C++ dynamic memory allocation

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

Big differences: `new` not only allocates the memory, it also calls the appropriate constructor if used on a class type (more on this later)

Small differences: `new` and `delete` are **keywords** rather than functions, so you don't use `(...)` when calling them

new usage

```
// dynamic1.cpp
1  #include <iostream>
2
3  int main() {
4      int *iptr = new int;
5      *iptr = 10;
6      std::cout << "value of iptr " << iptr << std::endl;
7      std::cout << "value in *iptr " << *iptr << std::endl;
8      return 0;
9  }
```

```
$ g++ -o dynamic1 dynamic1.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./dynamic1
```

```
value of iptr 0x24bcc20
value in *iptr 10
```

delete usage

`delete` deletes something allocated with `new`

`// dynamic2.cpp`

```
1  #include <iostream>
2  int main() {
3      int *iptr = new int;
4      *iptr = 10;
5      // do more with iptr
6      delete iptr;
7      std::cout << "after delete" << std::endl;
8      std::cout << "value in *iptr " << *iptr << std::endl;
9      std::cout << "value of iptr " << iptr << std::endl;
10     // note: new and delete don't use parentheses,
11     // unlike malloc() / free()
12     return 0;
13 }
```

`delete` usage

```
$ g++ -o dynamic2 dynamic2.cpp -std=c++11 -pedantic -Wall -Wextra

$ ./dynamic2

after delete
value in *iptr 0
value of iptr 0x145cc20
```

C++ dynamic array 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 **built-in** type (`int`, `float`, `char`, etc), then the values are *not initialized*, like with `malloc`

If `T` is a `class`, then `T`s default constructor is called for **each** element allocated (more on this soon)

C++ dynamic array allocation in action

```
// dynamic3.cpp
#include <iostream>

int main() {
    double *d_array = new double[10];
    for(int i = 0; i < 10; i++) {
        std::cout << (d_array[i] = i * 2) << " ";
    }
    std::cout << std::endl;
    delete[] d_array;
    return 0;
}
```

```
$ g++ -o dynamic3 dynamic3.cpp -std=c++11 -pedantic -Wall -Wextra
$ ./dynamic3
0 2 4 6 8 10 12 14 16 18
```

Quick in-class exercise I

```
#include <iostream>
#include <sstream>
#include <string>

using std::stringstream; using std::cout;
using std::cin; using std::string; using std::endl;
// TODO: Function to find and print all words
// in str with a length greater than k
void findWords(string str, int K) {
    string word;
    // using stringstream to break the string into tokens
    stringstream ss(str); // create a stringstream with content str
    // TODO: write code to find words in ss with a length greater
    // than k and print them out
}

int main() {
    string str = "Here is a bunch of space separated words";
    int k = 3;
    findWords(str, k);
    return 0;
}
```

Quick in-class exercise II

```
#include <iostream>
#include <vector>

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

void sum(/* TODO: complete this part */) {
    // TODO: complete the function to calculate the sum
    // of all v's values. Make use of references.
}

int main() {
    vector<int> v;
    for (int i = 0; i < 87; i = i + 2) {
        v.push_back(i);
    }
    int result = 0;
    sum (v, result);
    cout << "sum of all v's elements is: " << result << endl;
}
```