C++ Programming HOW−TO

Table of Contents

Table of Contents

C++ Programming HOW−TO

Al Dev (Alavoor Vasudevan) alavoor@yahoo.com

v35.0, 14 Mar 2001

This document provides a comprehensive list of C++ URL pointers, links to C++ online textbooks, and programming tips on C++. This document also provides a C++ library which imitates Java−language, and which has various methods to avoid memory problems in C++. Using this library you can compile Java's source code under C++. This document serves as a "Home of C++ language". The information given here will help you to program properly in C++ language and applies to all the operating systems that is − Linux, MS DOS, BeOS, Apple Macintosh OS, Microsoft Windows 95/98/NT/2000, OS/2, IBM OSes (MVS, AS/400 etc..), VAX VMS, Novell Netware, all flavors of Unix like Solaris, HPUX, AIX, SCO, Sinix, BSD, etc.. and to all other operating systems which support "C++" compiler (it means almost all the operating systems on this planet).

---

1. Introduction

The purpose of this document is to provide you with a comprehensive list of URL pointers and programming tips on C++. Also, this document provides a C++ library having Java−like String class, string tokenizer, memory functions and many other functions, which can be used in general C++ applications. Also various examples are given here which demonstrate the usage of this library.

This document is not a textbook on C++, and there are already several excellent "on−line Text books" on internet. If you are new to C++ and you never programmed in C++, then it is strongly suggested that you first read the online C++ Textbooks given in the chapter C++ Online Textbooks and then follow the subsequent chapters. It is suggested that you purchase a textbook on C++ for reference from online bookstores.

1.1 C++ v/s Java

C++ is one of the most powerful language and will be used for a long time in the future inspite of emergence of Java. C++ runs **extremely fast** and is in fact **10 to 20 times FASTER than** Java. Java runs very slow because it is a byte−code−interpreted language running on top of "virtual machine". Java runs faster with JIT (Just−In−Time) compiler, but it is still slower than C++. And optimized C++ program is about **3 to 4 times faster** than Java (with JIT compiler). Then, why do people use Java? Because it is easier to program in Java, as Java automates memory management, and programmers do not directly deal with memory allocations. This document attempts to automate the memory management in C++ to make it much more easy to use. The library given here will make C++ look like Java and will enable "C++" to compete with Java language.

Because of manual memory allocations, debugging the C++ programs consumes a major portion of time. This document will give you some better ideas and tips to reduce the debugging time.

1.2 Which one "C", "C++" or Java ?

It is recommended you do programming in object−oriented "C++" for all your application programming or general purpose programming. You can take full advantage of object oriented facilities of C++. The C++ compiler is lot more complex than "C" compiler and C++ programs may run bit slower than "C" programs. But speed difference between "C" and "C++" is very minute − it could be few milli−seconds which may have little impact for real−time programming. Since computer hardware is becoming cheaper and faster and memory 'RAM' is getting faster and cheaper, it is worth doing code in C++ rather than "C" as time saved in clarity and re−usability of C++ code offsets the slow speed. Compiler optimizer options like −O or −O3 can speed up C++/C which is not available in Java.

Nowadays, "C" language is primarily used for "systems programming" to develop operating systems, device drivers etc..

Note: ***Using the String, StringBuffer, StringTokenizer and StringReader classes given in this howto, you can code in C++ which "exactly" looks like Java. This document tries to close the gap between C++ and Java, by imitating Java classes in C++***

Java is platform independent language more suitable for developing GUI running inside web−browsers (Java applets) but runs very slow. Prefer to use web−server−side programming "Fast−CGI" with C++ and HTML, DHTML, XML to get better performance. Hence, the golden rule is *"Web−server side programming use C++ and web−client side (browser) programming use Java applets"*. The reason is − the server−side OS (Linux) is under your control and never changes, but you will never know what the client side web−browser OS is. It can be Internet appliance device (embedded linux+netscape) or computers running Windows 95/98/NT/2000 or Linux, Apple Mac, OS/2, Netware, Solaris etc..

The advantage of Java language is that you can create "Applets (GUI)" which can run on any client OS platform. Java was created to replace the Microsoft Windows 95/NT GUI APIs like MS Visual Basic or MS Visual C++. In other words − "Java is the cross−platform Windows−GUI API language of next century". Many web−browsers like Netscape supports Java applets and web−browser like Hot Java is written in java itself. But the price you pay for cross−platform portability is the performance, applications written in Java run very slow.

Hence, Java runs on "client" and C++ runs on servers.

1.3 Problems facing the current C++ compilers

Since C++ is super−set of C, it got all the bad features of "C" language.

For example, in "C" programming − memory leaks, memory overflows are very common due to usage of features like −

```
        Datatype  char * and char[]
        String functions like strcpy, strcat, strncpy, strncat, etc..
        Memory functions like malloc, realloc, strdup, etc..
```

The usage of **char *** and **strcpy** causes *horrible* memory problems due to *"overflow"*, *"fence past errors"*, *"step−on−others−toe"* (hurting other variable's memory locations) or *"memory leaks"*. The memory problems are extremely hard to debug and are very time consuming to fix and trouble−shoot. Memory problems bring down the productivity of programmers. This document helps in increasing the productivity of programmers via different methods addressed to solve the memory defects in "C++". Memory related bugs are very tough to crack, and even experienced programmers take several days or weeks to debug memory related problems. Memory bugs may be hide inside the code for several months and can cause unexpected program crashes. The memory bugs due to usage of **char *** and **pointers** in C/C++ is costing $2 billion every year in time lost due to debugging and downtime of programs. If you use **char *** and **pointers** in C++ then it is a very costly affair, especially if your program size is greater than 10,000 lines of code.

Hence, the following techniques are proposed to overcome the faults of "C" language. Give preference in the following order −

1. Use references instead of pointers.
2. Java style String class (given in this howto) or STDLib string class.
3. Character pointers (char *) in C++ **limit the usage** of char * to cases where you cannot use the String class.
4. Character pointers (char *) in C using extern linkage specification, if you do not want to use (char *) in C++.

To use "C char *", you would put all your "C" programs in a separate file and link to "C++" programs using the *linkage−specification* statement **extern "C"** −

```
extern "C" {
#include <stdlib.h>
}

extern "C" {
        comp();
        some_c_function();
}
```

The **extern "C"** is a linkage specification and is a flag that everything within the enclosing block (brace−surrounded) uses C linkage, not C++ linkage.

The **'String class'** utilises the constructor and destructor features to automate memory management and provides access to functions like *ltrim*, *substring*, etc..

See also related **'string class'** in the C++ compiler. The **string class** is part of the standard GNU C++ library and provides many string manipulation functions. Because the C++ **'string class'** and **'String class'** library provides many string manipulation functions, there is less need to use the character pointer approach to write your own string functions. Also, C++ programmers must be encouraged to use 'new', 'delete' operators instead of using 'malloc' or 'free'.

The **'String class'** does everything that **char *** or **char []** does. It can completely replace **char** datatype. Plus added benefit is that programmers do not have to worry about the memory problems and memory allocation at all.

1.4 COOP − C++ Object Oriented Programming−language

A problem with C++ is that it is a superset of C, and, although programmers can use the good (object oriented) features of C++ and avoid the bad features of C, there is nothing to force them to do so. So, many C++ programs are written with no object oriented features and continue to use the bad features of C that the use of C++ should have overcome.

