# AS 4048

B. Tech III rd sem CSE

OOPs with C++ (CS 2101)

## Section A

(i) - (C).
(ii) - (D).
(iii) - (A).
(iv) - (B).
(v) - (A).
(vi) - (C).
(vii) - (B).
(viii) - (C).
(ix) - (B).
(x) - (D).

## Section B

→ An example to each Question of section B which has been Attempted.

Note: Copies can be seen as per instruction and schedule.

# UNIT I

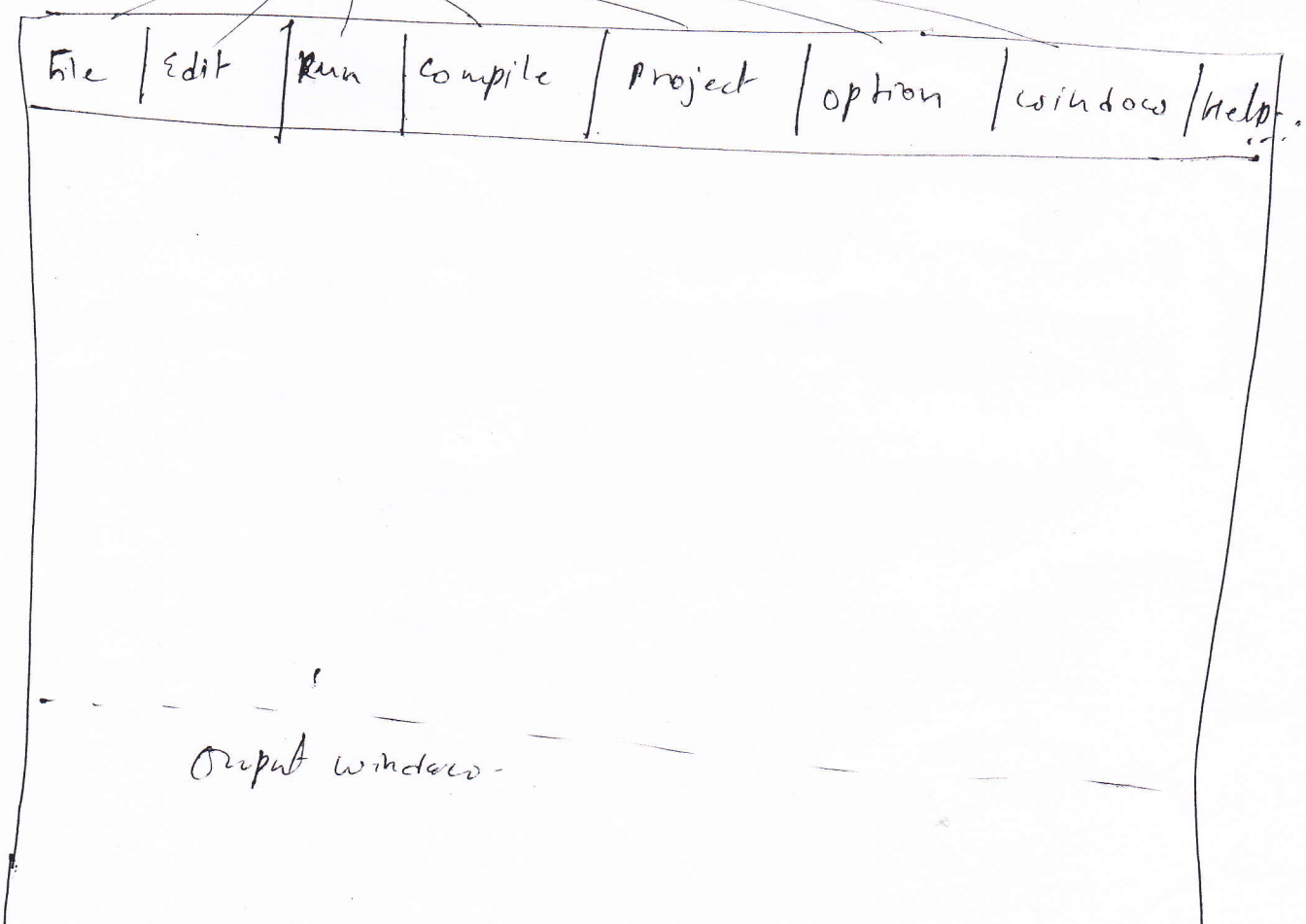
Q.2 Following unique advantages with explanation of each

