 cout << "Enter designation: ";
 cin >> designation;
```

```
 cout << "Enter basic pay (in double format): ";
 cin >> basicPay;
 employees.push_back(make_shared<PermanentEmployee>(empId, name, designation, basicPay));
 }
 else if (choice == 2) {
 string empId, name, designation;
 double basicPay, allowance;
 cout << "Enter empId: ";
 cin >> empId;
 cout << "Enter name: ";
 cin >> name;
 cout << "Enter designation: ";
 cin >> designation;
 cout << "Enter basic pay (in double format): ";
 cin >> basicPay;
 cout << "Enter allowance (in double format): ";
 cin >> allowance;
 employees.push_back(make_shared<ContractualEmployee>(empId, name, designation, basicPay,
allowance));
 }
 else if (choice == 3) {
 for (const auto& employee : employees) {
 cout << "EmpId: " << employee->getEmpId()
 << ", Name: " << employee->getName()
 << ", Designation: " << employee->getDesignation()
 << ", Salary: " << employee->calculateSalary() << endl;
 }
 }
 else if (choice == 4) {
 break;
 }
 else {
 cout << "Invalid choice. Please try again." << endl;
 }
 }
 return 0;
}
```

```
1. Add Permanent Employee
2. Add Contractual Employee
3. Display Salaries
4. Exit
Enter your choice: 1
Enter empId: 123
Enter name: Alice
Enter designation: Manager
Enter basic pay (in double format): 100000
1. Add Permanent Employee
2. Add Contractual Employee
3. Display Salaries
4. Exit
Enter your choice: 2
Enter empId: 100
Enter name: Bob
Enter designation: HR
Enter basic pay (in double format): 100000
Enter allowance (in double format): 25000
1. Add Permanent Employee
2. Add Contractual Employee
3. Display Salaries
4. Exit
Enter your choice: 3
EmpId: 123, Name: Alice, Designation: Manager, Salary: 210000
EmpId: 100, Name: Bob, Designation: HR, Salary: 125000
1. Add Permanent Employee
2. Add Contractual Employee
3. Display Salaries
4. Exit
Enter your choice:
```

Q4. Each cricketer has name, date of birth and matches played. Cricketer may be a bowler or batsman. For a bowler, number of wickets taken, average economy is stored. For a batsman, total runs scored, average score is stored. A double wicket pair is formed taking a bowler and a batsman. An all-rounder is both a bowler and batsman. Support must be there to show the details of a cricketer, bowler, batsmen, all-rounder and the pair. Design the classes and implement.

```
#include <bits/stdc++.h>
using namespace std;

class Cricketer {
protected:
string name;
string dateOfBirth;
int matchesPlayed;

public:
 Cricketer(const string &name, const string &dateOfBirth, int matchesPlayed)
```

```
 : name(name), dateOfBirth(dateOfBirth), matchesPlayed(matchesPlayed) {}

 virtual void showDetails() const {
 cout << "Name: " << name << endl;
 cout << "Date of Birth: " << dateOfBirth << endl;
 cout << "Matches Played: " << matchesPlayed << endl;
 }

 virtual ~Cricketer() = default;  //Using = default makes it clear that the destructor is intentionally virtual and uses the compiler-generated implementation.
};

class Bowler : public Cricketer {
 int wicketsTaken;
 double averageEconomy;

public:
 Bowler(const string &name, const string &dateOfBirth, int matchesPlayed, int wicketsTaken, double averageEconomy)
 : Cricketer(name, dateOfBirth, matchesPlayed), wicketsTaken(wicketsTaken), averageEconomy(averageEconomy) {}

 void showDetails() const override {
 Cricketer::showDetails();
 cout << "Wickets Taken: " << wicketsTaken << endl;
 cout << "Average Economy: " << averageEconomy << endl;
 }
};

class Batsman : public Cricketer {
 int totalRuns;
 double averageScore;

public:
 Batsman(const string &name, const string &dateOfBirth, int matchesPlayed, int totalRuns, double averageScore)
 : Cricketer(name, dateOfBirth, matchesPlayed), totalRuns(totalRuns), averageScore(averageScore) {}

 void showDetails() const override {
 Cricketer::showDetails();
 cout << "Total Runs: " << totalRuns << endl;
 cout << "Average Score: " << averageScore << endl;
 }
};
```