Therefore, I propose that we create a new version of C++ that does not allow the use of the bad features of C.

I propose that this new version of C++ be called **COOP** (say koop), which is an acronym for **C++ O**bject **O**riented **P**rogramming−language" . COOP should be pronounced like chicken coop. (The logo of COOP language is a big fat Hen inside coop!) I propose that the file extension for COOP files be .coo, which will not conflict with .c for C programs or .cpp for C++ programs.

To begin with write the COOP as a front end to C++. That is COOP pre−processes the code syntax and then uses the standard C++ compiler to compile the program. COOP acts as a front end to C++ compiler.

The following are some other proposed features of COOP:

- Is a super−set of C++ language but will force programmer to use obejct oriented programming.
- Pure Object−oriented langauge but retains syntax of C++.
- Remove all bad features of C++ in COOP.
- Prevent writing "C" like programming in COOP, something which C++ currently allows. Delete all C features which are considered bad or redundant/duplicates, like printf, fprintf, malloc, struct, free etc..
- No downward compatibility to "C" language.
- Code written in COOP will be easy to maintain and is easily understandable/readable.
- Code written in "COOP" will be re−usable (thru components, modules, objects). Supports re−usable software components, thereby facilitating Rapid Application Development.
- COOP is simple, robust, OOP, has bare mininum syntax (avoiding confusing, redundant, extra constructs of C++)

Also borrow ideas from −

- Java − Sun Microsystem put lot of effort, and you can simply utilize that.
- C# − Microsoft put lot of efforts, and you can simply utilize them. Specs are at http://msdn.microsoft.com/vstudio/nextgen/technology/csharpdownload.asp and see C# overview.
- Connective C++ at http://www.quintessent.com/products/cc++.

---

2. String Class Varieties

The string class is the most vital object in programming, and string manipulations are most extensively used and they comprise of 20 to 60% of total code. There are 3 variety of string classes. Ofcourse, you can build your own string class by simply inheriting from these string classes −

- String class given in this document Appendix A String.h

- GNU string class
    - GNU C++ Library − Univ of Tech, Sydney http://www.socs.uts.edu.au/doc/gnuinfo/libg++/libg++_18.html and user's guide
    - mirror site Gesellschaft http://www−aix.gsi.de/doc/gnu/libg++_18.html#SEC23 and user's guide
    - mirror site Techno, Russia http://www.techno.spb.ru/~xbatob/FAQ/GNU/libg++_19.html#SEC27 and user's guide
    - mirror site Univ of Utah http://www.math.utah.edu/docs/info/libg++_19.html#SEC27 and user's guide

- Qt String class at http://doc.trolltech.com/qstring.html mirror at http://www.cs.berkeley.edu/~dmartin/qt/qstring.html

* If none of these alternatives are suitable, you can build your own string class. You can start with one or more of the pre−built classes listed above (by using single or multiple inheritance.)

2.1 Multiple Inheritance − Sample Custom String class

As mentioned above, you can build your own custom string class from the pre−built classes by single or multiple inheritance. In this section we will build a sample custom string class by using multiple inheritance, inheriting from the GNU string class and the string class presented in Appendix H.

Start by downloading the sample file 'string_multi.h' from Appendix A . That file is reproduced below:

```
// ******************************************************************
// Sample program to demonstrate constructing your own string class
// by deriving from the String class and stdlib's "string" class
// ******************************************************************

#ifndef __STRING_MULTI_H_
#define __STRING_MULTI_H_

#include <string>
#include "String.h"

// Important Notes: In C++ the constructors, destructors and copy
//       operator are NOT inherited by the derived classes!!
//       Hence, if the operators like =, + etc.. are defined in
//       base class and those operators use the base class's contructors
//       then you MUST define equivalent constructors in the derived
//       class. See the sample given below where constructors mystring(),
//       mystring(char[]) are defined.
//
//       Also when you use operator as in atmpstr + mstr, what you are really
//       calling is atmpstr.operator+(mstr). The atmpstr is declared a mystring

class mystring:public String, string
{
        public:
                mystring():String() {}  // These are needed for operator=, +
                mystring(char bb[]):String(bb) {}  // These are needed for operator=, +
                mystring(char bb[], int start, int slength):String(bb, start, slength) {}
                mystring(int bb):String(bb) {}  // needed by operator+
                mystring(unsigned long bb):String(bb) {}  // needed by operator+
                mystring(long bb):String(bb) {}  // needed by operator+
                mystring(float bb):String(bb) {}  // needed by operator+
                mystring(double bb):String(bb) {}  // needed by operator+
                mystring(const String & rhs):String(rhs) {}  // Copy Constructor needed by operator+
                mystring(StringBuffer sb):String(sb) {}  // Java compatibility
                mystring(int bb, bool dummy):String(bb, dummy) {}  // for StringBuffer class

                int mystraa; // customizations of mystring
        private:
                int mystrbb; // customizations of mystring
};

#endif // __STRING_MULTI_H_
```

3. Download String

All the programs, examples are given in Appendix of this document. You can download as a single tar zip, the String class, libraries and example programs from

- Go here and click on C++Programming howto.tar.gz file http://www.aldev.8m.com
- Mirror sites : http://aldev.webjump.com, angelfire, geocities, virtualave, bizland, theglobe, spree, infoseek, bcity, 50megs

---

4. Usage of String class

To use String class, you should first refer to a sample program "example_String.cpp" given in Appendix A and the String class which is given in Appendix A.

The **'String class'** is a complete replacement for char and char * datatype. You can use **'String class'** just like char and get much more functionalities. You should link with the library 'libString.a' which you can build from the makefile given in Appendix A and copy the library to /usr/lib or /lib directory where all the "C++" libraries are located. To use the 'libString.a' compile your programs like −

```
        g++ example.cpp -lString
```

See illustration sample code as given below −

```
        String aa;

        aa = " Creating an Universe is very easy ";

        // You can use aa.val() like a 'char *' variable in programs
        for (unsigned long tmpii = 0; tmpii < aa.length(); tmpii++)
        {
                //fprintf(stdout, "aa.val()[%ld]=%c ", tmpii, aa.val()[tmpii]);
                fprintf(stdout, "aa[%ld]=%c ", tmpii, aa[tmpii]);
        }

        // Using pointers on 'char *' val ...
        for (char *tmpcc = aa.val(); *tmpcc != 0; tmpcc++)
        {
                fprintf(stdout, "aa.val()=%c ", *tmpcc);
        }
```

4.1 Operators

The **'String class'** provides these operators :−

- Equal to **==**
- Not equal to **!=**

- Assignment **=**
- Add to itself and Assignment **+=**
- String concatenation or addition **+**

For example to use operators −

```
String aa;
String bb("Bill Clinton");

aa = "put some value string";  // assignment operator
aa += "add some more"; // Add to itself and assign operator
aa = "My name is" + " Alavoor Vasudevan "; // string cat operator

if (bb == "Bill Clinton")  // boolean equal to operator
        cout << "bb is equal to 'Bill Clinton' " << endl;

if (bb != "Al Gore")   // boolean 'not equal' to operator
        cout << "bb is not equal to 'Al Gore'" << endl;
```

4.2 Functions

The functions provided by String class has the **same name** as that of Java language's String class. The function names and the behaviour is **exactly** same as that of Java's String class. StringBuffer class is also provided. This will facilitate portability of code between Java and C++ (you can cut and paste and do minimum changes to code). The code from Java's function body can be copied into C++ member function body and with very mininum changes the code will compile under C++. Another advantage is that developers coding in both Java and C++ do not need to remember two different syntax or function names.