- Relationship to the real world
- New data types
- object & class
- Reusability
- Data Abstraction
- Encapsulation
- Polymorphism & overloading.
- Message passing
- Inheritence
- Bottom up approach

(14 points)

- elimination of Redundent work
- data hiding
- data entered design approach
- reduced complexity
- Instances based.

Q.3 explanation of each tools (& sub tools) available for development.



Ans 3 New Header file.

step 1 : Open Dos editor or Notepad.

step 2 : write the following code in it

```
void cube ( )
{  int a;
cout << "enter value to cube ";
cin >> a;
  cout << (a*a*a) ; //or return if return type
                      is given
}
```

step 3 : save this file with .H extension like cube.H

step 4 : set the path / save file into include folder ~~TC/Include~~ (TC/Include).

step 5 : Include this Header file as

```
#include <cube.h>
```

step 6 : Use function cube by calling it.

→ A complete example.

# UNIT II

Q 5

| LHO \ RHO | char | short | int | long | float | double | long double |
|---|---|---|---|---|---|---|---|
| char | int | int | int | long | float | double | long double |
| short | int | int | int | long | float | double | long double |
| int | int | int | int | long | float | double | long double |
| long | long | long | long | long | float | double | long double |
| float | float | float | float | float | float | double | long double |
| double | double | double | double | double | double | double | long double |
| long | long double | long double | long double | long double | long double | long double | long double |

Q. 6.

Q 6 Operator precedence

1. Unary operators
2. Arithmetic Multiply, divide, remainder
3. Arithmetic Add, subtract
4. Relational
5. Equality
6. Logical and
7. Logical or
8. Conditional.
9. Assignment.

| Operator | Associativity |
|---|---|
| :: | L→R |
| → , . , ( ) , [ ] , postfix ++ , postfix -- | L→R |
| Prefix ++ , -- , ~ , ! unary +, - , * , & (type) , sizeof , new , delete | R→L |
| →* , * , * | L→R |
| * , / , % | L→R |
| + - | L→R |
| << , >> | L→R |
| <<= , >>= | L→R |
| == , != | L→R |
| & | L→R |
| ^ | L→R |
| \| | L→R |
| && | L→R |
| \|\| | L→R |
| ?: | L-R |

Q 7 Dynamic Memo. Alloc Operators

(i) pointers used in the proging have the dynamic property (i.e) data stored during runtime.

(ii) five areas of memory to store variables, stack, code space, register, globable name space & free store (Heap)

(iii) The local & global variables are allocated memory during compile time. These variables cannot be added during runtime. The main problem with the local variable they dont persist when function returns, the local variable thrown away. The global variable solved this problem at the cost of unrestricted access. which leads difficult to understand and maintain.

(iv) Heap is used when whenever a program needs to use variable amount of memory. The free store memory can be allocate & deallocate by two unary operator "new" & "delete"; Mallock & alloc,

New

pointer variable = new data-type

int * ptr = new int.;

initialization

*ptr = 85;

pointer variable = new data-type [size]

delete

delete pointer_variable;

delete ptr;

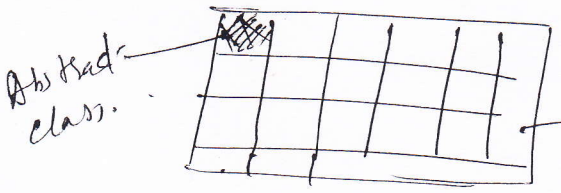
when you call delete on a pointer, the memory it points to is free. calling delete on that pointer again will crash your program

