git clone https://github.com/sunbeam-dac-aug24/cpp

Revision
OOSD

Datatypes

```
// C-Programmaing
struct Time{
// Members
int hrs; // variable
int mins;
}

void accept(struct Time *t){
pf("Enter hrs and mins");
sf(%d,%d,&t->hrs, &t->mins);
}

void display(struct Time *t){
pf(%d%d,t->hrs,t->mins);
}
```

```
int main(){
// variable of a struct
struct Time st; //object
struct Time et;

st.hrs;

accept(&st);
display(&st)
}
```

```
// CPP-Programming
struct Time{
// Members
int hrs;
int mins;

void accept(){
pf("Enter hrs and mins");
sf(%d,%d,&hrs, &mins);
}

void display(){
pf(%d%d,hrs,mins);
}
}
```

```
int main(){
// variable of a struct
struct Time st; //object
st.accept();
st.display();
}
```

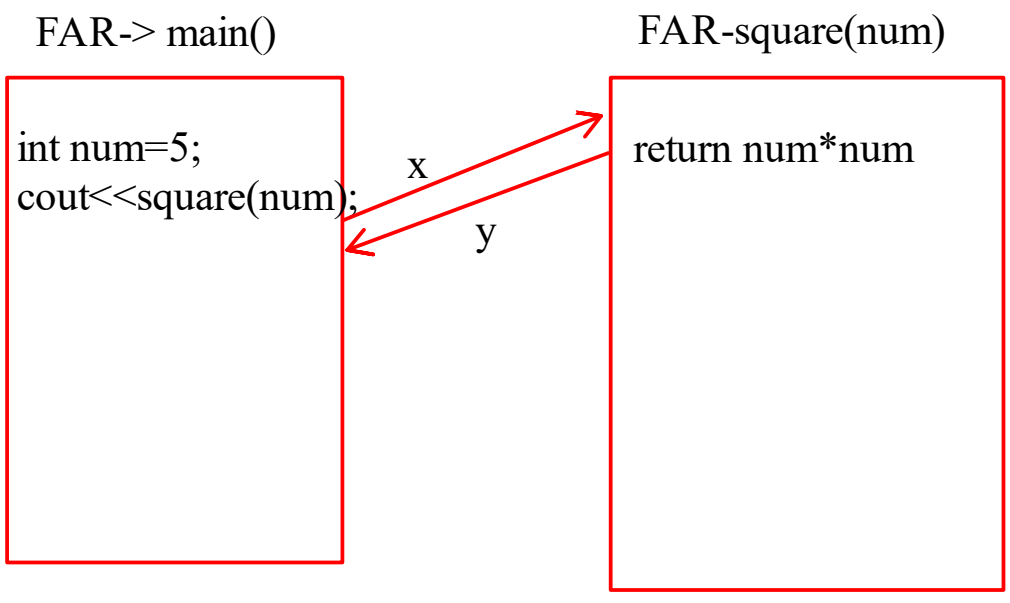
Inline Function

By default members of the structure are public

```
int main(){
int num=5;
cout<<square(num);
}
```

```
inline int square(int num){
return num*num;
}
```

```
int main(){
int num=5;
cout<<num*num; // at compile time
}
```



t amount of time for execution

x+y > t

inline

- inline is just a request made to the compiler

Class is a logical entity
Class is also called as blueprint of an object

```
class Time{
int hrs;
int mins;

void accept(){
}

void display(){

}
}
```

Object
It is Physical entity
It is also called as instance of a class

Access Specifiers
1. private
   - accessiable only within the class
2. public
   - accessiable within the class directly
   - accessiable outside the class on class object
3. protected
   - We will study at the time of inheritance

class consists of
1. static
2. non static

- Size of object is equal to size of all the non static data memembers of the class

```
Time t1; // hrs,mins
Time t2; // hrs,mins
Time t3;//hrs,mins
```

Object
- It defines 3 things
1. State
   - Data members of the class represents state of an object
2. Behaviour
   - Member functions of the class represents behaviour of an object
3. Identity
   - The unique data member of the class represents identity. If unique data member is not present then the address can be used as the identity