For example to convert integer to string do −

```
String  aa;

aa = 34;  // The '=' operator will convert int to string
cout << "The value of aa is : " << aa.val() << endl;

aa = 234.878;  // The '=' operator will convert float to string
cout << "The value of aa is : " << aa.val() << endl;

aa = 34 + 234.878;
cout << "The value of aa is : " << aa.val() << endl;
// The output aa will be '268.878'

// You must cast String to convert
aa = (String) 34 + " There can be infinite number of universes!! " + 234.878;
cout << "The value of aa is : " << aa.val() << endl;
// The output aa will be '34 There can be infinite number of universes!! 234.878'
```

Refer to Appendix A String.h for details about the String class function names. The same file String.h is reproduced here in next section.

5. String.h file

In C++ (or any object oriented language), you just read the "class data−structure" (i.e. interface) to begin using that object. You just need to understand the interface and not the implementation of the interface. In case of String class, you just need to read and understand the String class in String.h file. You **do not need** to read the entire implementation (String.cpp) in order to use String class. The object oriented classes are real time saver and they **very neatly hide** the implementation.

(In object oriented Java language there is the equivalent called **'Intefaces'** , which hides the implementation details.)

Given below is **String.h** file and see also Appendix A String.h

---

```
//
// Author : Al Dev  Email: alavoor@yahoo.com
// Use string class or String class
//
// To prevent memory leaks - a char class to manage character variables
// Always prefer to use String or string class
// instead of char[] or char *
//

#ifndef __STRING_H_
#define __STRING_H_

#include <iostream> // do not use iostream as program becomes bulky..
#include <stdio.h> // for FILE and sprintf()
#include <list.h> // for list

const short INITIAL_SIZE =      50;
const short NUMBER_LENGTH = 300;
const int MAX_ISTREAM_SIZE = 2048;

class StringBuffer;

// a small class with a VERY MINIMUM of functions and variables...
// This class to be kept small...
class String
{
        public:
                String();
                String(const char bb[]);  // needed by operator+
                String(const char bb[], int start, int slength); // subset of chars
                String(int bb);  // needed by operator+
                String(unsigned long bb);  // needed by operator+
                String(long bb);  // needed by operator+
                String(float bb);  // needed by operator+
                String(double bb);  // needed by operator+
                String(const String & rhs);  // Copy Constructor needed by operator+
                String(StringBuffer sb);  // Java compatibility
                String(int bb, bool dummy);  // for StringBuffer class
                virtual ~String();  // Made virtual so that when base class is deleted
                                                        // then the derived class destructor is called.

                char *val() {return sval;} // It is not safe to make sval public

                // Functions below imitate Java language's String object
```

```
unsigned long length() { return strlen(sval); }
char charAt(int where);
void getChars(int sourceStart, int sourceEnd,
                char target[], int targetStart);
char* toCharArray();
char* getBytes();

bool equals(String str2); // See also == operator
bool equals(char *str2); // See also == operator
bool equalsIgnoreCase(String str2);

bool regionMatches(int startIndex, String str2,
                int str2StartIndex, int numChars);
bool regionMatches(bool ignoreCase, int startIndex,
                String str2, int str2StartIndex, int numChars);

String toUpperCase();
String toLowerCase();

bool startsWith(String str2);
bool startsWith(char *str2);

bool endsWith(String str2);
bool endsWith(char *str2);

int compareTo(String str2);
int compareTo(char *str2);
int compareToIgnoreCase(String str2);
int compareToIgnoreCase(char *str2);

int indexOf(char ch, int startIndex = 0);
int indexOf(char *str2, int startIndex = 0);
int indexOf(String str2, int startIndex = 0);

int lastIndexOf(char ch, int startIndex = 0);
int lastIndexOf(char *str2, int startIndex = 0);
int lastIndexOf(String str2, int startIndex = 0);

String substring(int startIndex, int endIndex = 0);
String replace(char original, char replacement);
String replace(char *original, char *replacement);

String trim(); // See also overloaded trim()

String concat(String str2);  // See also operator +
String concat(char *str2); // See also operator +

String reverse(); // See also overloaded reverse()
String deleteCharAt(int loc);
String deleteStr(int startIndex, int endIndex); // Java's "delete()"

String valueOf(char ch)
        {char aa[2]; aa[0]=ch; aa[1]=0; return String(aa);}
String valueOf(char chars[]){ return String(chars);}
String valueOf(char chars[], int startIndex, int numChars);
String valueOf(bool tf)
        {if (tf) return String("true"); else return String("false");}
String valueOf(int num){ return String(num);}
String valueOf(long num){ return String(num);}
String valueOf(float num) {return String(num);}
String valueOf(double num) {return String(num);}
```

```
// See also StringBuffer class in this file given below

// ---- End of Java like String object functions -----

//////////////////////////////////////////////////////
//              List of additonal functions not in java
//////////////////////////////////////////////////////
String ltrim();
void ltrim(bool dummy); // dummy to get different signature
String rtrim();
void rtrim(bool dummy); // dummy to get different signature

void chopall(char ch='\n'); // removes trailing character 'ch'
void chop(); // removes one trailing character

void roundf(float input_val, short precision);
void decompose_float(long *integral, long *fraction);

void roundd(double input_val, short precision);
void decompose_double(long *integral, long *fraction);

void explode(char *separator); // see also token() and overloaded explode()
String *explode(int & strcount, char separator = ' '); // see also token()
void implode(char *glue);
void join(char *glue);
String repeat(char *input, unsigned int multiplier);
String tr(char *from, char *to); // translate characters
String center(int padlength, char padchar = ' ');
String space(int number = 0, char padchar = ' ');
String xrange(char start, char end);
String compress(char *list = " ");
String left(int slength = 0, char padchar = ' ');
String right(int slength = 0, char padchar = ' ');
String overlay(char *newstr, int start = 0, int slength = 0, char padchar = ' ');

String at(char *regx); // matches first match of regx
String before(char *regx); // returns string before regx
String after(char *regx); // returns string after regx
String mid(int startIndex = 0, int length = 0);

bool isNull();
bool isInteger();
bool isInteger(int pos);
bool isNumeric();
bool isNumeric(int pos);
bool isEmpty();  // same as length() == 0
bool isUpperCase();
bool isUpperCase(int pos);
bool isLowerCase();
bool isLowerCase(int pos);
bool isWhiteSpace();
bool isWhiteSpace(int pos);
bool isBlackSpace();
bool isBlackSpace(int pos);
bool isAlpha();
bool isAlpha(int pos);
bool isAlphaNumeric();
bool isAlphaNumeric(int pos);
bool isPunct();
bool isPunct(int pos);
bool isPrintable();
bool isPrintable(int pos);
```

```
bool isHexDigit();
bool isHexDigit(int pos);
bool isCntrl();
bool isCntrl(int pos);
bool isGraph();
bool isGraph(int pos);

void clear();
int toInteger();
long parseLong();

double toDouble();
String token(char separator = ' '); // ref StringTokenizer, explode()
String crypt(char *original, char *salt);
String getline(FILE *infp = stdin); // see also putline()
//String getline(fstream *infp = stdin); // see also putline()

void putline(FILE *outfp = stdout); // see also getline()
//void putline(fstream *outfp = stdout); // see also getline()

void swap(String aa, String bb); // swap aa to bb
String *sort(String aa[]);  // sorts array of strings
String sort(int startIndex = 0, int length = 0);  // sorts characters inside a string
int freq(char ch); // returns the number of distinct, nonoverlapping matches
void Format(const char *fmt, ...);
String replace (int startIndex, int endIndex, String str);

void substring(int startIndex, int endIndex, bool dummy);
void reverse(bool dummy); // dummy to get different signature
String deleteCharAt(int loc, bool dummy);
String deleteStr(int startIndex, int endIndex, bool dummy);
void trim(bool dummy); // dummy to get different signature
String insert(int index, String str2);
String insert(int index, String str2, bool dummy);
String insert(int index, char ch);
String insert(int index, char ch, bool dummy);
String insert(char *newstr, int start = 0, int length = 0, char padchar = ' ');

String dump(); // Dump the string like 'od -c' (octal dump) does

// required by java's StringBuffer
void ensureCapacity(int capacity);
void setLength(int len);
void setCharAt(int where, char ch);

// required by java's Integer class, Long, Double classes
int parseInt(String ss) {return ss.toInteger();}
int parseInt(char *ss)
        {String tmpstr(ss); return tmpstr.toInteger();}
long parseLong(String ss) {return ss.parseLong();}
long parseLong(char *ss)
        {String tmpstr(ss); return tmpstr.parseLong();}
float floatValue() {return (float) toDouble(); }
double doubleValue() {return toDouble(); }

///////////////////////////////////////////////
//              List of duplicate function names
///////////////////////////////////////////////
// char * c_str() // use val()
// bool find();  // Use regionMatches()