# UNIT III

Q. 8

Abstract classes

- Not used to create objects.
- designed only to act as a base class to be inherited by other subclass.
- design concept as index class for all rest of classes




storage classes

Automatic, Register, Extern, static

| storage type | Automatic | Static | External. | Reg |
|---|---|---|---|---|
| Visibility | Function | Function | File | \| |
| Lifetime | Function | prog | Prog. | Same as auto mati |
| Initialize value | not initialize | 0 | 0 | |
| storage | stack | data seg ment. | data segment. | Regist |
| purpose | variables used by a single funct | Same as auto but must retain value when function terminates | variables used by several functions | same as Auto mati |

Q9

## Pure virtual function

→ Fulfills the req. of Compiler
→ This is null function that is not called.
→ It follows the concept of polymorphism (~~dynamic~~).
→ Only declared not defined in base class
→ Virtual function with no body, no argument
→ Two form available (as below)
→ Combining the common features of a series of special classes in ~~the~~ a base class, not used to create object
→ Gives designers flexibility to free to defer specific implementation details until later and to concentrate on the issues of basic design
→ Member of same class
→ can be accessed by pointer of base class for derived class.
→ form 1 (when scope resolution operator are not used)

```
class abc
{
    virtual void show() = 0;

};
```

form 2

```
class first
{   virtual void getdata();

};
void first :: getdata()
{
=
}
```

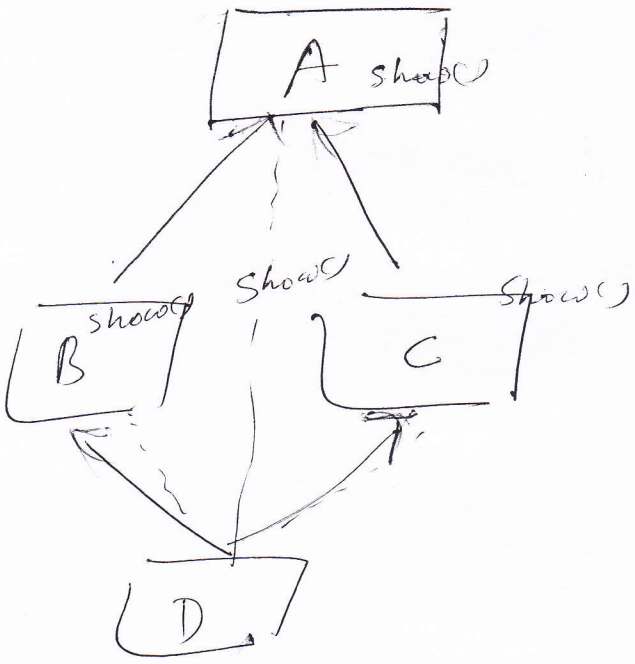
Q. 10 Returning values from function

(1) by value
(2) by reference
(3) by pointer

→ All above with an example to each.

# UNIT IV

Q. 11.




- The duplication of the member function in derived class can be avoided by making a common base class as virtual class. It resolve the ambiguity in Inheritance.

Syntax

```
class base A
{
   Body of base A;
}
class der B : virtual public base A
{
   body of der B;
}
class der C : public virtual A
  { Body of der C;
  }
class der D : public der B, public der C
{
  body of der D;
}
```

- One complete example of it with main()