Stack
- All the local varaibles will get the memory on this section

Heap
- All the dynamically allocated memory will be done on this section

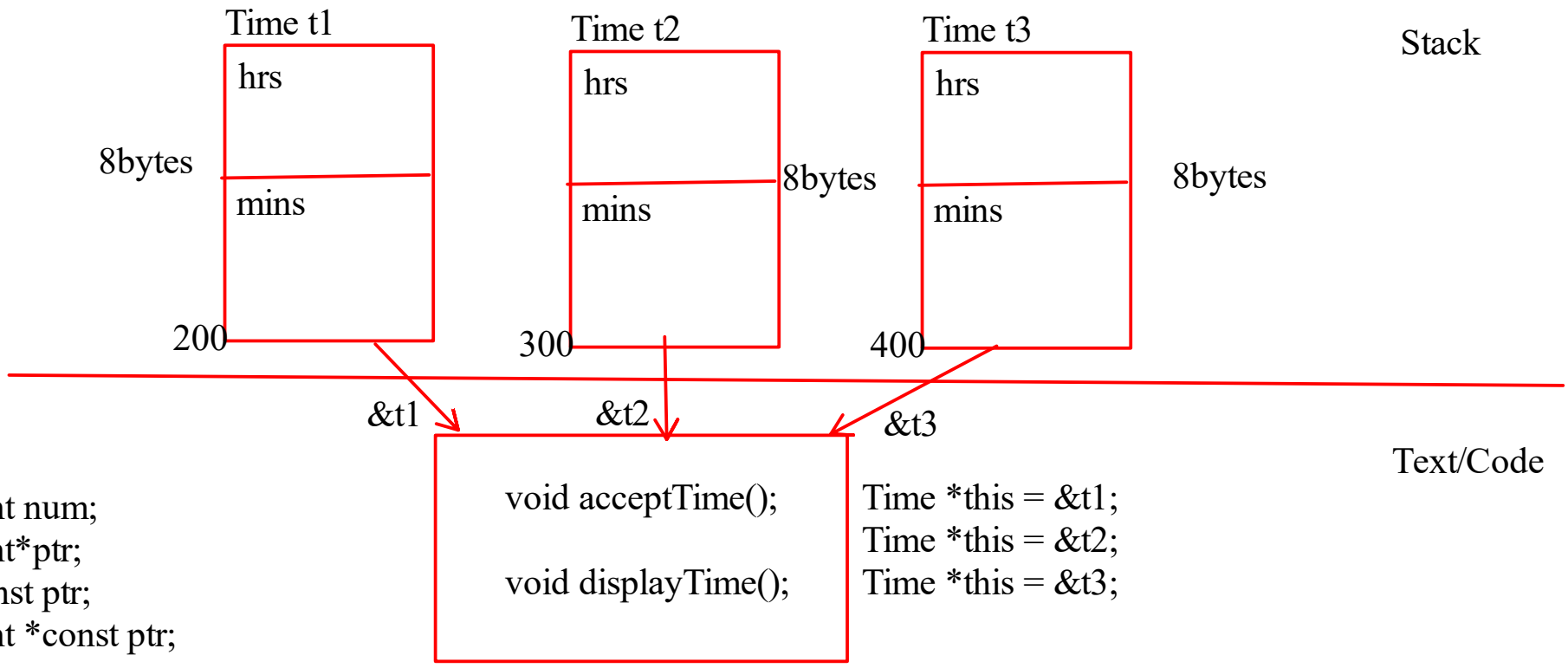
Data
- Global and static variables will get the memory on this section

stack
heap
data
text/code

```
class Employee{
int empid;

}
```

RAM
Memory
Sections

ostream -> cout
istream -> cin
Both these objects are declared as extern inside the iostream header file
cout -> we use an insertion operator (<<)
cin -> we use an extraction operator (>>)



const int num;
const int*ptr;
int *const ptr;
const int *const ptr;

this ptr will be only available inside non static member functions of the class
we can access only the datamemebrs of the class using this pointer.