// bool search();  // Use regionMatches()
// bool matches(); // Use regionMatches()
```

```
        // int rindex(String str2, int startIndex = 0); Use lastIndexOf()
        // String blanks(int slength);  // Use repeat()
        // String append(String str2); // Use concat() or + operator
        // String prepend(String str2);  // Use + operator. See also append()
        // String split(char separator = ' ');  // Use token()
        bool contains(char *str2, int startIndex = 0); // use indexOf()
        // void empty(); Use is_empty()
        // void vacuum(); Use clear()
        // void erase(); Use clear()
        // void zero(); Use clear()
        // bool is_float(); Use is_numeric();
        // bool is_decimal(); Use is_numeric();
        // bool is_Digit(); Use is_numeric();
        // float float_value(); Use toDouble();
        // float tofloat(); Use toDouble();
        // double double_value(); Use toDouble();
        // double numeric_value(); Use toDouble();
        // int int_value(); Use toInteger()
        // int tonumber(); Use toInteger()
        // String get(); Use substring() or val() but prefer java's substring
        // String getFrom(); Use substring() or val() but prefer java's substring
        // String head(int len); Use substring(0, len)
        // String tail(int len); Use substring(length()-len, length())
        // String cut(); Use deleteCharAt() or deleteStr()
        // String cutFrom(); Use deleteCharAt() or deleteStr()
        // String paste(); Use insert()
        // String fill(); Use replace()
        // char firstChar(); // Use substring(0, 1);
        // char lastChar(); // Use substring(length()-1, length());
        // String findNext(); Use token()

        // begin();  iterator. Use operator [ii]
        // end();  iterator. Use operator [ii]
        // copy();  Use assignment =  operator, String aa = bb;
        // clone();  Use assignment =  operator, String aa = bb;

        // All Operators ...
        String operator+ (const String & rhs);
        friend String operator+ (const String & lhs, const String & rhs);

        String& operator+= (const String & rhs); // using reference will be faster
        String& operator= (const String & rhs); // using reference will be faster
        bool operator== (const String & rhs); // using reference will be faster
        bool operator== (const char *rhs);
        bool operator!= (const String & rhs);
        bool operator!= (const char *rhs);
        char operator [] (unsigned long Index) const;
        char& operator [] (unsigned long Index);
        friend ostream &  operator<< (ostream & Out, const String & str2);
        friend istream &  operator>> (istream & In, String & str2);

        static  list<String>                 explodeH;  // list head

protected:
        char *sval; // Not safe to make sval public
        inline void verifyIndex(unsigned long index) const;
        inline void verifyIndex(unsigned long index, char *aa) const;

        void _str_cat(char bb[]);
        void _str_cat(int bb);
        void _str_cat(unsigned long bb);
        void _str_cat(float bb);
```

```
        private:
                // Note: All the private variables and functions begin
                // with _ (underscore)

                //static String *_global_String; // for use in add operator
                //inline void _free_glob(String **aa);
                void _str_cpy(char bb[]);
                void _str_cpy(int bb); // itoa
                void _str_cpy(unsigned long bb);
                void _str_cpy(float bb); // itof

                bool _equalto(const String & rhs, bool type = false);
                bool _equalto(const char *rhs, bool type = false);
                String *_pString;  // temporary pointer for internal use..
                inline void _allocpString();
                inline void _reverse();
                inline void _deleteCharAt(int loc);
                inline void _deleteStr(int startIndex, int endIndex);
                inline void _trim();
                inline void _ltrim();
                inline void _rtrim();
                inline void _substring(int startIndex, int endIndex);
};

// Imitate Java's StringBuffer object
// This class is provided so that the Java code is
// portable to C++, requiring minimum code changes
// Note: While coding in C++ DO NOT use this class StringBuffer,
// this is provided only for compiling code written in Java
// which is cut/pasted inside C++ code.
class StringBuffer: public String
{
        public:
                StringBuffer();
                StringBuffer(int size);
                StringBuffer(String str);
                ~StringBuffer();

                int capacity() {return strlen(sval);}
                StringBuffer append(String str2)
                        { *this += str2; return *this;} // See also operator +
                StringBuffer append(char *str2)
                        { *this += str2; return *this;} // See also operator +
                StringBuffer append(int bb)
                        { *this += bb; return *this;} // See also operator +
                StringBuffer append(unsigned long bb)
                        { *this += bb; return *this;} // See also operator +
                StringBuffer append(float bb)
                        { *this += bb; return *this;} // See also operator +
                StringBuffer append(double bb)
                        { *this += bb; return *this;} // See also operator +

                StringBuffer insert(int index, String str2)
                        { return String::insert(index, str2, true);}

                StringBuffer insert(int index, char ch)
                        { return String::insert(index, ch, true);}

                StringBuffer reverse()
                        { String::reverse(true); return *this;}
```

```
                // Java's "delete()". Cannot use name delete in C++
                StringBuffer deleteStr(int startIndex, int endIndex)
                        { String::deleteStr(startIndex, endIndex, true); return *this;}
                StringBuffer deleteCharAt(int loc)
                        { String::deleteCharAt(loc, true); return *this;}

                StringBuffer substring(int startIndex, int endIndex = 0)
                        { String::substring(startIndex, endIndex, true); return *this;}
};

static String Integer("0"); // java's Integer.parseInt(String);
static String Long("0"); // java's Long.parseLong(String);

// Imitate java's Float class and Float.floatValue()
// provided to compile java code in C++
class Float: public String
{
        public:
                Float(String str);
                Float valueOf(String str2) {return Float(str2);}
                float floatValue() {return (float) toDouble(); }
};

// Imitate java's Double class and Double.doubleValue()
// provided to compile java code in C++
class Double: public String
{
        public:
                Double(String str);
                Double valueOf(String str2) {return Double(str2);}
                double doubleValue() {return toDouble(); }
};

// Imitate java's StringTokenizer class
// provided to compile java code in C++ and vice−versa
class StringTokenizer: public String
{
        public:
                StringTokenizer(String str);
                StringTokenizer(String str, String delimiters);
                StringTokenizer(String str, String delimiters, bool dflag);
                ~StringTokenizer();

                int     countTokens();
                bool    hasMoreElements();
                bool    hasMoreTokens();
                String  nextElement(); // in java returns type 'Object'
                String  nextToken();
                String  nextToken(String delimiters);
        private:
                int             mvCurrentPosition; // current index on string
                int             mvTotalTokens;
                int             mvRemainingTokens;
                char *  mvpListOfDl; // list of delimiters
                char *  mvpWorkStr; // temp work string
                char *  mvpOrigStr; // original string passed
                bool    mvDlFlag;  // delimiter flag
                inline  void vPrepWorkStr(char *delimiters = NULL);
};

// Imitate java's StringReader class
// provided to compile java code in C++
```

```
class StringReader: public String
{
        public:
                StringReader(String str);
                void close() {}  // close the stream
                void mark(int readAheadLimit);
                bool markSupported() {return true;} // tell whether this stream supports the mark() operation, which it does
                int read();
                int read(char cbuf[], int offset, int length);
                bool ready() {return true;} // tell whether this stream is ready to read
                void reset();
                long skip(long ii);
        private:
                unsigned long   _curpos;
                unsigned long   _mark_pos;
};

// Imitate java's StringWriter class
// provided to compile java code in C++
class StringWriter: public String
{
        public:
                StringWriter();
                StringWriter(int bufferSize);
                void close() {clear();}
                void flush() {clear();}
                StringBuffer getBuffer() {return (StringBuffer) *this;}
                String  toString() {return (String) *this;}
                void write(int);
                void write(String);
                void write(char *str1);
                void write(char str1[], int startIndex, int endIndex);
                void write(String str1, int startIndex, int endIndex);
};

// Global variables are defined in String.cpp

#endif // __STRING_H_
```

