Constructing and Destructing

- Version 1: Dr. Ofir Pele
- Version 2: Dr. Miri Ben-Nissan
- Version 3: Dr. Erel Segal-Halevi

	C	C++	Java
Objects on the stack:			
Construct	No	Yes	*Impossible*
Destruct	No	Yes	
Objects on the heap:			
Construct	No (`malloc`)	Yes (`new`)	Yes (`new`)
Destruct	No (`free`)	Yes (`delete`)	No (done automatically by garbage collector)

C++ Laws of Construction and Destruction

1. Every object must be **constructed** before it is used.

- Stack object: when it is defined.
- Heap object: when it is **new**-ed.

2. Every object must be **destructed** after it stops being of use.

- Stack object: when gets out of scope.
- Heap object: when it is **delete**d.

Constructors *(folder 4)*

```
class MyClass
{
public:
1   MyClass();
2   MyClass( int i );
3   MyClass( double x, double y );
    ...
};

int main() {
    MyClass a; // Calls 1
    MyClass b {5}; // Calls 2
    MyClass c {1.0, 0.0}; // Calls 3
}
```

Constructors and Arrays *(folder 4)*

```
class MyClass
{
public:
1   MyClass();
2   MyClass( int i );
3   MyClass( double x, double y );
    ...
};

int main() {
    MyClass a[5]; // Calls 1 five times
    MyClass b[5] {11, 22}; // Calls 2 two times
    MyClass c[5] { {11,22}, 33}; // Calls 3 then 2
}
```

Constructors – parameterless ctor

```
class MyClass {
public:
   MyClass(); // parameterless ctor.
   //...
};
//...
int main() {
   MyClass a; // parameterless ctor called
   // …
}
```

Constructors – default parameterless ctor

```
class MyClass {
public:
   // No ctors
};

int main() {
  MyClass a; // default parameterless ctor:
}
```

Constructors – no default parameterless ctor

```
class MyClass {
public:
   MyClass(int x); // no parameterless ctor.

};

int main() {
   MyClass a;      // compiler error –
   MyClass b[5];   // no parameterless ctor.
}
```

Constructors – explicit default parameterless ctor

```
class MyClass {
public:
   MyClass(int x);
   MyClass() = default;
};

int main() {
   MyClass a; // default parameterless ctor
}
```

Constructors – deleted default parameterless ctor

```
class MyClass {
public:
   MyClass() = delete;
};

int main() {
   MyClass a; // compiler error –
              // no parameterless ctor.
        // (why would someone do this??)
}
```

Destructors

Goal: Ensure proper “cleanup”:

- Free allocated memory;
- Close opened files or db connections;
- Notify related objects, etc.

Use: Called for:

- A stack object – when it goes out of scope.
- A heap object – when it is explicitly deleted.

Destructors *(folder 5)*

```
#include <cstdlib>
class MyClass
{
public:
   MyClass();  // constructor
   ~MyClass(); // destructor
private:
   char* _mem;
};
MyClass::MyClass()
{
   _mem = new char[1000];
}
MyClass::~MyClass()
{
   delete[] _mem;
}
```

```
int main()
{
   MyClass a;
   if( ... )
   {
      MyClass b;

   }

}
```

Destructors – common errors *(folder 5)*

1. Forgetting to write a destructor.
   Might cause a memory leak.

2. Calling a destructor twice.
   Possible reason - shallow copy.

Memory allocation in C

```
IntList* L =
(IntList*)malloc(sizeof(IntList));
```

Does not call constructor!

Internal data members are not initialized

```
free(L);
```

Does not call destructor!

Internal data members are not freed

Memory allocation in C++

Special operators:

```
IntList *L = new IntList;
```

1. Allocate memory
2. Call constructor

```
delete L;
```

3. Call destructor
4. Free memory

new

Can be used with any type:

```
int *i = new int;
char **p = new (char *);
```

New & Constructors

```
class MyClass
{
public:
 1 MyClass();
 2 MyClass( int i );
 3 MyClass( double x, double y );

};

MyClass* a;
a = new MyClass; // Calls 1
a = new MyClass {5}; // Calls 2
a = new MyClass { 1.0, 0.0 }; // Calls 3
```

New & arrays

To allocate arrays, use

```
int *a = new int[10]; //array of 10 ints
size_t n = 4;
IntList *array = new IntList[n];
// array of n IntLists.
// - must have a parameterless ctor!
array[0].func();
IntList* b0 = new IntList;
b0→func();
```

Delete & arrays

Special operation to delete arrays

```
int *a = new int[10];
int *b = new int[10];

delete [] a; // proper delete command
delete b;    // apparently works,
// but may cause segmentation fault
// or memory leak (folder 6)
```

Allocate array of objects w/o def. cons.

```
size_t n = 4;
MyClass **arr = new MyClass *[n];
// array of n pointers to MyClass (no
// cons. is invoked)

for (size_t i=0; i<n; ++i)
{
   arr[i] = new MyClass (i);
   // each pointer points to a MyClass
   // object allocated on the heap, and
   // the cons. is invoked.
}
```

Free an allocated array of pointers to objects

```
size_t n = 4;
for (size_t i=0; i<n; ++i)
{
   delete (arr[i]);
   // invoked the dest. of each MyClass
   // object allocated on the heap, and
   // free the memory.
}
delete [] arr;
// free the memory allocated for the
// array of pointers. No dest. is invoked
```

RAII – Resource Acquisition Is Initialization

A class that uses resources (file, memory, database, locks …) should:

- Acquire them in the constructor.
- Release them in the destructor.