C++ class

C++ class

- A class is a user defined data type.
- It holds its own data members and member functions.
- Syntax for defining a class:

```
class ClassName{
        <access specifier>
        // Can be public, private or protected.
        // Default access specifier is private.

        <Data members>
        // int a, b, c;
        // float f;

        <Member functions>
        // Constructors
        // Destructor
        // Other functions

};
```

Demo: Porting a Java class for Complex numbers to C++ class

```
int i = 0;
int j = 5;
char c = 'A';
```

Built-in data variables can be initialized when they are declared.

Can we initialize user-defined objects at declaration?

C++ provides ‘constructors’ to make this possible

Constructors in C++

- *Special* member functions → same name as the class
- Constructors initialize objects → 'constructs' data for objects
- Constructors have no return type

```
class Complex{
     double re;
     double im;

     public:
     // Default constructor
     Complex();
     // Constructor with inputs
     Complex(double real, double imag);
     ...
};
```

Constructors in C++

```
// Definition of default constructor
Complex::Complex(){
     real = 0; img = 0;
}
// Definition of constructor with inputs
Complex::Complex(double real, double imag){
     re = real; im = imag;
}
int main()
{
     Complex c1, c2(1.0,2.5);
     // [some code here]
     return 0;
}
```

- `c1` is initialized by the default constructor `c1 = 0.0 + i 0.0`
- `c2` is initialized by the other constructor `c2 = 1.0 + i 2.5`

Copy constructor

- Syntax

```
Complex(const Complex& c);
```

- Definition is as follows

```
Complex::Complex(const Complex& c){
        re = c.re; im = c.im;
}
```

- Copy constructor receives ***reference*** to its ***own*** class as a parameter
- Cannot receive ***value*** instead of ***reference***

```
complex(complex c);
```

Note: you can use this in Java

Constructors in action

C++ automatically invokes the right overloaded constructor depending on the signature.

```
class Complex {
          ...
          public:
          Complex(); // Default
          Complex(double real, double imag);
          Complex(Complex& c); // Copy constructor
          ...
};


int  main(){
          Complex c0;
          Complex c1(1.0, 2.5);
          Complex c2(c1);
          ...
```

Pointer to class object

- Pointer to a class-object can be declared as

```
            ClassName *p;
Example:    Complex *p;
```

- Members are accessed using '->' operator.

Example:

```
Complex a(5.0, 6.0);
Complex *p;
p = &a;
cout << p->toString() << endl;
```

Memory allocation using **new**

- Syntax for memory allocation using **new**:

```
T *p;          // p points to data-type T
p = new T;     // One object of type T is allocated in Heap
               // and p points to the allocated object.
```

- We can also initialize the memory using **new** operator:

```
int *p = new int(7);            // int object is initialized to 7
float *q = new float(5.25);     // float object is initialized to 5.25
```

- To allocate a block (an array) of memory, the syntax is:

```
p = new T[SIZE];
Example:
int *p = new int[10];   // allocates memory for array of 10 integers
                        // in Heap and p points to the start of array.
```

- If **new** fails to allocate memory, then it throws an exception.

Memory deallocation using **`delete`**

- Programmer is responsible for freeing allocated memory.
- Syntax for deallocating memory using **delete** :

```
int *p = new int(5);     // single object is allocated
int *q = new int[10];   // array of objects are allocated
...
delete p;          // single object is deleted
delete [ ]q;     // array of objects are deleted
```

Be careful of memory leaks when you dynamically allocate memory.
Use the Valgrind tool to check for memory leaks.