6. Renaming the String class

6.1 Case 1: Simple rename

If you do not like the String class name then you can use **"typedef"** to rename the String class.

In all the files where you do include String.h, insert these lines:

```
// If you do not like the class name String, then you can rename using typedef
typedef String StringSomethingElseIwant;

// Your remaing code may be like this ....
int main()
{
        StringSomethingElseIwant aa_renstr;
        aa_renstr = "I renamed the String Class using typedef";
```

```
        .......etc...
}
```

See the example_String.cpp.

6.2 Case 2: Resolve conflict

If there is a conflict with another class−name having the same name, and you want to use both this class and conflicting class then you use this technique − in all the files where you do include String.h, insert these lines:

```
#define  String  String_somethingelse_which_I_want
#include "String.h"
#undef String

#include "ConflictingString.h"  // This also has String class...

// All your code goes here...
main()
{
        String_somethingelse_which_I_want aa;
        String bb; // This string class from conflicting string class

        aa = " some sample string";
        bb = " another string abraka−dabraka";
        .......
}
```

The pre−processor will replace all literals of String to "String_somethingelse_which_I_want" and immdiately undefines String. After undef the conflicting string class header file is included which defines the "String" class.

7. File Class

You would use the File class to manipulate the operating system files. This class is an imitation of Java's File class and will be very useful in C++ programming. Using this File class in C++ you can do if file **exists()** ?, if directory **exists()** ?, file **length()** and other functions.

- C++ File class is at File.h http://www.angelfire.com/nv/aldev/cpphowto/File.h and File.cpp http://www.angelfire.com/nv/aldev/cpphowto/File.cpp

- Java java.io.File class definition http://java.sun.com/j2se/1.3/docs/api/java/io/File.html
- Quick Reference on File Class http://unicornsrest.org/reference/java/qref11/java.io.File.html
- File Class summary http://www.idi.ntnu.no/~database/SIF8020/java−api/java.io.File.html

8. C++ Zap (Delete) function

The **delete** and **new** operators in C++ are much better than the malloc and free functions of C. Consider using new and zap (delete function) instead of malloc and free as much as possible.

To make **delete** operators even more cleaner, make a Zap() inline function. Define a zap() function like this:

```
// Put an assert to check if x is NULL, this is to catch
// program "logic" errors early. Even though delete works
// fine with NULL by using assert you are actually catching
// "bad code" very early

// Defining Zap using templates
template <class T>
inline void zap(T& x) {
     assert(x != NULL);
     delete x;
     x = NULL;
}

template <class T*>
inline void zap(T*& x) {
     assert(x != NULL);
     delete x;
     x = NULL;
}

// For delete array
template <class T>
inline void zaparr(T& x) {
     assert(x != NULL);
     delete [] x;
     x = NULL;
}
```

The zap() function will delete the pointer and set it NULL. This will ensure that even if multiple zap()'s are called on the same deleted pointer then the program will not crash. For example −

```
        zap(pFirstname);
        //zap(pFirstname); // no core dumps.  Because pFirstname is NULL now
        //zap(pFirstname); // no core dumps.  Because pFirstname is NULL now

        zap(pLastname);
        zap(pJobDescription);

        int *iiarray = new int[10];
        zaparr(iiarray);
```

There is nothing magical about this, it just saves repetative code, saves typing time and makes programs more readable. The C++ programmers often forget to reset the deleted pointer to NULL, and this causes annoying problems causing core dumps and crashes. The zap() takes care of this automatically. Do not stick a typecast in the zap() function −− if something errors out on the above zap() function it likely has another error somewhere.

Also my_malloc() , my_realloc() and my_free() should be used instead of malloc(), realloc() and free(), as they are much cleaner and have additional checks. For an example, see the file "String.h" which is using the my_malloc() and my_free() functions.

WARNING : Do not use free() to free memory allocated with 'new' or 'delete' to free memory allocated with malloc. If you do, then results will be unpredictable.

See the zap examples in example_String.cpp.

9. Pointers are problems

Pointers are not required for general purpose programming. In modern languages like Java there is no support for pointers. Pointers make the programs messy and programs using pointers are very hard to read.

Avoid using pointers as much as possible and use references. Pointers are really a great pain. It is possible to write an application without using pointers. *You should pointers only in those cases where references will not work.*

A **reference** is an alias; when you create a reference, you initialize it with the name of another object, the target. From the moment on, the reference acts as an alternative name of the target, and anything you do to the reference is really done to the target.

Syntax of References: Declare a reference by writing the type, followed by the reference operator (&), followed by the reference name. References **MUST** be initialized at the time of creation. For example −

```
        int             weight;
        int     & rweight = weight;

        DOG             aa;
        DOG & rDogRef = aa;
```

Do's of references −

- Do use references to create an alias to an object
- Do initialize all references
- Do use references for high efficiency and performance of program.
- Do use **const** to protect references and pointers whenever possible.

Do not's of references −

- **IMPORTANT:** Don't use references to NULL objects
- Don't confuse the address of operator & with reference operator. The references are used in the declarations section (see Syntax of References above).
- Don't try to reassign a reference
- Don't use pointers if references will work
- Don't return a reference to a local object

* Don't pass by reference if the item referred to may go out of scope

10. Usage of my_malloc and my_free

Try to avoid using malloc and realloc as much as possible and use **new** and **zap(delete)**. But sometimes you may need to use the "C" style memory allocations in "C++". Use the functions **my_malloc()** , **my_realloc()** and **my_free()**. These functions do proper allocations and initialisations and try to prevent memory problems. Also these functions (in DEBUG mode) can keep track of memory allocated and print total memory usage before and after the program is run. This tells you if there are any memory leaks.

