Constructors, Destructors and Inline Functions

Class and Object

- **Class:**
  - Common template containing data and functions
  - Logical entity - no memory set aside

- **Object/Instance:**
  - Real world entity and physical entity (memory set aside)
  - Examples: Student, Employee, etc.
  - Identity- Every object is unique
  - State: Current values of its fields, e.g., Student id, Name, Phone No., Address, etc.
  - Behavior - methods/procedures that update the state

Example with Class

```
#include <iostream>
using namespace std;
# define SIZE 100
// This creates the class
stack.
class stack {
int stck[SIZE];
int tos;
public:
void init();
void push(int i);
int pop();
};
```

Member functions

```
void stack::init() {
tos = 0; }

void stack::push(int i) {
if(tos==SIZE) {
cout << "Stack is full.\n";
return;
}
stck[tos] = i;
tos++;
}
```

scope resolution operator :: indicates that the member function belongs to a class.

In C++, all functions must be prototyped

```
class class-name {
private data and
functions
public:
public data and
functions
} object name list;
```

Example Cont..

```
int stack::pop() {
if(tos==0) {
cout << "Stack underflow.\n";
return 0;
}
tos--;
return stck[tos]; }
int main()
{
// create two stack objects
stack stack1, stack2;
stack1.init();
stack2.init();
```

```
stack1.push(1);
stack2.push(2);
stack1.push(3);
stack2.push(4);
stack1.tos = 0; // Error, tos is private.
cout << stack1.pop() << " ";
cout << stack1.pop() << " ";
cout << stack2.pop() << " ";
cout << stack2.pop() << "\n";
return 0;
}
```

The output : 3 1 4 2

Two separate objects

dot operator for accessing data member or member function

Struct vs Class in C++

```
class X {
  // private by default

int a;

public:

  // public member function
  int f() { return a = 5; };

};
```

```
struct Y {
  // public by default

  int f() { return a = 5; };

private:

  // private data member
  int a;
};
```

Struct vs Class in C++ Example

```
#include <iostream>
using namespace std;
struct X {
 int a;
 int b;
};
class X obj_X;
//struct X obj_X;
//We can use any one
//Functions can also be there
int main() {
  obj_X.a = 0;
  obj_X.b = 1;
  cout << "Here are a and b: "
  << obj_X.a << " "
  << obj_X.b << endl;
}
```

Data Member details

- No member can be an object of the class that is being declared.
- A member can be a pointer to the class that is being declared.
- No member can be declared as auto, extern, or register.

Constructors

- Set function is not the proper way of initializing an object, user may forget to initilize values.
- C++ allows objects to initialize themselves when they are created.
- Automatic initialization is performed through the use of a constructor function.
- A constructor is a special function that is a member of a class and has the same name as that class.
- They do not have any return values, i.e., no return type (not even void).
- The constrcutor of that class is called automatically when an object of a class is declared in the main function.

Default Constructors

- When a class has a constructor, all objects of that class will be initialized (delcaration is an action statement in C++).
- The constrcutor without arguments is called as a default constructor.
- If no user-defined constructor exists for a class, then the compiler implicitly declares a default constructor (without a body) and if the user hasn’t declared other constructors.

Example

```
#include <iostream>
 using namespace std;
 class sample {
private:
      double  X;

public:
  void setData( double);
      double
getData( void );
      sample();
//Constructor
   };
```

```
 // Member functions
definitions
sample::sample(void) {
cout << "Object is being
created" << endl;
}
void sample::setData (double
Y)
{
   X= Y;
}
double
sample::getData( void ) {
   return X;
}
```

```
// Main function
int main() {
   sample s; ← Constructor
                called.
  // set data
  s.setData(10.0);
 cout << "Data is : "
<< s.getData()
<<endl;
 return 0;
}
```

Output:

Object is being created

Data is: 10.0

Constructors

- Global objects:
  - An object's constructor is called only once the program starts the execution
- Local objects:
  - An object's constructor is called each time the declaration statement is executed.
- It is not possible to take the address of a constructor.
- Can the constructor perform any type of operation?
- Can the constructor be declared in a private section?