Ans 12

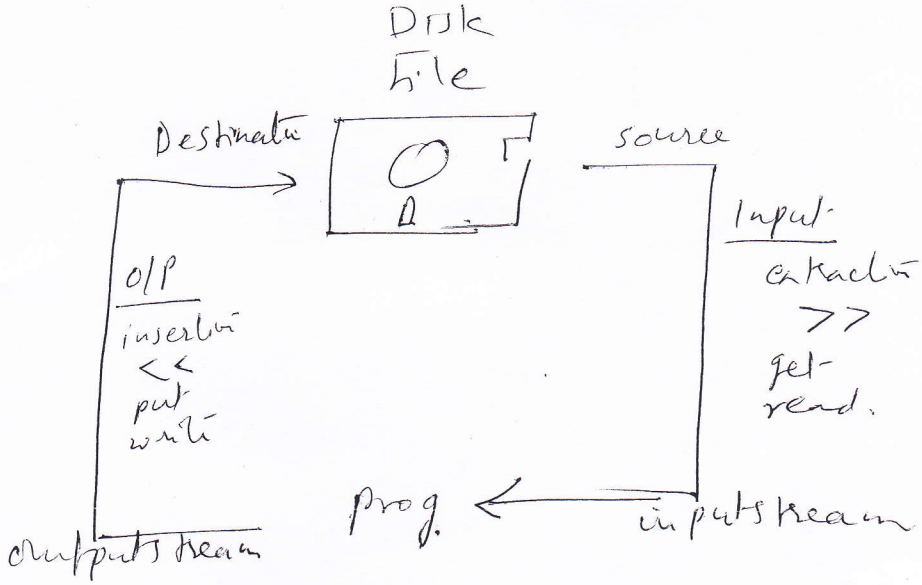
| Static polymorphism | Dynamic |
|---|---|
| → Early binding, Compile time binding | → late, ~~Comp.~~ runtime |
| → Association done at the time of compilation | . |
| → Types used<br>- Operator overloading<br>- function overloading<br>- type casting (overloading) | → virtual function |
| → one example (complete prog) | → one example (complete prog) |

Q 13

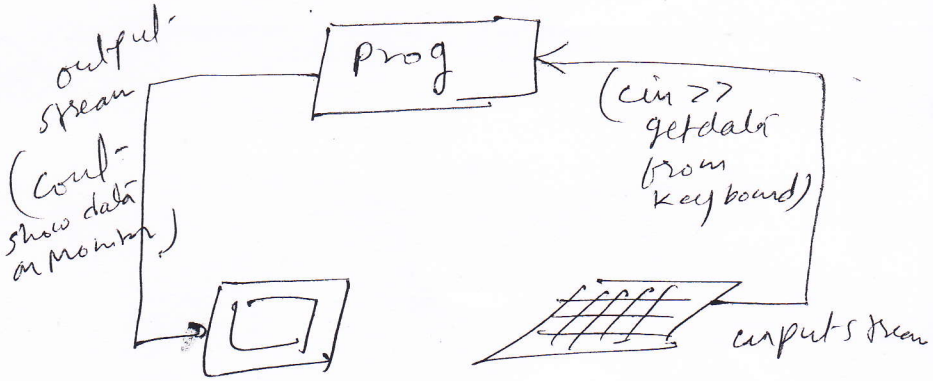
```
#include <iostream.h>
#include <conio.h>
class complex
{
    float real;
    float imag;
  public:
void input()
  { cout<<"\nEnter value for ~~Imaginary~~ real & imag ";
    cin>>real>>imag;

  }
void display()
  { cout<<"\n Imagi Complex values are ";
    cout<<real<<"+i"<<imag<<"i";
  }
void add_complex(complex x, complex y)
  {   temp complex temp;
      temp.real = x.real + y.real
  }       imag = x.image + y.image.

void main()
{  clrscr();
  complex c1, c2, c3;
   c1.input();
   c2.input();
   c3.add_complex(c1, c2);
    c3.display();
  getch();
}
```

Ans 14

## Prog - file interaction




## console keyboard - Prog interaction




## Explaination of

- ifstream
- ofstream
- fstream.
- filebuf
- fstreambase

Ans: 15

## Manipulators for C++

1) endl;
2) hex / oct / dec / showcase
3) setbase
4) setw() ——— cout
5) setfill()
6) setprecision
7) ends
8) ws
9) flush
10) setiosflag()
11) resetiosflags()

