

CSC 211: Object Oriented Programming

Dynamic Memory Allocation, Destructors

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Fall 2020



Original design and development by Dr. Marco Alvarez

Dynamic Memory Allocation

The **new** and **delete** operators

- Used to create and destroy variables, objects, or arrays while the program is running
- Memory allocated with the **new** operator does **NOT** use the **call stack**
 - ✓ new allocations go into the **heap** (area of memory reserved for dynamic memory allocation)
- Programmer **must** destroy all variables, objects, and arrays created dynamically
 - ✓ using the **delete** operator

```
#include <iostream>

int main( ) {
    int *p1, *p2;

    p1 = new int;
    *p1 = 10;
    p2 = p1;
    *p2 = 20;
    p1 = new int;
    *p1 = 30;

    std::cout << *p1 << ' ' << *p2 << '\n';

    delete p1;
    delete p2;

    return 0;
}
```

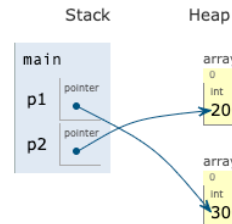
Tracing the code

C++ (gcc 4.8, C++11)
EXPERIMENTAL! known limitations

```
1 #include <iostream>
2
3 int main() {
4     int *p1, *p2;
5
6     p1 = new int;
7     *p1 = 10;
8     p2 = p1;
9     *p2 = 20;
10    p1 = new int;
11    *p1 = 30;
12
13    std::cout << *p1 << ' ' << *p2 << '\n';
14
15    delete p1;
16    delete p2;
17
18    return 0;
19 }
```

Print output (drag lower right corner to resize)

30 20



<http://pythontutor.com/cpp.html#mode=edit>

5

Syntax for new and delete

```
#include "date.h"
#include <iostream>

int main() {
    // creating a single variable
    int *p = new int;
    *p = 5;

    // creating an array
    int *array = new int[20];
    for (int i = 0 ; i < 20 ; i++) {
        array[i] = 0;
    }

    // creating an object
    Date *today = new Date(11, 18, 2019);
    (*today).print();

    // delete all allocated objects
    delete p;
    delete [] array;
    delete today;

    return 0;
}
```

6

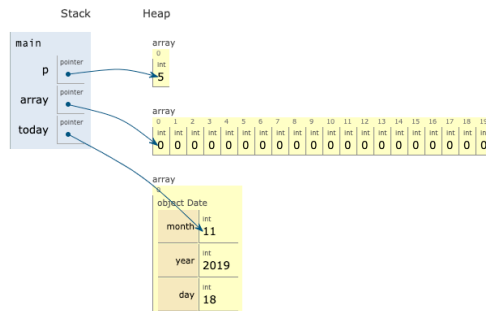
Tracing the code

C++ (gcc 4.8, C++11)
EXPERIMENTAL! known limitations

```
31 int main() {
32     // creating a single variable
33     int *p = new int;
34     *p = 5;
35
36     // creating an array
37     int *array = new int[20];
38     for (int i = 0 ; i < 20 ; i++) {
39         array[i] = 0;
40     }
41
42     // creating an object
43     Date *today = new Date(11, 18, 2019);
44     (*today).print();
45
46     // delete all allocated objects
47     delete p;
48     delete [] array;
49     delete today;
50
51     return 0;
52 }
```

Print output (drag lower right corner to resize)

11-18-2019



<http://pythontutor.com/cpp.html#mode=edit>

7

Pointers and objects

- Data members and methods of an object can be accessed by dereferencing a pointer

```
Date *today = new Date(11, 18, 2019);
(*today).print();
```

- Or ... can use the **-> operator**

```
Date *today = new Date(11, 18, 2019);
today->print();
```

8

Destructors

Destructor

- Special `method` automatically called when objects are destroyed
 - ✓ it is used to delete any memory created dynamically
- Objects are destroyed when ...
 - ✓ ... they exist in the stack and go out of scope
 - ✓ ... they exist in the heap and the delete operator is used
- A destructor ...
 - ✓ ... is a member function (usually public)
 - ✓ ... must have the same name as its class preceded by a ~
 - ✓ ... is automatically called when an object is destroyed
 - ✓ ... does not have a return type (not even `void`)
 - ✓ ... takes no arguments

10

Destructor

- Special `method` automatically called when objects are destroyed
 - ✓ it is used to delete any memory created dynamically
- Objects are destroyed when ...
 - ✓ ... they exist in the stack and go out of scope
 - ✓ ... they exist in the heap and the delete operator is used
- A destructor ...
 - ✓ ... is a member function (usually public)
 - ✓ ... must have the same name as its class preceded by a ~
 - ✓ ... is automatically called when an object is destroyed
 - ✓ ... does not have a return type (not even `void`)
 - ✓ ... takes no arguments

11

```
#ifndef _DYNARRAY_H_
#define _DYNARRAY_H_

class DynArray {
private:
    unsigned int size;
    unsigned int capacity;
    int *array;

public:
    // allocates an array with default capacity
    // default: 25
    DynArray();
    // allocates an array with capacity
    DynArray(int capacity);

    // frees memory allocated by one of the
    // constructors
    ~DynArray();

    // appends a value to the end of the array
    // automatically doubles the capacity when
    // array is full
    // throws exception if memory cannot be
    // allocated anymore
    void append(int value);

    // gets the value at index idx
    // throws exception if idx is invalid
    int get(unsigned int idx);
};
```

12