Revision
- Compulsary follow all the major pillars of oop

```
int main(){
struct Date d1;

struct Date d2;
}
```

```
struct Date{
int day;
int month;
int year;
}

acceptDate(struct Date d1)
menu-driven code
```

- Size of struct is equal to all the sum of all the size of data members present inside the struct

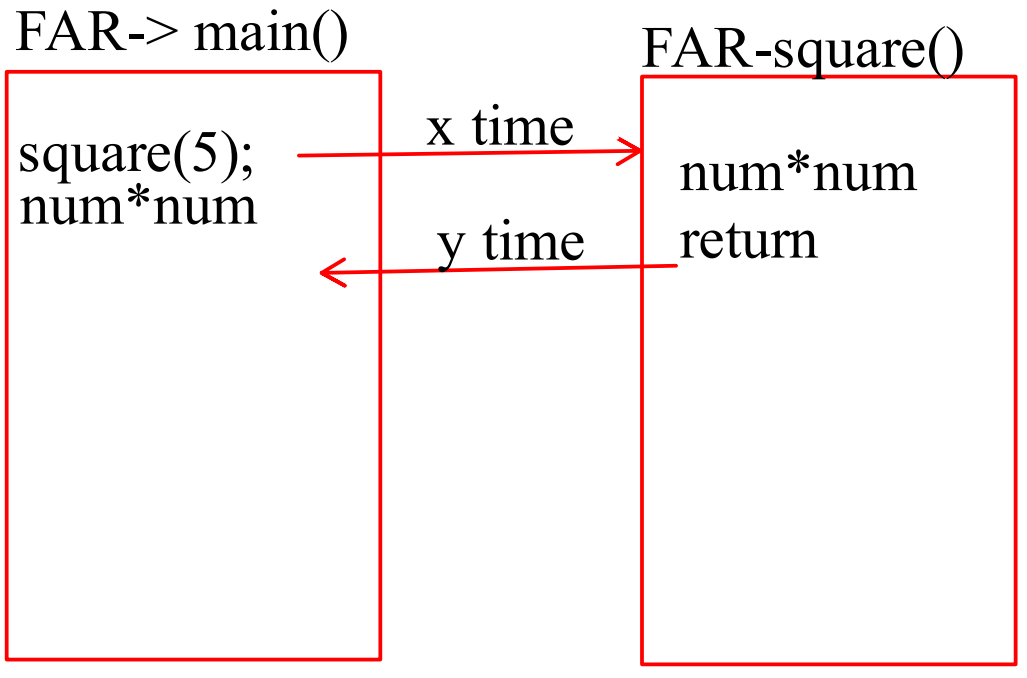
Access Specifiers in struct
public
private

- By default the members of the sructure are public

Inline Functions

```
inline void square(int num){
cout<<"Square = "<<num*num<<endl;
return
}

int main(){
//square(5);
cout<<"Square = "<<5*5<<endl;
return 0;
}
```



(x + y) > z (execution Time)

Class
- It is a logical entity
- It consists of data memebrs (variables declared inside the class) and member functions(functions defined inside the class)
- Members of the class can be static or non static
- It is also called as blueprint of an object

Object
- It is a physical entity
- It is a varaible of a class type
- It is also called as instance of a class
- Object defines 3 things

State
- Data members of the class represents state of an object

Behaviour
- Member functons of the class represents the behaviour of an object

Identity
- Unique data members represent the identity and if they are not present then the address of the object represnts its identity

```
class BankAccount{
int accno
string name;
double balance;

void deposit();
void withdraw();
void displayDetails();
}
```

```
BankAccount a1; // 1001,"Anil",10000
BankAccount a2;//1002, "Mukesh",20000
BankAccount a3;// 1003

a1.deposit()
a1.withdraw()
```

```
class Time{
int hr;
int mins;

};
```

```
Time t1; /// &t1
Time t2;
```

Stack
- Local Variables

Data
- static
- global

Heap
- Dynamic Memory allocation

Text/Code Section

STACK

By default the members of the class are private

Access Specifiers in class

1. private
   - The memebrs are accessiable directly inside the class however they are not accessiable outside the class
2. public
   - The members are accessiable inside the class directly and outside the class on class object
3. protected
   - The behaviour we will study at the time of inheritance

- The memory for the member functions of the class will be given on text/code section only once
- The non static data memebrs will get the memory inside the object.
- the size of object is equal to the size of all the non static data members of the class.
- the size of the object of an empyt class is 1 byte.