The my_malloc and my_realloc is defined as below. It allocates little more memory (SAFE_MEM = 5) and initializes the space and if it cannot allocate it exits the program. The 'call_check(), remove_ptr()' functions are active only when DEBUG_MEM is defined in makefile and are assigned to ((void)0) i.e. NULL for non−debug production release. They enable the total−memory used tracing.

```
void *local_my_malloc(size_t size, char fname[], int lineno)
{
        size_t  tmpii = size + SAFE_MEM;
        void *aa = NULL;
        aa = (void *) malloc(tmpii);
        if (aa == NULL)
                raise_error_exit(MALLOC, VOID_TYPE, fname, lineno);
        memset(aa, 0, tmpii);
        call_check(aa, tmpii, fname, lineno);
        return aa;
}

char *local_my_realloc(char *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpjj = 0;
        if (aa) // aa !=  NULL
                tmpjj = strlen(aa);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (char) * (tmpqq);
        aa = (char *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);

        // do not memset memset(aa, 0, tmpii);
        aa[tmpqq-1] = 0;
        unsigned long kk = tmpjj;
        if (tmpjj > tmpqq)
                kk = tmpqq;
        for ( ; kk < tmpqq; kk++)
                aa[kk] = 0;
        call_check(aa, tmpii, fname, lineno);
        return aa;
}
```

See my_malloc.cpp. and the header file my_malloc.h. for full implementation of the my_malloc program.

An example on usage of my_malloc and my_free as below:

```
char    *aa;
int     *bb;
float   *cc;
aa = (char *) my_malloc(sizeof(char)* 214);
bb = (int *) my_malloc(sizeof(int) * 10);
cc = (float *) my_malloc(sizeof(int) * 20);

aa = my_realloc(aa, sizeof(char) * 34);
bb = my_realloc(bb, sizeof(int) * 14);
cc = my_realloc(cc, sizeof(float) * 10);
```

Note that in my_realloc you do not need to cast the datatype as the variable itself is passed and correct my_realloc is called which returns the proper datatype pointer. The my_realloc has overloaded functions for char*, int* and float*.

11. Debug files

To debug any C++ or C programs include the file debug.h and in your 'Makefile' define DEBUG_STR, DEBUG_PRT, DEBUG_MEM to turn on the traces from the debug.h functions. When you remove the '−DDEBUG_STR' etc.. then the debug function calls are set to ((void)0) i.e. NULL, hence it has no impact on final production release version of project. You can generously use the debug functions in your programs and it will not increase the size of production executable.

See the file debug.cpp for implementation of debug routines. And see the file my_malloc.cpp for sample which uses debug.h and debug functions.

See the sample Makefile .

12. IDE tools for C++

The following IDE tools (Integrated Development Environment) are available for C++:

- KDE Kdevelop
- Blatura site C++ Tools
- Amulet Amulet
- App Dev suite Angoss
- Make replacement Brass
- S/W product metrics CCC
- Project mgmt, edit, compile, debug C−Forge
- Dev environment Code Crusader
- Graphic gdb Code Medic
- Code analysis CodeWizard
- Gen HTML, LaTex for C++ cod Doc C++
- GUI toolkit openGL Ftk
- C++ and Java IDE GLG IDE
- HP IDE HP Eloquence

- IDE C++, Java, Pascal RHIDE
- IDE for C++, Java SNiff
- IDE for C++, Java Wipeout
- X−based dev env XWPE

---

13. C++ Online Textbooks and Docs

- "C++ Annotations" online book main site: http://www.icce.rug.nl/docs/cplusplus better site : http://www.physik.tu−muenchen.de/cip−tutorials/Documents/c++−annotations/cplusplus.html
- "Teach Yourself C++ in 21 days" online textbook http://guides.oernii.sk/c++/index.htm
- "C++ Online" Textbook http://www.bruceeckel.com/DownloadSites
- C++ Open books: http://www.softpanorama.org/Bookshelf/cpp.shtml and click on Open Books.
- "Who's Afraid of C++?" online textbook http://www.steveheller.com/whos
- "Introduction to Object Oriented Programming" an ebook http://www.gnacademy.org/uu−gna/text/cc/Tutorial/tutorial.html

- Porting C++ to Java http://www.ibm.com/java/education/portingc
- C/C++ Journals http://www.math.utah.edu/pub/tex/bib/toc/cccuj.html
- Yahoo C++ category site http://dir.yahoo.com/Computers_and_Internet/Programming_Languages/C_and_C__/C__
- C Library Reference Guide http://www.acm.uiuc.edu/webmonkeys/book/c_guide/index.html
- Online textbooks C++/Java http://www.informit.com/free_library

Java books which will be useful for C++ programmers:

- Great Web reference site http://www.webreference.com/index2.html
- Many java books http://freewarejava.com/books/index.shtml
- Intro to Java V3.0 http://math.hws.edu/javanotes mirror http://matrix.mvhs.fuhsd.org/~deruiter/javanotes3/contents.html
- Web Library: http://www.itlibrary.com
- Intermediate Level Java book: http://www.bruceeckel.com
- Thinking in Java : http://www.sce.carleton.ca/netmanage/java/thinking/toc.html
- John Hopkins Univ − Java resources http://www.apl.jhu.edu/~hall/java
- online java tutorial http://chortle.ccsu.ctstateu.edu/cs151/cs151java.html
- Practical guide for Java http://java.sun.com/docs/books/tutorial/index.html
- Java http://journals.ecs.soton.ac.uk/java/tutorial/TOC.html

---

14. C++ Coding Standards

Coding standard is very essential for readability and maitainence of programs. And it also greatly inproves the productivity of the programmer. The GNU C++ compiler **must enforce** coding discipline. The following is suggested − inside class definition:

- All public variables must begin with **m** like **mFooVar**. The **m** stands for *member*.
- All protected variables must begin with **mt**, like **mtFooVar** and methods with t, like **tFooNum()**. The **t** stands for *protected*.
- All private variables must begin with **mv**, like **mvFooVar** and methods with v, like **vFooLone()**. The

v stands for *private*.

- All public, protected and private variables must begin with uppercase after **m** like F in **mFooVar**.
- All pointer variables must be prefixed with p, like
    - ♦ Public variables **mpFooVar** and methods like FooNum()
    - ♦ Protected variables **mtpFooVar** and methods with t like tFooNum()
    - ♦ Private variables **mvpFooVar** and methods with v like vFooNum()

The compiler should generate error if the code does not follow above standard. The C++ compiler can provide a flag option to bypass strict coding standard to compile old source code, and for all new code being developed will follow the uniform world−wide coding standard.