```
class AllRounder : public Cricketer {
 int wicketsTaken;
 double averageEconomy;
 int totalRuns;
 double averageScore;

public:
 AllRounder(const string &name, const string &dateOfBirth, int matchesPlayed, int wicketsTaken,
double averageEconomy, int totalRuns, double averageScore)
 : Cricketer(name, dateOfBirth, matchesPlayed), wicketsTaken(wicketsTaken),
averageEconomy(averageEconomy), totalRuns(totalRuns), averageScore(averageScore) {}

 void showDetails() const override {
 Cricketer::showDetails();
 cout << "Wickets Taken: " << wicketsTaken << endl;
 cout << "Average Economy: " << averageEconomy << endl;
 cout << "Total Runs: " << totalRuns << endl;
 cout << "Average Score: " << averageScore << endl;
 }
};

class DoubleWicketPair {
 Bowler *bowler;
 Batsman *batsman;

public:
 DoubleWicketPair(Bowler *bowler, Batsman *batsman) : bowler(bowler), batsman(batsman) {}

 void showDetails() const {
 cout << "Double Wicket Pair:" << endl;
 cout << "Bowler Details:" << endl;
 bowler->showDetails();
 cout << "Batsman Details:" << endl;
 batsman->showDetails();
 }
};

int main() {
 vector<unique_ptr<Cricketer>> cricketers;
 int choice;

 while (true) {
 cout << "Menu:\n1. Add Bowler\n2. Add Batsman\n3. Add All-Rounder\n4. Show Cricketer Details\n5.
```

```
Show Double Wicket Pair Details\n6. Exit\nEnter your choice: ";
 cin >> choice;

 if (choice == 6) break;

 switch (choice) {
 case 1: {
 string name, dob;
 int matches, wickets;
 double economy;
 cout << "Enter name: "; cin >> ws; getline(cin, name);
 cout << "Enter date of birth: "; getline(cin, dob);
 cout << "Enter matches played: "; cin >> matches;
 cout << "Enter wickets taken: "; cin >> wickets;
 cout << "Enter average economy: "; cin >> economy;
 cricketers.push_back(make_unique<Bowler>(name, dob, matches, wickets, economy));
 break;
 }
 case 2: {
 string name, dob;
 int matches, runs;
 double score;
 cout << "Enter name: "; cin >> ws; getline(cin, name);
 cout << "Enter date of birth: "; getline(cin, dob);
 cout << "Enter matches played: "; cin >> matches;
 cout << "Enter total runs: "; cin >> runs;
 cout << "Enter average score: "; cin >> score;
 cricketers.push_back(make_unique<Batsman>(name, dob, matches, runs, score));
 break;
 }
 case 3: {
 string name, dob;
 int matches, wickets, runs;
 double economy, score;
 cout << "Enter name: "; cin >> ws; getline(cin, name);
 cout << "Enter date of birth: "; getline(cin, dob);
 cout << "Enter matches played: "; cin >> matches;
 cout << "Enter wickets taken: "; cin >> wickets;
 cout << "Enter average economy: "; cin >> economy;
 cout << "Enter total runs: "; cin >> runs;
 cout << "Enter average score: "; cin >> score;
 cricketers.push_back(make_unique<AllRounder>(name, dob, matches, wickets, economy, runs,
score));
 break;
```

