CSC 181F Lecture Notes

Reading Assignment

	K.N. King	Chapter 19
	D.B. Wortman	Slides 278 .. 289
Supplementary Reading		
	S. McConnell	Chapter 6

References for C++

Bjarne Stroustrup
The C++ Programming Language (3rd ed.)
Addison-Wesley, 1997

S.B. Lippman
C++ Primer (3nd ed)
Addison-Wesley, 1997

Scott Meyers
Effective C++ (2nd ed.)
Addison-Wesley, 1998

Richard Johnsonbaugh and Martin Kalin
Object-Oriented Programming in C++
Prentice Hall, 1995

C++ – Overview

- Classes
  - information hiding
  - guaranteed initialization and finalization
  - dynamic typing
  - user controlled memory management
  - overloaded operators
- Other advantages
  - better type checking
  - better exception handling
  - overloaded function names
  - class templates
  - default function arguments
  - references as well as pointers

Miscellaneous Minor Extensions in C++

- Comments: beginning with // and end at the first new-line character.

  // This is a comment.

  // So is this.

- Simpler type casting: *type-name* (identifier)

  final = **float** (your_mark) * 1.4 + 2.0;

- Tags are automatically type names.

  struct Complex { **double** re, im };

  is equivalent to

  typedef struct { **double** re, im } Complex;

- Variable definitions may occur at the point at which they are first used.

  for (**int** J = 0 **;** J < N **;** J+ +) **...**

Reference Type

- References provides an alternative name (**an alias**) for storage

  Example: **int** X **;**

  int & refX = X **;**

  You *must* initialize a reference variable when you declare it.

  Compiler will automatically compute addresses as required.

- Parameters may be passed by reference (passing an address)

  Example: **void** swap(**int** & A **, int** & B) **;**

  ...

  swap(J **,** K) **;**

  This allows you to alter a data object in the calling function

  Compiler automatically generates the reference at the point of call.

  No more forgotten & s.

- Functions may return a reference

  struct NODE & makeNode(**int** value) **;**

  WARNING: Don't return a reference to a variable local to the function.

Function Parameters & Inline Functions

- Functions with no argument in a function prototype are interpreted as specifying no parameters:

  int F( ) **;** is equivalent to **int** F(**void**) **;**

- Inline functions: **inline** is a *request* that a function be expanded "inline".

  - Place the keyword **inline** before the function definition

    Place the function definition above all the functions that call it

  inline float cube(**float** s) { **return** s * s * s }

  ...

  Z = cube(X) **;** /* Z = X * X * X ; */

  Y = cube(Z + 5.0) **;** /* Y = (Z+5.0)*(Z+5.0)*(Z+5.0) **;** */

Default Function Arguments

- Default values may be supplied for function arguments in the prototype for a function.

- If arguments are missing in the invocation of the function, the default values are used.

- Example:

  void F(**int** val **, float** S = 12.6 **, char** T = '\n' **, char *** msg = "Error") **;**

  f(14, 48.3, '\ t', "OK") **;**

  f(14, 48.3, '\ t') **;**

  f(14, 48.3) **;**

  f(14) **;**

- The defaults must be added from right to left.

  A parameter without defaults cannot occur after a parameter with defaults.

- The arguments are assigned to the corresponding parameters from left to right; you cannot skip over arguments.

Function Overloading

- C++ permits identically named functions within the same scope if they can be distinguished by number and type of parameters (**signature**).

  void print (**int** i){ printf("%d\n", i); }

  void print (**char** *s){ printf("%s\n", s);}
- Compiler considers a reference to a type and the type itself to be the same. Compiler discriminates between **const** and non-**const** variables. Parameter Signature is used to resolve overloading, not the function return type.
- C++ operators can be overloaded except for

  :: (scope resolution) . (member selection)

  .* (member selection through pointer)
- Use function overloading when functions perform basically the same task but with different forms of data.

C++ Operators: new and delete

- **new** operator allocates storage dynamically

  int * int_ptr1 = **new int**

  new returns the null pointer if no storage is available.
- **new** operator can allocate an arbitrary number of contiguous cells dynamically

  int *int_ptr2 = **new int** [50];

  If successful, the first cell's address is stored in int_ptr2.
- **delete** and **delete** [ ] free storage allocated by **new** .

  delete int_ptr1;

  delete [ ] int_ptr2;
- **WARNING:** The operators **new** , **delete** , and **delete** [ ] should be used together and not intermixed with C storage management function.

Preview of C++ Input and Output

- Input and Output in C++ are provided by standard I/O libraries. `iostream.h` defines the most widely used C++ I/O library.
- Predefined objects, `cin` (standard input), `cout` (standard output), and `cerr` (standard error) are available.
- Input (extraction) is performed by the extraction operator (>>). Output (insertion) is performed by the insertion operator (<<).

C++ Input and Output Example

```
#include < iostream.h>
main()
{
    int val1, val2 ;
    cout << "Please enter two integers: " << endl
    cin >> val1 >> val2 ;

    cout << "The sum of " << val1 << " and " << val2
        << " is " << val1 + val2 << endl ;
}
```

Note that `cout << endl` writes newline and flushes output stream.