```
struct time{

};

void accept(){

}

int main(){

int num1;
int num2;
int num3;

num1;
time t1;

accept();


}
```

Document

num1-

num2-

time -

accept -

acceptData()

const double PI=3.14

class

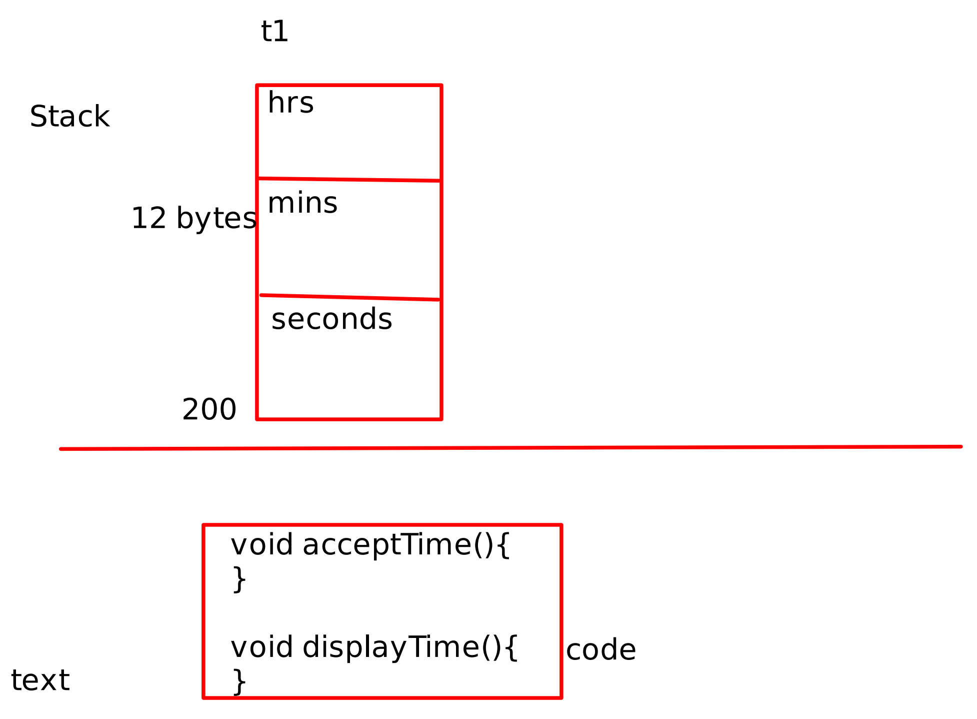
It is logical entity
It consists of
Variables
// data members

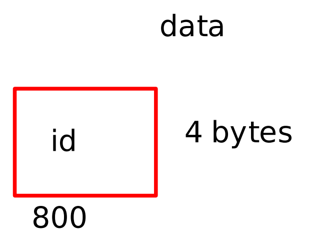
functions
// member functions

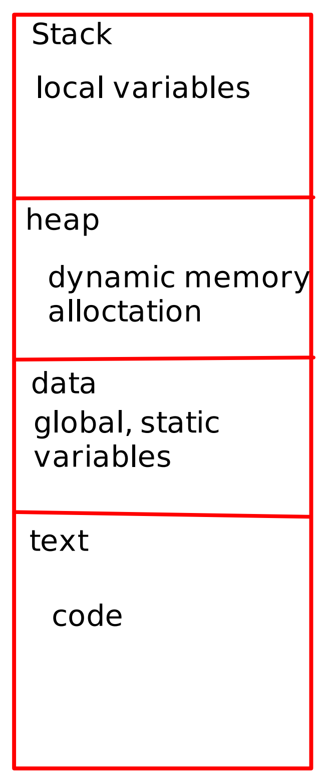
```
struct Time{

// data members
int hrs;
int mins;

// member functions
void acceptTime(){

}

void displayTime(){

}

};

main(){

struct Time t1;
}
```

```
class Time{

// data members
int hrs;
int mins;

// member functions
void acceptTime(){

}

void displayTime(){

}

};

main(){

Time t1; // Object
}
```

size of object is sum of all the non static Data Members of the class
Member functions do not get space inside object







