Defining a Class

class Class_Name {
private:
    Include private data and functions.
public:
    Include public data and functions.
};
Notice the ; after }.

public and private

- Usually, data items will be private (to be accessed or changed via member functions).
- Private functions are used by the class itself – not available outside. These are often referred to as utility functions.
- Public member functions form the interface between class objects and the user. They can be used to access or change private data.

Two important operators:

1. Scope resolution operator (::). This is used to scope an identifier back to the class in which it was defined – commonly used to give a function definition prototyped inside a class definition.
2. Dot operator (.) Used to access public data or functions of a class. Usage object.function_name( ) or object.data_item
More on dot operator

When writing (defining) member functions or friend functions, the dot operator can be used to access private as well as public items in the class.

NOTE: friend functions have not yet been defined, so don’t worry about that now.

Terms to Know

- data type
- abstract data type (ADT)
- information hiding
- data abstraction
- encapsulation

These are discussed in the text. Know what they mean.

A sample class

Suppose we desire to build a new type of integer that would allow up to 50 digits for an integer. We need to begin by asking some questions.
Questions about our integers

- What’s a good name for the class?
- What do we wish to do with these integers?
- What data items are required?
- What member functions are required?
- Are there any other questions you can think of?

Next Step?

Define the class! As you go about this process, recall how member functions are called by objects. (dot operator)
Pay close attention to what is public and what is private.

Class Constructors

- A constructor is called when an object of a given class is declared or instantiated. It sets up the object and may or may not assign values to the data items in the object.
- You as the creator of a class, can provide constructors to effect your own implementation.
- If you don’t provide a constructor, a default one is automatically provided; however, you can’t be sure what is in the object at instantiation.
Creating Your Own Constructor(s)

- It should be a public member function.
- It should have no return type.
- It should have the same name as the class.
- It may or may not have arguments.
- Via function overloading (different argument lists) you can have as many constructors as you need.

Example - Three Constructors

```cpp
class foo {
    private:
        int n, m;
    public:
        foo();  // default constructor
        foo(int);  // constructor with one argument
        foo(int, int);  // constructor with two arguments  
                        // plus other public member functions
    };
```

Declaring foo objects:

```cpp```
```cpp
foo a, b(5), c(5, 2);
```
```
This defines three objects and constructors are called accordingly, based on the overloaded function definitions.

WARNING: In calling the constructor with no arguments for object a, we did not write a().
Read why on page 267. A constructor with no arguments is a default constructor. It is a good idea to always include your own default constructor. Read why on page 266.
```
Use of the const modifier

There are two primary uses of the const modifier when working with class definitions.
1. Declaring a member function to be constant.
2. Declaring a parameter that is passed by reference as constant.

Declaring a member function as constant.

If a member function in a class does not change any data item within the class, it should be declared with const. How? Place the const modifier immediately following the closing parenthesis of the argument list.

```cpp
void PrintObject( ) const;
```

Result of declaring a function with const

Declaring the member function as constant means that the function cannot alter any data item. The dot operator linking an object declared with const can only be used to link functions that are also declare with const.
Example

Consider the following using a class definition for a Fraction object.
const Fraction X(2,5);
X.PrintFraction();

Even though PrintFraction does not modify the fraction X, this call could be done only if PrintFraction has been defined using const.

Another result of const.

If a member function has been declared as constant, any attempt within the definition to alter a data item in the object will result in a compiling error; that is, data items cannot be left values.

Declaring reference parameters as constant.

When passing user-defined objects as parameters to functions, it is often desirable to pass them by reference. Why?
1. It’s more efficient in that large amounts of data items don’t have to be copied.
2. Copying an object that contains pointers can be problematic unless appropriate copy constructors are created.
Passing a constant object to a member function by reference is not allowed unless the corresponding reference parameter is also declared constant. This has to be the case since functions can change reference parameters otherwise.

Example using the LargeInt class

Consider the member function:

```cpp
void AddLargeInt(
    const LargeInt &);
```

Recall that this was to add the argument LargeInt to the calling object and put the result in the calling object. So the calling object is changed, but the argument object is not. Thus the reference argument is constant and the member function itself is not constant.

The Example continued:

Now we could have the following:

```cpp
LargeInt X;
const LargeInt Y("111222333");
...code to initialize these
X.AddLargeInt(Y);
```

Since Y is constant, it can only be passed by reference to a parameter declared with const.
Summary

A non-constant object may be passed by reference to a constant parameter. A constant object may not be passed by reference to a non-constant parameter. The dot operator may link a non-constant object to a member function that is constant or one that is non-constant. The dot operator may not link a constant object to a non-constant member function.

Good rules to follow

If a member function does not change an object, declare it constant. This will allow greatest access by constant and non-constant objects. When passing an object to member functions by reference, if the member function does not alter the argument object, declare the corresponding parameter as constant.

Self-Exercise

Go back over the class examples we have done. Which member functions should be constant? Which arguments should be passed by reference, and which of those should be constant?
Inline Functions

When a member function is declared in the class, it's definition can be included at that point. Normally this is done for only short function definitions. Look at the example on page 285 of your text. Which functions are inline? Which functions in the tic-tac-toe program are inline?

Inline functions (cont.)

As indicated in your text, inline function definitions is more than just a "notational variant". What is the difference? How are these handled differently than externally (outside the class) defined functions?