**Classes and Objects**

1. **Class definition**

To create a class (should be defined in header file)

We can define attributes and methods: (We can always access class attributes inside methods)

Text

Description automatically generated

//<-- notice this semicolon!

\*\* Split function declarations from definitions. We declare methods inside the class (in a header), then define the methods outside the class (in a .cpp file of the same name, #include the header file).

To define a method outside a class, we add Classname:: before method name

Graphical user interface, text, application, email

Description automatically generated

1. **Access Control: Public and Private**

By default, everything in a class is private, meaning class members (attributes and methods) are limited to the scope of the class. If you try to access a private class member, you’ll get an error.

It is a good pratice to use getters and setters (public) and all variables be private.

For getters (and searching functions), we can add a **const** keyword at the end, to catch error whenever the function is modifying the variables. We can also add const to protect parameter.

Node\* search (int) const;

NOTE that **In C++, access control is per-class, not per-object!**

* Objects of a class also cannot access its private members

1. **Instantiating Objects**

To instantiate an object (**on stack**), we can do like how we create a new instance of int, double:

<Classname> <InstanceName> ;

Then we can access/change the attributes and functions of the objects accordingly using “.”

To instantiate object **on the heap**, we can do:

Cat \*andrew;

andrew = **new** Cat;

andrew -> resetName(); // equal to (\*andrew).resetName();

**new** will return a **address to the object** (that fits with the pointer)

If we do this way, meaning that we have an object declared in the heap

🡪 We will have to delete the object. **delete** andrew

1. **Constructor**

A constructor is a special kind of method that defines how an object of a class is created.

It has the same name as the class and no return type **(not even void)**

**Constructors must be public**

We can have **Multiple Constructors.** A default constructor is not required, even if we have custom constructors.

// time.h

Text

Description automatically generated

**Example: Linked list**

// Node.h // node.cc

Text

Description automatically generated Text

Description automatically generated

// "**const**" means these 2 member functions

cannot modify any member variable

(can still modify other variables)

// linked\_list.h

Text

Description automatically generated

1. **Destructors**

Whenever we call **new**, we should consider **delete** to free the memory on the heap.

* However, NOTE that if we do:

Cat\* Tom = new Cat();

delete Tom;

* + Say that Tom dynamically declares an array on the heap (as a member variable)
  + delete Tom will free Tom, but the array on the heap isn't! 🡪 Memory Leak!
  + Therefore, we have to write more code, to free all the values inside.

🡪 Destructors!

Text

Description automatically generated

|  |
| --- |
| We generally don’t need to call the destructor; it will be called automatically in any of the scenarios:   * The object moves out of scope. * The object is explicitly deleted. * When the program ends. |

It has the same name as the class and no return type, but is preceded by a ~ operator and takes no parameters.

**\*\*Note that we do delete name only, that will only delete the first element.**

**If we do new[] we must do delete[]**

So in general, we are using destructors to clean up the memory allocated by the implementation of the class itself. For example, you match **new** to a member of the class 🡪 If we only delete the object, we will leak the memory pointed by the members.

**?** We don't need to create destructors if we are not allocating new memory inside the class.