C++ dynamic memory allocation

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org

JOHNS HOPKINS UNIVERSITY

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

Source markdown available at github.com/BenLangmead/c-cpp-notes

C++ dynamic memory allocation

`new` and `delete` are C++ versions of `malloc` and `free`

Big difference: `new` allocates memory *and calls the appropriate constructor*

Small difference: `new` and `delete` are "keywords" rather than functions, so we don't use `(...)` when calling them

C++ dynamic memory allocation

```
#include <iostream>

using std::cout;  using std::endl;

class DefaultSeven {
public:
    DefaultSeven() : i(7) { }
    int get_i() { return i; }
private:
    int i;
};

int main() {
    DefaultSeven s;
    DefaultSeven *sptr = new DefaultSeven(); // using new
    cout << "s.get_i() = " << s.get_i() << endl;
    cout << "sptr->get_i() = " << sptr->get_i() << endl;
    return 0;
}
```

C++ dynamic memory allocation

```
$ g++ -c new_eg1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o new_eg1 new_eg1.o
$ ./new_eg1
s.get_i() = 7
sptr->get_i() = 7
```

new called the default constructor for us in both cases

C++ dynamic memory allocation

`delete` deletes something allocated with `new`

```
int main() {
    DefaultSeven *sptr = new DefaultSeven();
    ... // use sptr
    delete sptr;
    // note: new and delete don't use parentheses,
    // unlike malloc() / free()
    return 0;
}
```

C++ dynamic memory allocation

`T * fresh = new T[n]` allocates an array of `n` elements of type `T`

Use `delete[] fresh` to deallocate – always use `delete[]` (not `delete`) to deallocate a pointer returned by `new T[n]`

If `T` is a `class`, then `T`'s default constructor is called for *each* element allocated

If `T` is a "built-in" type (`int`, `float`, `char`, etc), then the values are *not initialized*, like with `malloc`

C++ dynamic memory allocation

```
#include <iostream>
using std::cout;  using std::endl;

class DefaultSeven {
public:
    DefaultSeven() : i(7) { }
    int get_i() { return i; }
private:
    int i;
};

int main() {
    DefaultSeven *s_array = new DefaultSeven[10];
    for(int i = 0; i < 10; i++) {
        cout << s_array[i].get_i() << " ";
    }
    cout << endl;
    delete[] s_array;
    return 0;
}
```

C++ dynamic memory allocation

```
$ g++ -c new_eg2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o new_eg2 new_eg2.o
$ ./new_eg2
7 7 7 7 7 7 7 7 7 7
```

Default constructor was indeed called for all 10 elements