Dynamic Memory Allocation

Syntax:

```
type *ptr = new type;   // allocate memory for one variable of the type given
// ...use ptr...
delete ptr;             // deallocate the memory pointed to by ptr
```

OR

```
// allocate memory for num variables of the type given (num is an integer)
type *ptr = new type[num];
// ...use ptr...
delete[] ptr;           // deallocate the memory pointed to by ptr
```

Example 1

```
int *ptr = new int;     // make a new int on the heap
*ptr = 10;              // set it to 10
delete ptr;             // delete it
ptr = nullptr;          // good practice
```

Example 2

```
int *ptr = new int;     // make a new int on the heap
*ptr = 10;              // set it to 10
int *ptr2 = *ptr        // OK; two pointers pointing to that location
delete *ptr2;           // OK; delete the memory through the other pointer
delete *ptr;            // error; can't delete the same memory twice
*ptr = 11;              // error; can't access this memory after deleting it
```

Example 3

```
int *ptr = new int;     // make a new int on the heap
*ptr = 10;              // set it to 10
int *ptr2 = new int;    // make a second int on the heap
*ptr2 = 20;             // set it to 20
int *temp = ptr;
ptr = ptr2;
ptr2 = ptr;             // ptr now points to 20, ptr2 points to 10
delete ptr1;            // OK
delete ptr2;            // OK
delete temp;            // error; temp points to 10, which has already been deleted
```

Example 4

```
double *ptr = new double[3];  // make an array of 3 doubles on the heap
ptr[0] = 5;                   // OK
ptr[1] = 10;                  // OK
ptr[2] = 15;                  // OK
ptr[3] = 20;                  // index out of bounds (C++ will not flag it, though!)
cout << ptr[0] << ptr[1] << ptr[2] << endl;     // all OK
delete[] ptr;                 // OK
ptr[0] = 30;                  // error; the memory has been deleted
```