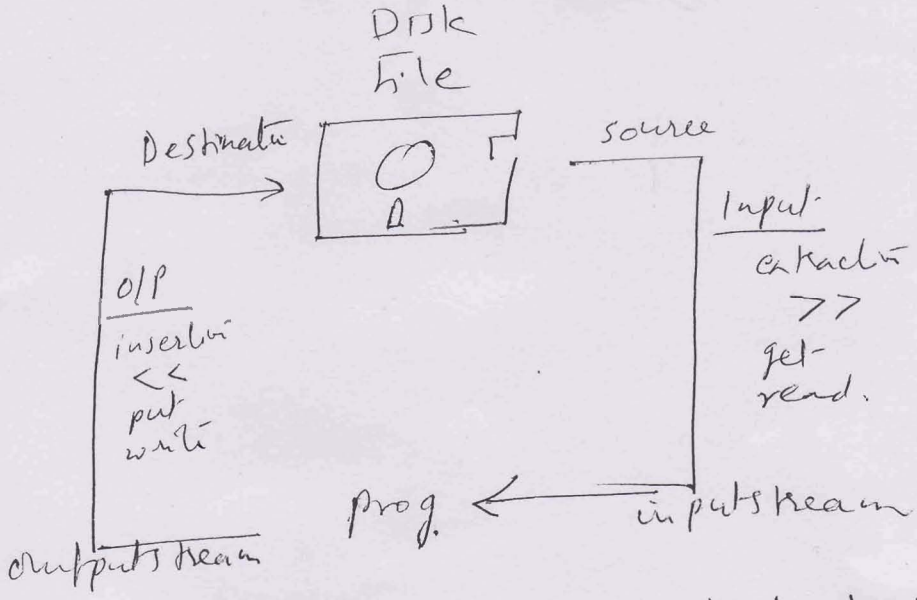
All with an example

Hws - 16

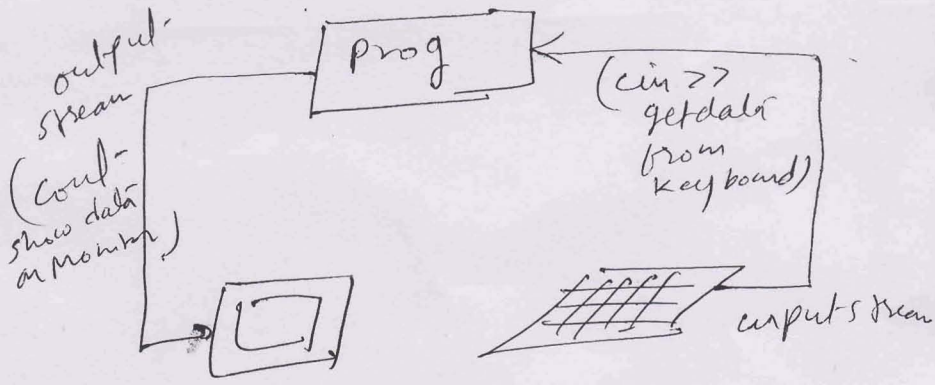
```
#include <iostream.h>
#include <conio.h>

void main()
{   clrscr();
  int count = 0;
  char c ;
  cout<<"\nInput text : ";
   cin.get (c);
while ( c != '\n')
{
      cout .put(c); //optional
      count ++;
      cin.get(c);          ⟶ or → c = cin.get (c);
  }
   cout<<"\n Number of char = "<<count ;
 getch();

}
```

Ans 14

## Prog - file interaction




## console - keyboard - Prog interaction




## Explanation of

- ifstream
- ofstream
- fstream.
- filebuf
- fstreambase

Ans 4 New Header file.

Step 1 : Open Dos editor or Notepad.

Step 2 : write the following code on it

```
void cube ( )
{  int a;
cout << "Enter value to cube ";
cin >> a;
 cout << (a*a*a) ; //or return if return type
                      is given
}
```

Step 3 : save this file with .H extension like cube.H

step 4 : set the path / save file into include folder ~~TC/Include~~ (TC/include).

step 5 : Include this Header file as

#include <cube.h>

step 6 : Use function cube by calling it.

→ A complete example.