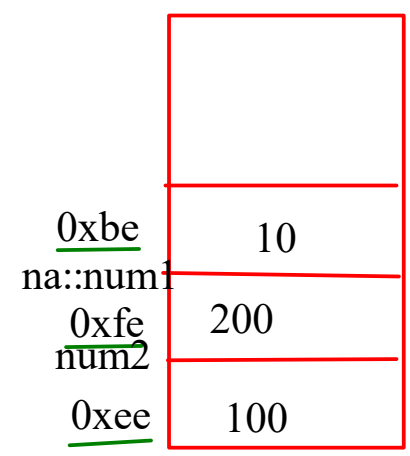
this->hrs;
this->min;

```
namespace utility{

}

namespace std{

}

namespace sql{

}
```

```
namespace util{
class Date
{
day,
month
year
}
}
```

```
namespace AttendanceSystem{
class Employee{

}

class Student{


}

class Attendance{

}
}
```

```
namespace sql{
class Date{

}
}
```



Namespace
- It is a container used to categorize the code.
- It cannot be created locally
- It cannot be instantiated
- to access members of the namespace use name of the namespace and scope resolution operator (::)
- to access members of the namespace directly use using directive

6 months -> Package
J2EE
.net
React

Function Overloading

```
//square_i -> Managaled name
void square(int num){
cout<<num*num<<endl;
}

//square_d
void square(double num){
cout<<num*num<<endl;
}
```

```
square(10);
square(11.22);

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

div(10,2.5);
div(10.5,2);
```

```
//div_i_d
void div(int numerator, double denominator){
cout<<numerator/denominator<<endl;
}

//div_d_i
void div(double numerator, int denominator){
cout<<numerator/denominator<<endl;
}
```

```
//add_i_i
void add(int num1, int num2){
cout<<num1+num2<<endl;
}

//add_i_i_i
void add(int num1, int num2,int num3){
cout<<num1+num2+num3<<endl;
}

//add_i_i_i_i
void add(int num1, int num2,int num3, int num4){
cout<<num1+num2+num3+num4<<endl;
}
```

Name Mangling

Defining the function multiple times with same name but different signature is called as function overloading
Different signature deatermines -
1. change in no of parameters
2. if no of parameters is same thet change their types
3. If no and types are same then their order should be changed.

It is an example of compile time polymorphism

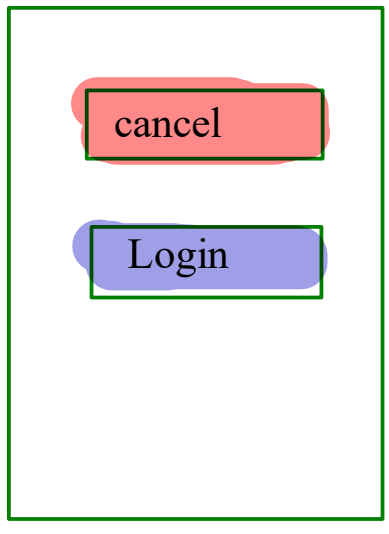
Default Argument Function

```
void add(int num1, int num2,int num3=0, int num4=0){
cout<<num1+num2+num3+num4<<endl;
}
```

```
add(10,20,30,40);
add(10,20,30);
add(30,40,50);
add(10,20);
```

```
void createButton(string name="btn",string color="grey"){

}
```

```
createButton("login","blue");
createButton("delete","red");
createButton("save");
createButton();
```



Types of Member Function
1. Constructor -> Used to initialize the object
2. Destructor -> used to deallocate the memory/release the resources consumed
3. Mutator -> It is used to provide the write permission(change value) for a single data member
4. Inspector -> It is used to provide the read permission (fetch the value) for a single data member
5. Facilitators -> It provides the facility to perform the operations

```
class BankAccount{
int accno; // getAccno();
string name; // getName();
double balance; // setBalance(), getbalance()
}
```