In the sample code given below **t** stands for **protected**, **v** stands for **private**, **m** stands for **member−variable** and **p** stands for **pointer**.

```
class SomeFunMuncho
{
        public:
                int     mTempZimboniMacho; // Only temporary variables should be public as per OOP
                float   *mpTempArrayNumbers;
                int     HandleError();
                float   getBonyBox();  // Public accessor as per OOP design
                float   setBonyBox();  // Public accessor as per OOP design
        protected:
                float   mtBonyBox;
                int     *mtpBonyHands;
                char    *tHandsFull();
                int     tGetNumbers();
        private:
                float   mvJustDoIt;
                char    mvFirstName[30];
                int     *mvpTotalValue;
                char    *vSubmitBars();
                int     vGetNumbers();
};
```

When your program grows by millions of lines of code, then you will greatly appreciate the naming convention as above. The readability of code improves, because just by looking at the variable name like **mvFirstName** you can tell that it is member of a class and is a private variable.

Visit the C++ Coding Standards URLs

- C++ FAQ Lite − Coding standards http://new−brunswick.net/workshop/c++/faq/coding−standards.html
- Rice university coding standard http://www.cs.rice.edu/~dwallach/CPlusPlusStyle.html
- Identifiers to avoid in C++ Programs http://oakroadsystems.com/tech/cppredef.htm
- Coding standards from Possibility http://www.possibility.com/Cpp/CppCodingStandard.html and mirror site
- Coding standards for Java and C++ from Ambysoft http://www.ambysoft.com/javaCodingStandards.html
- Rules and recommendations http://www.cs.umd.edu/users/cml/cstyle/
- Indent and annotate http://www.cs.umd.edu/users/cml/cstyle/indhill−annot.html
- Elemental rules http://www.cs.umd.edu/users/cml/cstyle/Ellemtel−rules.html
- C++ style doc http://www.cs.umd.edu/users/cml/cstyle/Wildfire−C++Style.html

- C++ Coding Standards by Brett Scolcum http://www.skypoint.com/~slocum/prog/cppstds.html
- Logikos C++ Coding Standards http://www.logikos.com/standards/cpp_std.html
- NRad C++ coding standards http://cadswes.colorado.edu/~billo/standards/nrad
- BEJUG C++ coding standards http://www.meurrens.org/ip−Links/java/joodcs/ToddHoff.html

  See also

- For rapid navigation with ctags Vim color text editor
- To improve productivity see C++ Beautifier HOWTO

---

15. C++ Online Docs

There are **MORE THAN ONE MILLION** online articles/textbooks/reference guides on C++ language. That is because C++ is used extensively for a very long period of time. You can find them using the Internet search engines like Yahoo, Lycos, Excite etc..

Visit the following C++ sites :−

- C++ STL basic string class http://www.sgi.com/Technology/STL/basic_string.html
- See the section STL References

- C++ Crash−proof site http://www.troubleshooters.com/codecorn/crashprf.htm
- C++ Memory site http://www.troubleshooters.com/codecorn/memleak.htm

- GNU Main site http://www.gnu.org and gnu C++ site at http://www.gnu.org/software/gcc/gcc.html
- GNU C++ Library − socs http://www.socs.uts.edu.au/doc/gnuinfo/libg++/libg++_18.html
- GNU C++ Library − gsi http://www.gsi.de/doc/gnu/libg++_toc.html
- GNU C++ Library − techno http://www.techno.spb.ru/~xbatob/FAQ/GNU/libg++_toc.html
- GNU C++ Library − utah http://www.math.utah.edu/docs/info/libg++_toc.html
- AS University C++ Standard String class http://www.eas.asu.edu/~cse200/outline

- Java JString for C++ http://www.mike95.com/c_plusplus/classes/JString/JString_cpp.asp
- C++ Language Reference http://www.msoe.edu/~tritt/cpplang.html
- C++ Program examples and samples http://www.msoe.edu/~tritt/cpp/examples.html
- Neil's C++ stuff http://www.cyclone7.com/cpp

Internet has vast amounts of documentation on C++. Visit the search engines like Yahoo, Lycos, Infoseek, Excite. Type in the keywords **'C++ tutorials' 'C++ references' 'C++ books'** . You can narrow down the search criteria by clicking on *Advanced* search and select *search by exact phrase*

- http://www.yahoo.com
- http://www.lycos.com
- http://www.infoseek.com
- http://www.excite.com
- http://www.mamma.com

15.1 C++ Tutorials

There are many on−line tutorials available on internet. Type 'C++ tutorials' in the search engine.

- C++ Tutorial http://www.xploiter.com/programming/c/index.shtml
- C++ Tutorial IISc, India
  http://www.csa.iisc.ernet.in/Documentation/Tutorials/StyleGuides/c++−style.html
- C++ Tutorial Brown Univ http://wilma.cs.brown.edu/courses/cs032/resources/C++tutorial.html
- C++ Tutorial http://home.msuiit.edu.ph/~ddd/tutorials/cpp/cpplist.htm
- C++ Tutorial IOstreams http://osiris.sunderland.ac.uk/~cs0pdu/pub/com365/Sched3/iocpp.html

15.2 Useful links

Bird's eye view of C++ URLs (about 153 url links)
http://www.enteract.com/~bradapp/links/cplusplus−links.html

This URL: http://www.snippets.org portable C code contains over 360 files.

15.3 C++ Quick−Reference

Type 'C++ Reference' in the search engine.

- C++ quick ref http://www.cs.jcu.edu.au/~david/C++SYNTAX.html
- C++ Standard Library Quick Reference
  http://www.halpernwightsoftware.com/stdlib−scratch/quickref.html
- C++ STL from halper http://www.halpernwightsoftware.com/stdlib−scratch/quickref.html

15.4 C++ Usenet Newsgroups

- C++ newsgroups : comp.lang.c++.announce
- C++ newsgroups : comp.lang.c++.*
- C++ newsgroups : http://marshall−cline.home.att.net/cpp−faq−lite

---

16. Memory Tools

Use the following memory debugging tools

- The **MPatrol** is a very powerful memory debugging tool. It is at http://www.cbmamiga.demon.co.uk/mpatrol and at http://www.rpmfind.net go here and search mpatrol. If you are using linux then you must download the mpatrol*.src.rpm file from the rpmfind.net. To update the mpatrol*.src.rpm to latest version, you can use the old mpatrol.spec file and latest mpatrol*.tar.gz file to rebuild new *.src.rpm.
- On linux contrib cdrom see mem_test*.rpm package and at http://www.rpmfind.net go here and search mem_test.
- On linux cdrom see ElectricFence*.rpm package and at http://www.rpmfind.net go here and search electricfence.
- Purify Tool from Rational Software Corp http://www.rational.com

- Insure++ Tool from Parasoft Corp http://www.parasoft.com
- Linux Tools at http://www.xnet.com/~blatura/linapp6.html#tools
- Search the Internet engines like Yahoo, Lycos, Excite, Mamma.com for keyword "Linux memory debugging tools".

---

17. Related URLs

You MUST use a color editor like 'Vim' (Vi improved) while coding in C++. Color editors greatly increase your productivity. Visit the URL for Vim howto below.

Visit following locators which are related to C, C++ −

- Vim color text editor for C++, C http://metalab.unc.edu/LDP/HOWTO/Vim−HOWTO.html
- C++ Beautifier HOWTO http://metalab.unc.edu/LDP/HOWTO/C−C++Beautifier−HOWTO.html
- Source code control system for C++ programs (CVS HOWTO) http://metalab.unc.edu/LDP/HOWTO/CVS−HOWTO.html
- Linux goodies http://www.aldev.8m.com and mirrors at webjump, angelfire, geocities, virtualave, bizland, theglobe, spree, infoseek, bcity, 50megs

---

18. PIKE (C++ Scripting Language)

The major disadvantage of C++ is that you must recompile and link the object files to create a executable anytime you make a small change.