namespace is a container which contains
variables
function
structure
class

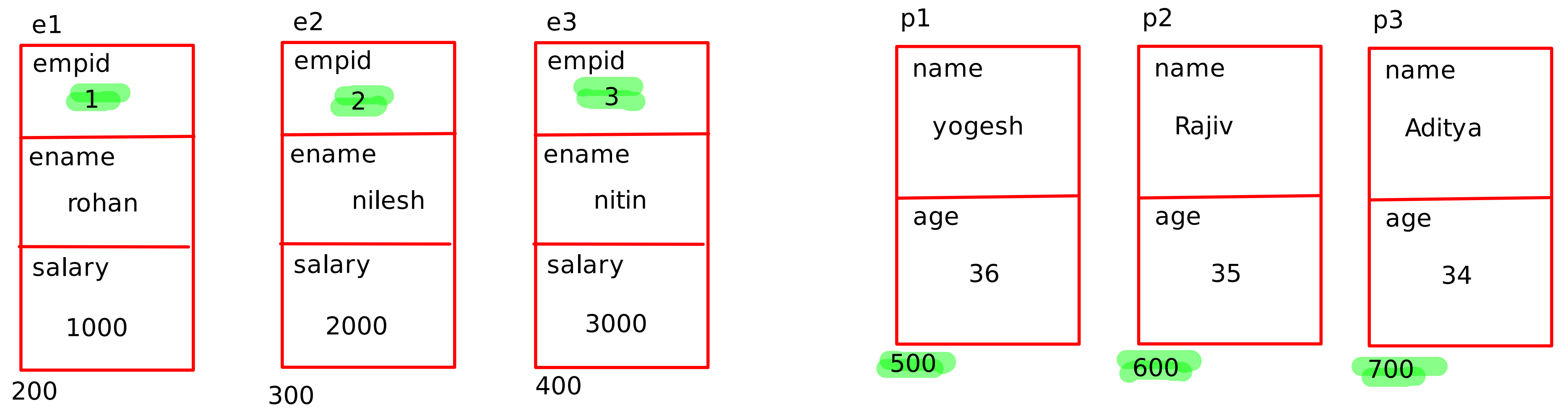
we cannot create object of a namespace
we cannot define namespace under local scope

cout -> ostream
cin-> istream

cin,cout as external objects inside iostream
these are declared inside the namespace called as std

```
cout << "Enter value of num1 - ";
cin>> num1;
```

Object defines 3 things-
1. state -> Data members defined inside the class
2. behaviour -> Member functions represent behaviour of an object
3. Identity -> unique datamembers inside class will represent identity. If unique data members are not present then addess will be used for the identity



Access Specifiers in class

1. private -> Members are accessiable/visible only within the class

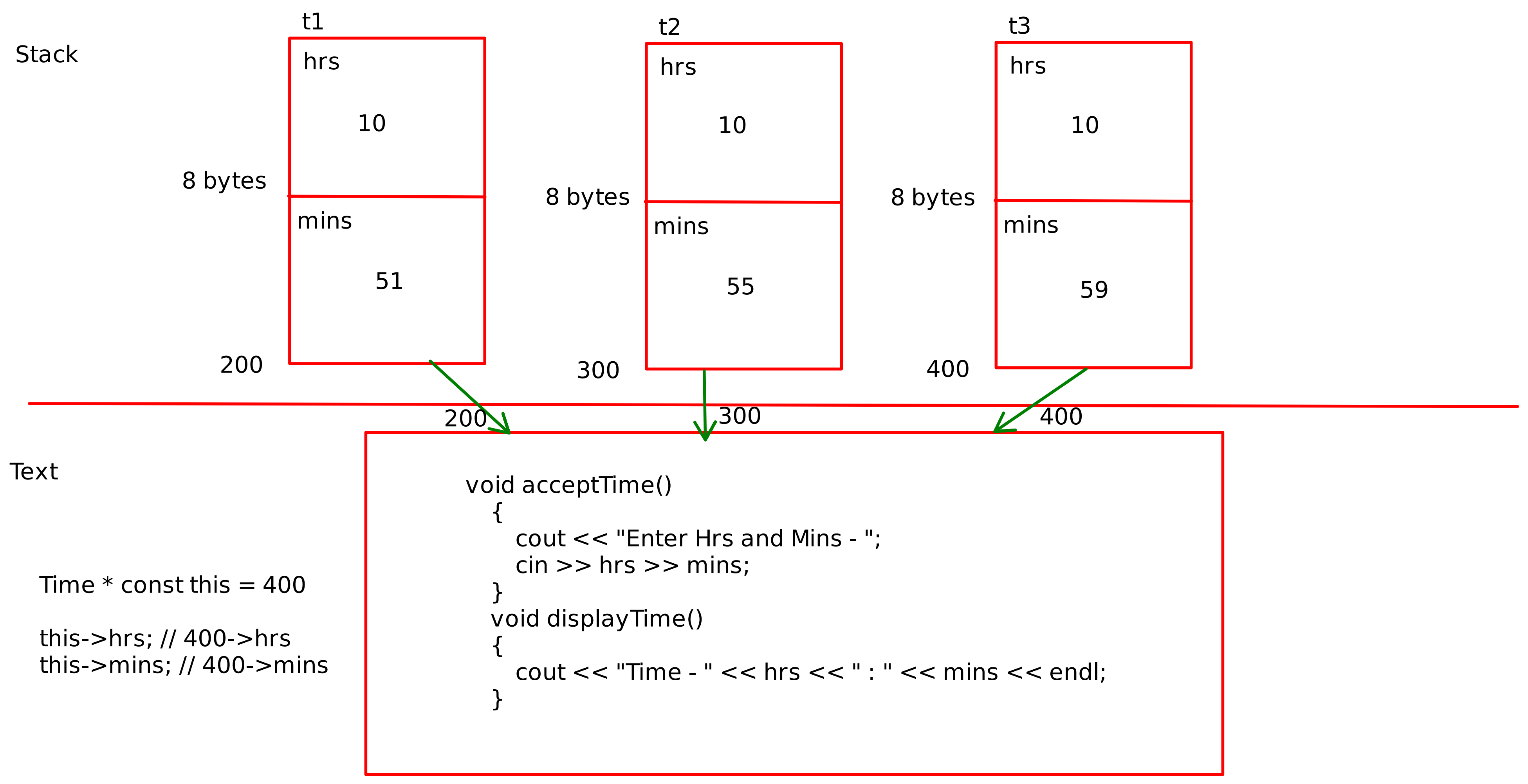
2. protected-> members are accessiable with the class and the derived class directly however they are not accessiable/visible outside the class