```
}
case 4: {
for (const auto &cricketer : cricketers) {
cricketer->showDetails();
cout << endl;
}
break;
}
case 5: {
int bowlerIndex, batsmanIndex;
cout << "Enter bowler index: "; cin >> bowlerIndex;
cout << "Enter batsman index: "; cin >> batsmanIndex;

if (bowlerIndex < cricketers.size() && batsmanIndex < cricketers.size() &&
dynamic_cast<Bowler*>(cricketers[bowlerIndex].get()) &&
dynamic_cast<Batsman*>(cricketers[batsmanIndex].get())) {

DoubleWicketPair pair(
dynamic_cast<Bowler*>(cricketers[bowlerIndex].get()),
dynamic_cast<Batsman*>(cricketers[batsmanIndex].get())
);
pair.showDetails();
} else {
cout << "Invalid indices or types." << endl;
}
break;
}
default:
cout << "Invalid choice. Please try again." << endl;
}
}
return 0;
}
```

```
1. Add Bowler
2. Add Batsman
3. Add All-Rounder
4. Show Cricketer Details
5. Show Double Wicket Pair Details
6. Exit
Enter your choice: 1
Enter name: Alice
Enter date of birth: 23/10/2002
Enter matches played: 500
Enter wickets taken: 678
Enter average economy: 6.7
Menu:
1. Add Bowler
2. Add Batsman
3. Add All-Rounder
4. Show Cricketer Details
5. Show Double Wicket Pair Details
6. Exit
Enter your choice: 2
Enter name: Bob
Enter date of birth: 24/10/2003
Enter matches played: 500
Enter total runs: 15000
Enter average score: 35.87
Menu:
1. Add Bowler
2. Add Batsman
3. Add All-Rounder
4. Show Cricketer Details
5. Show Double Wicket Pair Details
6. Exit
Enter your choice: 5
Enter bowler index: 0
Enter batsman index: 1
Double Wicket Pair:
Bowler Details:
Name: Alice
Date of Birth: 23/10/2002
Matches Played: 500
Wickets Taken: 678
Average Economy: 6.7
Batsman Details:
Name: Bob
Date of Birth: 24/10/2003
Matches Played: 500
Total Runs: 15000
Average Score: 35.87
```

Assignment 6

Q1. In a library, for each book book-id, serial number (denotes copy number of a book), title, author, publisher and price are stored. Book-id and serial number together will be unique identifier for a book. Members are either student or faculty. Each member has unique member-id. Name, e-mail, address are also to be stored. For any transaction (book issue or return), members are supposed to place transactions slip. User will submit member-id, book-id, and serial number (only for book return). While processing a transaction, check the validity of the member. While issuing, availability of a copy of the book is to be checked. While returning a book, it is to be checked whether this copy was issued to the member or not. A student member can have 2 books issued at a point of time. For faculty members it is 10. Transaction information is to be stored like date of transaction, member-id, book-id, serial number, returned or not. An entry is made when book is issued and updated when the book is returned. For storing the information consider files.

Design the classes and implement

```
#include <bits/stdc++.h>

using namespace std;


class Book {

public:

   string bookId;

   int serialNumber;

   string title;

   string author;

   string publisher;

   double price;

   // Default constructor

   Book() : serialNumber(0), price(0.0) {}
```

```
    // Parameterized constructor
    Book(string bookId, int serialNumber, string title, string author, string publisher, double price) :bookId(bookId), serialNumber(serialNumber), title(title), author(author), publisher(publisher), price(price) {}
};

class Member {
public:
    string memberId;
    string name;
    string email;
    string address;
    int maxBooksAllowed;
    vector<string> issuedBooks;

Member(string memberId, string name, string email, string address, int maxBooksAllowed): memberId(memberId), name(name), email(email), address(address), maxBooksAllowed(maxBooksAllowed) {}

    bool canIssueMoreBooks() const {
        return issuedBooks.size() < maxBooksAllowed;
    }
    void issueBook(const string& bookId) {
        issuedBooks.push_back(bookId);
    }
    void returnBook(const string& bookId) {
issuedBooks.erase(remove(issuedBooks.begin(),issuedBooks.end(),bookId),issuedBooks.end();
```

