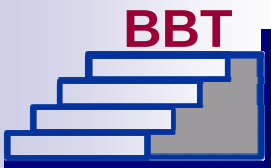

CPP

Object oriented concept

Class and object

Inline function

- Inlining replaces method calls with a macro-like expansion of the called method within the calling method.
- Format:
  - **inline** *function definition*

Separate Compilation

- some things commonly found in header files:
  - Function prototypes
  - Symbolic constants defined using #define or const
  - Structure declarations
  - Class declarations
  - Template declarations
  - Inline functions

Variable Storage

Storage	Duration	Scope	Linkage	Declaration
automatic	automatic	block	none	in a block (optionally with the keyword **auto**)
register	automatic	block	none	in a block with the keyword **register**
static with no linkage	static	block	none	in a block with the keyword **static**
static with external linkage	static	file	external	outside of all functions
static with internal linkage	static	file	internal	outside of all functions with the keyword **static**

Namspace

namespace *space*{
    // any declarations …
}

- Namespaces can be located at the global level or inside other namespaces, but they cannot be placed in a block
- Namespaces are open, meaning you can add names to existing namespaces

Using-Declarations and Using-Directives

- Declarations:

using *space*::*name;*

  - A using-declaration adds a particular name to the declarative region in which it occurs. Preclude decelerate other variable as the same name in this region.

- Directives

using namespace *space;*

  - the using-directive makes all the names available

Object-oriented programming

- Abstraction
- Encapsulation and data hiding
- Polymorphism
- Inheritance
- Reusable code

Class

- a class specification has two parts
  - A class declaration, which describes the data component, in terms of data members, and the public interface, in terms of member functions
  - The class method definitions, which describe how certain class member functions are implemented

Public and private

- These labels describe access control for class members.
  - Any program that uses an object of a particular class can access the **public** portions directly.
  - A program can access the **private** members of an object only by using the public member functions
- **Encapsulation** is gathering the implementation details together and separating them from the abstraction**.**
- **Data hiding** (putting data into the private section of a class) is an instance of encapsulation and so is hiding functional details of an implementation in the private section

Implementing Class Member Functions

- When you define a member function, you use the scope operator (::) to identify the class to which the function belongs.
- Class methods can access the private components of the class.

Inline Methods

- Any function with a definition in the class declaration automatically becomes an inline function.

Constructors

- C++ provides for special member functions, called class constructors, especially for constructing new objects and assigning values to their data members.
  - The name is the same as the class name
  - the constructor has no return value, It has no declared type.
- The Default Constructor
  - The default constructor is the constructor used to create an object when you don't provide explicit initialization values.

Destructors

- The program undertakes the responsibility of tracking that object until it expires. At that time, the program automatically calls a special member function destructor
- the destructor has a special name: the class name preceded by a tilde (~).

const Member Functions

- Class const member functions declared and defined are to place the const keyword after the function parentheses.
- you should make class methods const whenever they don't modify the invoking object.

this Pointer

- The this pointer points to the object used to invoke a member function.

An Array of Objects

- If you want to create an array of class objects, the class must have a default constructor.

Class Scope Constants

- Example:

```
class Stock{
private:
    enum {Len = 30};  // class-specific constant
    char company[Len];
```

- Or

```
class Stock{
private:
    static const int Len = 30;    // declare a constant! WORKS
    char company[Len];
```

- You only can use this technique for declaring static constants with integral and enumeration values. You can't store a double constant this way.

Operator Overloading

- To overload an operator, you use a special function form called an operator function. An operator function has the form:

operator*op*(*argument-list*)

Friends

- C++ provides another form of access, the friend. Friends come in three varieties:
  - Friend functions
  - Friend classes
  - Friend member functions
- To create a friend function is to place a prototype in the class declaration, prefixing the declaration with the keyword **friend**

Automatic Conversions

- A constructor that can be used with just one argument works as a conversion function. This is implicit conversion.
- Recent C++ implementations have a new keyword, **explicit**, to turn off the automatic aspect.

Type Casts for Classes

- To convert an object to type typeName, use a conversion function of this form:

  operator typeName();

- Note the following points:
  - The conversion function must be a class method.
  - The conversion function must not specify a return type.
  - The conversion function must have no arguments.

Implicit Member Functions

- A default constructor if you define no constructors
- A copy constructor if you don't define one
- An assignment operator if you don't define one
- A default destructor if you don't define one
- An address operator if you don't define one

The Copy Constructor

- The copy constructor is used to copy an object to a newly created object. That is, it's used during initialization, not during ordinary assignment. The copy constructor for a class has this prototype:

```
Class_name(const Class_name &);
```

The Assignment Operator

- C++ allows class object assignment. It does so by automatically overloading the assignment operator for a class. This operator has the following prototype:

```
Class_name & Class_name::operator=(const Class_name &);
```