Product* arr[3]

```
class Product{
id
title
price

virtual void accept(){
id,title,price
}
virtual getDiscountedPrice(){

}
}
```

```
class Book:public Product{
author

    void accept(){
    Product::accept()
    author
    }
}
```

```
class Tape:public Product{
artist

    void accept(){
    Product::accept()
    artist
    }
}
```

```
claculateBill(Product **arr){
double totalBill=0;
for(.....){
if(typeid(*Person[i])==typeid(Book)){
//book
double discountedprice;
discountedPrice=Person[i]->getPrice() - Person[i]->getPrice() * 0.1
totalBill += discountedPrice;
}
else{
//tape
double discountedprice;
}

}

}
```

Q2

OOP
Major
Minor

```
if(denominator ==0){
}
else{
logic
}

if(user.hasloggedin){
}
else{

}
```

try
- To check for the statemets within it if they are throwing any exceptions

catch
- To handle the thrown exceptions

throw
- To generate an exception

```
class Exception{
string message;

public:
Exception():Exception(""){}
Exception(string message):message(message){}
virtual void printStackTrace(){
cout<<"Exception : Exception"<<endl;
cout<<"Message : "<<message<<endl;
}
}

class InvalidTimeException:public Exception{
public:

InvalidTimeException(){}
InvalidTimeException(strig message):Exception(message){}
}

class InvalidDateException:public Exception{
public:

InvalidDateException(){}
InvalidDateException(strig message):Exception(message){}
}
```

Templates

- Data Structure Algorithms

```
T -> int
T->double
T-> Employee
void swap(T n1,T n2){
T temp = n1;
n1 = n2;
n2 = temp;
}
```

```
double->%
Grade-> 'A'
```

```
int -> years
Grade->'A'
```

map<>

```
class Point1{
int x;
int y;

};
```

```
class Point2{
double x;
double y;

};
```

```
class Point3{
char x;
int y;

};
```

```
class Point4{
char x;
double y;

};
```

```
class Point5{
int x;
double y;

};
```

```
class Point6{
double x;
int y;

};
```

```
class Point7{
int x;
char y;

};
```

```
class Point8{
double x;
char y;

};
```

Datastructure
- array
- linkedlist
- queue
- stack

stack
int
double
char
employee
product
student

```
struct stack{
int data;
int top;
}
push();
pop();
```

```
template<typename T>
class stack{
T *data;
int top;

push();
pop();
}
```

```
class Node{
Node *prev;
T data
Node *next;
}
```

Templates - (STL) - Standard Template Library
Generics - Collection

STL
File I/O
shallow deep
Copy
Design pattern
operator overloading
conversion function
->

1. Menu Driven
2. Class -> Association and Inheritance
3. Logic Building
4. STL and File IO
5. Exception Handling
6. comments, namingconvention, modularity

Friend Function

- It is a non member function of the class which can access the private and protected members of the class directly on class object.
- Friend functions do not get this pointer.

Friend class

- It is a class if declared as a friend inside another class can access its private and protected members directly on class object.