```
    }
};

class Student : public Member {
public:
    Student(string memberId, string name, string email, string address)
        : Member(memberId, name, email, address, 2) {}
};

class Faculty : public Member {
public:
    Faculty(string memberId, string name, string email, string address)
        : Member(memberId, name, email, address, 10) {}
};

class Transaction {
public:
    string memberId;
    string bookId;
    int serialNumber;
    time_t date;
    bool isReturned;

    Transaction(string memberId, string bookId, int serialNumber, time_t date, bool isReturned)
        : memberId(memberId), bookId(bookId), serialNumber(serialNumber), date(date), isReturned(isReturned) {}
```

```
};

class Library {
    map<string, Book> books;
    map<string, Member*> members;
    list<Transaction> transactions;

public:
    void addBook(const Book& book) {
        string key = book.bookId + "-" + to_string(book.serialNumber);
        books[key] = book;
    }

    void addMember(Member* member) {
        members[member->memberId] = member;
    }

    bool isBookAvailable(const string& bookId, int serialNumber) const {
        string key = bookId + "-" + to_string(serialNumber);
        return books.find(key) != books.end();
    }

    bool isMemberValid(const string& memberId) const {
        return members.find(memberId) != members.end();
    }
```

```
void issueBook(const string& memberId, const string& bookId, int serialNumber) {
    if (!isMemberValid(memberId)) {
        cout << "Invalid member." << endl;
        return;
    }

    Member* member = members[memberId];
    if (!member->canIssueMoreBooks()) {
        cout << "Member has reached the maximum limit of issued books." << endl;
        return;
    }

    if (!isBookAvailable(bookId, serialNumber)) {
        cout << "Book is not available." << endl;
        return;
    }

    member->issueBook(bookId);
    transactions.push_back(Transaction(memberId, bookId, serialNumber, time(0), false));
    cout << "Book issued successfully." << endl;
}

void returnBook(const string& memberId, const string& bookId, int serialNumber) {
    if (!isMemberValid(memberId)) {
```

```
            cout << "Invalid member." << endl;
            return;
        }

        Member* member = members[memberId];
        auto it = find(member->issuedBooks.begin(), member->issuedBooks.end(), bookId);
        if (it == member->issuedBooks.end()) {
            cout << "This book was not issued to the member." << endl;
            return;
        }

        member->returnBook(bookId);
        for (auto& transaction : transactions) {
                    if (transaction.memberId == memberId && transaction.bookId == bookId &&
transaction.serialNumber == serialNumber && !transaction.isReturned) {
                transaction.isReturned = true;
                cout << "Book returned successfully." << endl;
                return;
            }
        }

        cout << "Transaction not found." << endl;
    }
};

int main() {
```

```
Library library;
int choice;
string memberId, bookId, name, email, address, title, author, publisher;
int serialNumber, memberType;
double price;
while (true) {
    cout << "Library Management System" << endl;
    cout << "1. Add Book" << endl;
    cout << "2. Add Member" << endl;
    cout << "3. Issue Book" << endl;
    cout << "4. Return Book" << endl;
    cout << "5. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;
    cin.ignore();

    switch (choice) {
        case 1:
            cout << "Enter book ID: ";
            getline(cin, bookId);
            cout << "Enter serial number: ";
            cin >> serialNumber;
            cin.ignore();
            cout << "Enter title: ";
            getline(cin, title);
```