3. public -> Members are accessiable/visible within the class directly and even outside the class on class object

```
Time t;
t.hrs;
t.mins
```

Base -> Protected

Dervied



Time * const this = 400

this->hrs; // 400->hrs
this->mins; // 400->mins

```
void acceptTime()
   {
      cout << "Enter Hrs and Mins - ";
      cin >> hrs >> mins;
   }
   void displayTime()
   {
      cout << "Time - " << hrs << " : " << mins << endl;
   }
```

this pointer is present in all the non static member functions of the class.
this pointer is a const pointer which is passed internally.
It points to the current calling object

1. Add
2. Sub
3. Mul

```
void add(int n1,int n2){
cout<<"Addition ="<<n1+n2;
}

void addDouble(int n1,int n2){
cout<<"Addition ="<<n1+n2;
}


add(10,20);
addDouble(10.20,20.25);
```

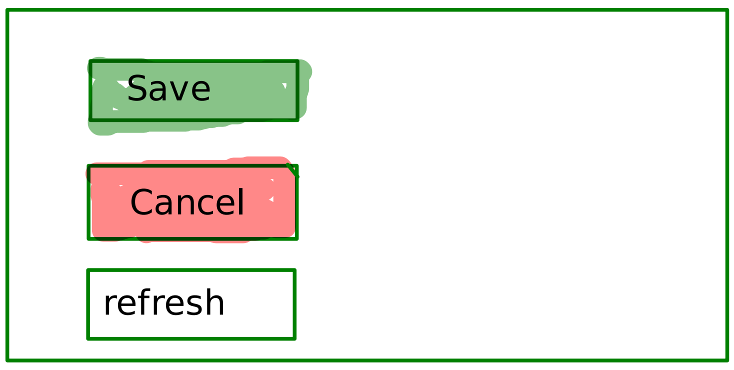
Polymorphism

function Overloading

definining multiple functions with same name
but their signature should be different

1. No of parameters should be different
2. No of para are same then type should be different
3. If no and type of para are same then their order should be different

It is beacuse of the concept called as name mangling



```
void cretaeButton(string name,string color){
// Logic to cretae the button oon the UI
}

void cretaeButton(string name,string color,int cornerradius){
// Logic to cretae the button oon the UI
}

void cretaeButton(string name){
// Logic to cretae the button oon the UI
}
```

```
int main(){

cretaeButton("save","color");

cretaeButton("cancel","red",5);

createButton("refresh");

return  0;
}
```

Default Argument Function

```
void cretaeButton(string name,string color="white",int cornerradius=1){
// Logic to cretae the button oon the UI
}
```

The function which have default values assigned to its parameters
is called as Default Argument Function

# Types Of Member Functions
1. Construtor
2. Destructor
3. Mutator -> Setter
4. Inspector -> Getter
5. Facilitator

```
class Demo{

int *ptr = malloc();

void f1(){
}
void f2(){
}
void f3(){
}
void f4(){
}
void f5(){
}
void f6(){
}
void f7(){
}

}
```

```
class Employee{
public :
id,
name,
sal
}

main(){
Employee e;
e.id=10;
cout<<e.name;
e.name= "rohan";//error
read as well as write/manipulate
}
```