Destructors in C++

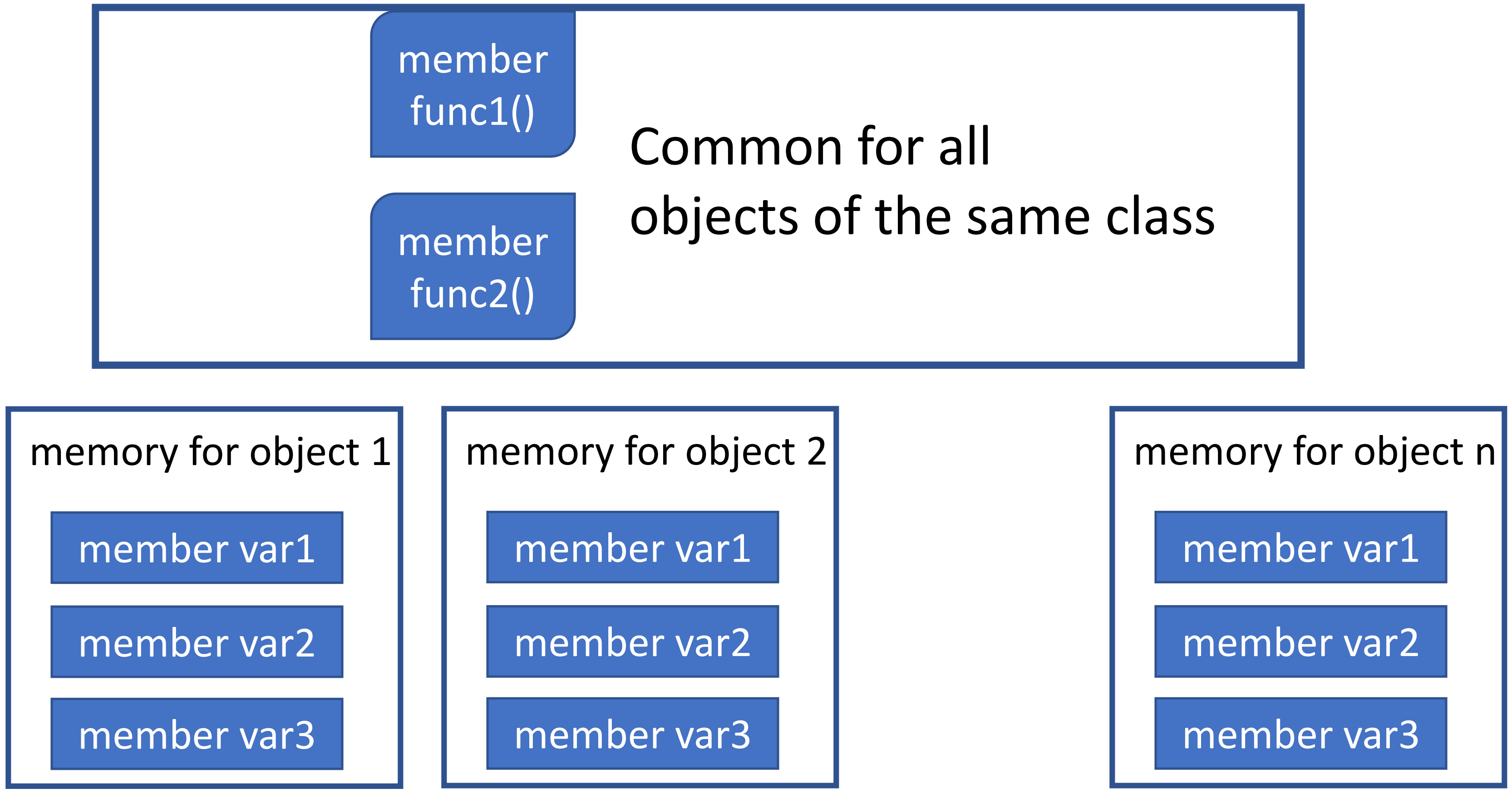
- Destructor is a special member function which ‘destroys’ or deletes objects of the class.
- Destructor does not take any argument nor returns.
- Destructor has the same name as the class with a ~ ahead. Example: destructor for the ‘Complex’ class is ~Complex().

```
class Complex {
   ...
   ~Complex();
};
```

- Destructor is automatically called when
  - an automatic object goes out of scope (example: the function finishes or the program ends.
  - **delete** operator is called on dynamically created objects using **new**.

Memory layout of C++ class objects

- Only one copy of member functions is kept in the text segment of memory.
- Member variables (as they are the ‘data’) are stored separately.


member func1()
member func2()
Common for all objects of the same class
memory for object 1
member var1
member var2
member var3
memory for object 2
member var1
member var2
member var3
memory for object n
member var1
member var2
member var3