```
        cout << "Enter author: ";
        getline(cin, author);
        cout << "Enter publisher: ";
        getline(cin, publisher);
        cout << "Enter price: ";
        cin >> price;
        library.addBook(Book(bookId, serialNumber, title, author, publisher, price));
        cout << "Book added successfully." << endl;
        break;
    case 2:
        cout << "Enter member ID: ";
        getline(cin, memberId);
        cout << "Enter name: ";
        getline(cin, name);
        cout << "Enter email: ";
        getline(cin, email);
        cout << "Enter address: ";
        getline(cin, address);
        cout << "Enter member type (1 for Student, 2 for Faculty): ";
        cin >> memberType;
        cin.ignore();
        if (memberType == 1) {
            library.addMember(new Student(memberId, name, email, address));
        } else if (memberType == 2) {
            library.addMember(new Faculty(memberId, name, email, address));
```

```
        } else {
            cout << "Invalid member type." << endl;
        }
        cout << "Member added successfully." << endl;
        break;
    case 3:
        cout << "Enter member ID: ";
        getline(cin, memberId);
        cout << "Enter book ID: ";
        getline(cin, bookId);
        cout << "Enter serial number: ";
        cin >> serialNumber;
        library.issueBook(memberId, bookId, serialNumber);
        break;
    case 4:
        cout << "Enter member ID: ";
        getline(cin, memberId);
        cout << "Enter book ID: ";
        getline(cin, bookId);
        cout << "Enter serial number: ";
        cin >> serialNumber;
        library.returnBook(memberId, bookId, serialNumber);
        break;
    case 5:
        cout << "Exiting..." << endl;
```

```
        return 0;
    default:
        cout << "Invalid choice." << endl;
}}}
```

```
1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Exit
Enter your choice: 1
Enter book ID: 123
Enter serial number: 10000
Enter title: CSE
Enter author: Alice
Enter publisher: JU
Enter price: 10000
Book added successfully.
Library Management System
1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Exit
Enter your choice: 2
Enter member ID: 1
Enter name: Bob
Enter email: ju@gmail.com
Enter address: Jadavpur
Enter member type (1 for Student, 2 for Faculty): 2
Member added successfully.
Library Management System
1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Exit
Enter your choice: 3
Enter member ID: 1
Enter book ID: 123
Enter serial number: 10000
Book issued successfully.
Library Management System
1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Exit
Enter your choice: 4
Enter member ID: 1
Enter book ID: 123
Enter serial number: 10000
Book returned successfully.
```

Q2. Consider a class Student with roll, name and score as attributes. Support to take and display data is also there. One wants to works with array of Student objects. May collect data for array elements, display those. In case index goes out of bounds, exception is to be raised with suitable message.

```
#include <bits/stdc++.h>

using namespace std;

class Student {

private:

   int roll;

   string name;

   double score;


public:

   // Constructor

   Student(int roll, const std::string &name, double score)

      : roll(roll), name(name), score(score) {}

   // Method to display student data

   void display() const {

      cout << "Roll: " << roll << ", Name: " << name << ", Score: " << score << endl;

   }

};


int main() {

   int n;

   cout << "Enter number of students: ";

   cin >> n;
```

```
// Dynamic array of students
vector<Student*> students(n, nullptr);  // Initially filled with nullptrs
int choice = 0;

while (choice != 3) {
   cout << "Choose 1 to enter data, 2 to display data, otherwise exit: ";
   cin >> choice;

   switch (choice) {
      case 1: {
         int i;
         cout << "Enter student number (between 1 and " << n << "): ";
         cin >> i;
         --i;          // Adjusted for 0-based indexing

        if (i < 0 || i >= n) {
            cerr << "Error: Index out of bounds" << endl;
            break;
         }              // Checking if index is within bounds
         // Collecting student data
         int roll;
         string name;
         double score;
         cout << "Enter roll, name and score for student:" << endl;
```