- Console I/O

cout is an object of an ostream class
It is declared in the iostream header file as extern
We use insertion operator (<<) with the cout object

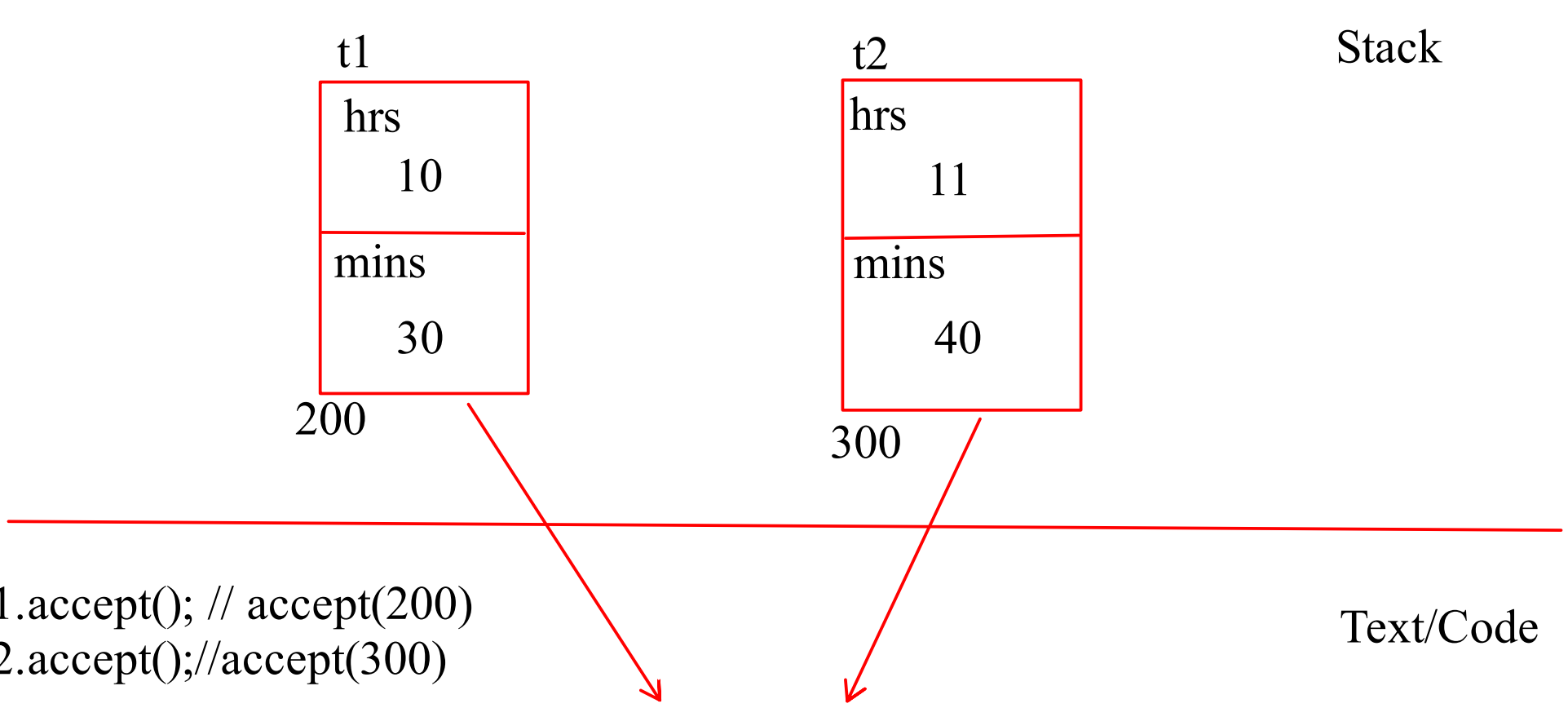
cout<<

cin is an object of istream class
It is declared in the iostream header file as extern
We use extraction operator(>>) with the cin object

-Both the objects are declared inside the namespace called as std.