Example- Inline Functions

- Calling a function:
  - requires storing the arguments, local variables on to the stack
  - pushing the registers onto the stack
  - restoring the registers
  - execution-time overhead

- Short functions in C++ are not called, their code is expanded in line at the point of each invocation -called inline function
- Efficient and makes function run faster.
- Puts copy of function's code in place of a function call
- Speeds up performance but increases file size
- Which functions to be made Inline?
- Request to the compiler

Example- Inline Functions

```
#include <iostream>
using namespace std;
inline int max(int a, int b)
{
return a>b ? a : b;
}
int main() {
cout << max(10, 20);
cout << " " << max(99, 88);
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
cout << (10>20 ? 10 : 20);
cout << " " << (99>88 ? 99 : 88);
return 0;
}
```

Example- Inline Functions

```
#include <iostream>
using namespace std;
class myclass {
int a, b;
public:
void init(int i, int j);
void show()
{
cout << a << " " << b <<
"\n"; }
};
```

```
// Create an inline
function.
inline void
myclass::init(int i, int j)
{
a = i;
b = j;
}
```

```
int main()
{
   myclass x;
   x.init(10, 20);
   x.show();
   return 0;
}
```

Parameterized Constructors

- Constructors with parameters are called as parameterized constructors.
- The signature of the constructors must be provided as a part of class.
- In this case, an appropriate number of arguments must be passed by user in order to call the constructor.

Parameterized Constructor Example

```
#include<iostream>
using namespace std;

class sample {
private:
      double  X, Y;

  public:
//Parameterized Constructor
sample(double, double);
double display( );
};
```

```
sample::sample(double A,
double B) {
cout << "Object is being
created”<< endl;
X = A;
Y = B;
}
double  sample :: display()
{
cout << "X = " << X << "Y = " <<
Y << endl;
}
```

```
// Main function
int main() {
sample s(10.4, 20.5);
s.display();
return 0;
}
```

Output:

Object is being created

X = 10.4

Y = 20.5

Parameterized Constructor Example

```
#include <iostream>
#include <cstring>
using namespace std;
const int IN = 1;
const int CHECKED_OUT = 0;
class book {
char author[40];
char title[40];
int status;
public:
book(char *n, char *t, int s);
int get_status() {return status;}
void set_status(int s) {status = s;}
void show();
};
```

```
book::book(char *n, char *t, int s)
{
strcpy(author, n);

strcpy(title, t);
status = s;
}

void book::show()
{
cout << title << " by " << author;
cout << " is ";
if(status==IN) cout << "in.\n";
else cout << "out.\n";
}
```

```
int main()
{
book b1("Twain", "Tom Sawyer",
IN);
book b2("Melville", "Moby Dick",
CHECKED_OUT);
b1.show();
b2.show();
return 0;
}

//Output
//Tom Sawyer by Twain is in.
//Moby Dick by Melville is out.
```

Use of Constrcutors

- An elegant approach to initilize an object of a class.
- The manipulation of values can not be done by users
- Users need not initialize values

Destructors

- The destructor function is called when an object is destroyed.
- It is common to have to perform some actions when an object is destroyed.
- For example, an object that allocates memory when it is created will want to free that memory when it is destroyed.
- The name of destructor is the name of its class preceded by a ~.

Destructors Example

```
# include <iostream>
using namespace std;
class myclass {
int a;
public :
myclass (); // constructor
~ myclass (); // destructor
void show ();
};

myclass :: myclass () {
cout << "In constructor \n";
a = 10;}

myclass ::~ myclass () {
cout << " Destructing ...\ n"; }

void myclass :: show ()
{
cout << a << "\n";
}
int main () {
    myclass ob;
    ob. show ();
    return 0;
    }
```

Destructors Cont..

- A class's destructor is called when an object is destroyed.
- Local objects are destroyed when they go out of scope.
- Global objects are destroyed when the program ends.
- It is not possible to take the address of a destructor.

References

- C++: The Complete Reference, 4th Edition by Herbert Schildt , McGraw-Hill
- Teach Yourself C++ 3rd Edition by Herbert Schildt,
- The C+ + Programming Language, Third Edition by Bjarne Stroustrup, Addison Wesley