```
            cin >> roll;
            cin.ignore();  // Ignore leftover newline character
            getline(cin, name);
            cin >> score;
            students[i] = new Student(roll, name, score);
            break;
        }

        case 2: {
            int j;
            std::cout << "Enter student number (between 1 and " << n << ") whose detail you wish to display: ";
            std::cin >> j;
            --j;
            if (j < 0 || j >= n || students[j] == nullptr) {
                cerr << "Error: Index out of bounds or student data not available" << endl;
                break;
            }      // Check if index is within bounds and if the student is not nullptr
            students[j]->display();
            break;
        }

        default:
            choice = 3;
    }
}
```

```
    // Cleanup dynamically allocated Student objects
    for (Student* student : students) {
        delete student;
    }

    return 0;
}
```

```
Enter number of students: 2
Choose 1 to enter data, 2 to display data, otherwise exit: 1
Enter student number (between 1 and 2): 1
Enter roll, name and score for student:
30
Alice
95
Choose 1 to enter data, 2 to display data, otherwise exit: 1
Enter student number (between 1 and 2): 2
Enter roll, name and score for student:
20
Bob
99
Choose 1 to enter data, 2 to display data, otherwise exit: 2
Enter student number (between 1 and 2) whose detail you wish to display: 1
Roll: 30, Name: Alice, Score: 95
Choose 1 to enter data, 2 to display data, otherwise exit: 2
Enter student number (between 1 and 2) whose detail you wish to display: 2
Roll: 20, Name: Bob, Score: 99
Choose 1 to enter data, 2 to display data, otherwise exit: 1
Enter student number (between 1 and 2): 5
ERROR!
Error: Index out of bounds
```

Q3. Implement a class template for 1D array. Elements may be any basic data type. Provision to find maximum element, sum of the elements must be there.

```
#include <bits/stdc++.h>

using namespace std;
```

```
template <typename T>
class ArrayTemplate {
private:
    vector<T> array;

public:
    ArrayTemplate(const vector<T>& arr) : array(arr) {}

    T findMax() const {
        if (array.empty()) {
            cout << "Array is empty.\n";
            return T();              // Default value for T
        }
        T max = array[0];
        for (const T& element : array) {
            if (element > max) {
                max = element;
            }
        }
        return max;
    }

    double sum() const {
        double sum = 0;
```

```
        for (const T& element : array) {
            sum += static_cast<double>(element);
        }
        return sum;
    }
};

int main() {
    int choice;
    cout << "Choose data type:\n1. Integer\n2. Double\nOthers: Exit\n";
    cin >> choice;

    while (choice == 1 || choice == 2) {
        if (choice == 1) {
            int n;
            cout << "Enter the number of elements: ";
            cin >> n;
            vector<int> intArrayInput(n);
            cout << "Enter the elements:\n";
            for (int i = 0; i < n; i++) {
                cin >> intArrayInput[i];
            }
            ArrayTemplate<int> intTemplateInput(intArrayInput);
            cout << "Max: " << intTemplateInput.findMax() << endl;
            cout << "Sum: " << intTemplateInput.sum() << endl;
```

```
    } else if (choice == 2) {
        int m;
        cout << "Enter the number of elements: ";
        cin >> m;
        vector<double> doubleArrayInput(m);
        cout << "Enter the elements:\n";
        for (int i = 0; i < m; i++) {
            cin >> doubleArrayInput[i];
        }
        ArrayTemplate<double> doubleTemplateInput(doubleArrayInput);
        cout << "Max: " << doubleTemplateInput.findMax() << endl;
        cout << "Sum: " << doubleTemplateInput.sum() << endl;
    }

    cout << "Choose data type:\n1. Integer\n2. Double\nOthers: Exit\n";
    cin >> choice;
  }

  cout << "Exiting...\n";
  return 0;
}
```

```
Choose data type:
1. Integer
2. Double
Others: Exit
1
Enter the number of elements: 2
Enter the elements:
1 3
Max: 3
Sum: 4
Choose data type:
1. Integer
2. Double
Others: Exit
2
Enter the number of elements: 3
Enter the elements:
4.5
24.5
67.8
Max: 67.8
Sum: 96.8
Choose data type:
1. Integer
2. Double
Others: Exit
5
Exiting...


=== Code Execution Successful ===
```