this pointer

- It is a pointer that is passed to all the non static member functions of the class.
- this pointer points at the current calling object i.e it stores the address of the current calling object
- It is a pointer that is passed internally
- It is a constant pointer
- Using this pointer is optional



namespace
It is a container used to oraginize the code.

```
namespace utility{
class Date{
int day;
int month;
int year;
}
}
```

```
namespace sql{
class Date{
string date;
int day;
int month;
int year;
}
}
```

```
namespace NAttendance{
    class Employee{
    }
    class Student{
    }
    class Date{
    }
    class Attendance{
    }
}
```

```
int NA_num1 = 10;
int NB_num1 = 20;

namespace NA{
int num1 = 10;
}

namespace NB{
int num1 = 20;
}
```

NA::

Chips

Healthy NonHealthy

Function Overloading
- Defininng multiple functions with same name but different signature
- The signature can differ in
    1. No of parameters
    2. If no of parameters are same then the types can be different
    3. If no and types are same then theor order should be different
- It is an example of compile time polymorphism

```
void multiply(int n1, int n2){
cout<<n1*n2;
} // multiply_i_i
```

```
multiply(10,2);
multiply(10,2,3);
```

```
void muliply(int n1, int n2,int n4){
cout<<n1*n2*n3;
}//multiply_i_i_i
```

```
void square(int n){
cout<<"square = "<<n*n;
}//square_i
```

```
void square(double n){
cout<<"square = "<<n*n;
}//square_d
```

```
square(5);
square(5.5);
```

```
void division(double n1, int n2){
cout<<n1/n2;
}//division_d_i
```

```
void division(int n1, double n2){
cout<<n1/n2;
}// division_i_d
```

```
division(10.6,2);
division(10,2.5);
```

Managaled Name

Name Mangling

- In case of function overloading the return type of the function does not matter.
- Function overloading is not considered on the return type of function
- It is possible due to the name mangling the compiler performs.
- The name that are assigned by the compiler to the overloaded functions is called as mangaled name

Default Argument Function
- These are the functions that have default values to the paarameters

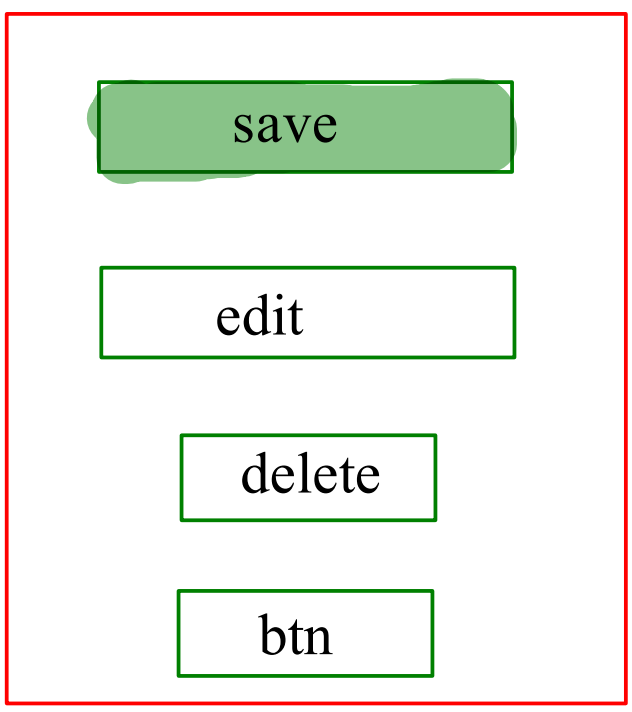
```
void add(int n1, int n2, int n3=0, int n4=0){
        cout<<n1+n2+n3+n4;
}
```

```
add(10,20,30,40);
add(10,20,30);
add(10,20)
```

```
void mul(int n1, int n2, int n3=1, int n4=1){
        cout<<n1*n2*n3*n4;
}
```

```
mul(2,3);
mul(2,3,4);
mul(2,3,4,5);
```

- A function that have defualt values assigned to its paramteres is called as default argument function
- the defualt values for the parameters should be given from the rightmost parameter




```
createButton(string name="btn",int px=10, string  color="white"){
}

createButton("save",20,"green");
createButton("edit",20);
createButton("delete");
createButton();
createButton("back","red"); // NOT OK need to perform overloadi
```

Types of member Functions
1. Constructor
2. Destructor
3. Mutator
4. Inspector
5. Facilitator