The scripting language like PIKE eliminates the linking and re−compiling and will really speed up the development process.

As memory (RAM) prices are dropping and CPU speeds are increasing, scripting language like PIKE will **EXPLODE in popularity**. PIKE will become most widely used scripting language as it is object oriented and it's syntax is very identical to that of C++ language.

Programming productivity will increase by **five times** by using the Pike C++ scripting language. And Pike is very useful for 'proof of concept' and developing prototypes rapidly.

The Pike is at http://pike.roxen.com and at http://www.roxen.com.

The Roxen Web server is completely written in Pike, which demonstrates how powerful Pike is. Pike runs much faster than Java for some operations and is quite efficient in using memory resources.

---

19. Templates

- http://babbage.cs.qc.edu/STL_Docs/templates.htm Mirror at: http://www.mike95.com/c_plusplus/tutorial/templates

- This tells about #pragma template : − http://www.dgp.toronto.edu/people/JamesStewart/270/9697f/notes/Nov25−tut.html

- Very GOOD site: http://www.cplusplus.com/doc/tutorial/tut5−1.html http://www.cplusplus.com/doc/tutorial
- For certification of C++: goto http://examware.com and click on "Tutorials" and then C/C++ button
- C++ Open books: http://www.softpanorama.org/Lang/cpp.shtml and click on tutorials
- Templates tutorial : http://www.infosys.tuwien.ac.at/Research/Component/tutorial/prwmain.htm

---

20. STL References

Please visit the following sites for STL −

- Very good intro to iterators http://www.cs.trinity.edu/~joldham/1321/lectures/iterators/
- CMU univ http://www.cs.cmu.edu/afs/andrew/course/15/129/Cpp/10−STL/
- Intro to STL SGI http://www.sgi.com/Technology/STL/stl_introduction.html
- Mumits STL Newbie guide http://www.xraylith.wisc.edu/~khan/software/stl/STL.newbie.html Mumit Khan's informative STL introduction is full of examples http://abel.hive.no/C++/STL/stlnew.html
- ObjectSpace examples: ObjectSpace has contributed over 300 examples to the public domain and these are a very good start for beginners. ftp://butler.hpl.hp.com/stl/examples.zip
- Joseph Y. Laurino's STL page. http://weber.u.washington.edu/~bytewave/bytewave_stl.html
- Musser's STL docs and examples. Very nice. http://www.cs.rpi.edu/~musser/stl.html
- STL Newbie home site. http://www.xraylith.wisc.edu/~khan/software/stl/STL.newbie.html
- Marian Corcoran's STL FAQ. ftp://butler.hpl.hp.com/stl/stl.faq

STL Tutorial:

- The best doc on tutorial − http://www.decompile.com/html/tut.html Mirror: http://mip.ups−tlse.fr/~grundman/stl−tutorial/tutorial.html
- very good − http://www.yrl.co.uk/~phil/stl/stl.htmlx
- C++ Standard Template LibraryAnother great tutorial, by Mark Sebern http://www.msoe.edu/eecs/cese/resources/stl/index.htm
- By Jak Kirman: http://129.187.112.56/Misc/Computer/stl−tutorial/tutorial_9.html Mirrors : http://www.cs.brown.edu/people/jak/proglang/cpp/stltut/tut.html http://saturn.math.uaa.alaska.edu/~afjhj/cs403/stl/tutorial.html
- Technical University Vienna by Johannes Weidl http://dnaugler.cs.semo.edu/tutorials/stl mirror http://www.infosys.tuwien.ac.at/Research/Component/tutorial/prwmain.htm
- Colorful Tutorial http://www.codeproject.com/cpp/stlintroduction.asp

Ready−made Components for use with the STL

- STL components Collected by Boris Fomitche http://www.metabyte.com/~fbp/stl/components.html

Main STL sites −

- C++ STL from SGI http://www.sgi.com/Technology/STL
- C++ STL from RPI univ http://www.cs.rpi.edu/projects/STL/htdocs/stl.html
- Lycos C++ STL site http://dir.lycos.com/Computers/Programming/Languages/C%2B%2B/Class_Libraries/STL

- STL for C++ Programmers http://userwww.econ.hvu.nl/~ammeraal/stlcpp.html
- C++ STL from halper http://www.halpernwightsoftware.com/stdlib−scratch/quickref.html

20.1 Overview of the STL

The STL offers the programmer a number of useful data structures and algorithms. It is made up the following components.

- Containers. There are two types:
    - Sequential. This group comprises the vector, list and dequeue types.
    - Sorted Associative. This group comprises the set, map, multiset and multimap types.

- Iterators. These are pointer like objects that allow the user to step through the contents of a container.

- Generic Algorithms. The STL provides a wide range of efficently implemented standard algorithms (for example find, sort and merge) that work with the container types. (Some of the containers have special purpose implementations of these algorithms as member functions.)

- Function Objects. A function object is an instance of a class that provides a definition of operator(). This means that you can use such an object like a function.

- Adaptors. The STL provides
    - Container adaptors that allow the user to use, say, a vector as the basis of a stack.
    - Function adaptors that allow the user to construct new function objects from existing function objects.

- Allocators. Every STL container class uses an Allocator class to hold information about the memory model the program is using. I shall totally ignore this aspect of the STL.

I will be considering the use of the vector, list, set and map containers. To make use of these containers you have to be able to use iterators so I shall have something to say about STL iterators. Using the set and map containers can mean having to supply a simple function object to the instantiation so I shall also have something to say about function objects. I will only briefly mention the algorithms supplied by the STL. I will not mention adaptors at all.

I have taken liberties with some of the types of function arguments −− for example most of the integer arguments referred to in what follows actually have type size_type which is typedef'ed to an appropriate basic type depending on the allocation model being used. If you want to see the true signatures of the various functions discussed have a look at the Working Paper or the header files.

There are a number of utility classes supplied with the STL. The only one of importance to us is the pair class. This has the following definition:

```
template<class T1, class T2>
class pair {
public:
T1 first;
T2 second;
pair(const T1& a, const T2& b) : first(a), second(b) {}
```

```
};
```

and there is a convenient function make_pair with signature:

```
pair<T1,T2> make_pair(const T1& f, const T2&,s)
```

as well as implementations of operator== and operator < . There is nothing complicated about this template class and you should be able to use it without further guidance. To use it #include the header file pair.h. It crops up in a number of places but particularly when using the set and map classes.

20.2 Header Files

To use the various bits of the STL you have to #include the appropriate header files. These may differ slightly from compiler to compiler but for g++ the ones of interest are:

- *vector.h* for the vector type.
- *list.h* for the list type.
- *set.h* for the set type.
- *map.h* for the map type.
- *algo.h* for access to the generic algorithms.

If you don't want to remember which is which you could just use stl.h which includes all the above (and all the other header files of the STL as well).

20.3 The Container Classes Interface

The container classes have many member functions that have the same names. These functions provide the same (or very similar) services for each of the classes (though, of course, the implementations will be different). The following table lists the functions that we shall consider in more detail. A star opposite a function name indicates that the container type heading the column provides a member function of that name.

Operation	Purpose	Vector	List	Set	Map
==	comparison	*	*	*	*
<	comparison	*	*	*	*
begin	iterator	*	*	*	*
end	iterator	*	*	*	*
size	no. of elements	*	*	*	*
empty	is container empty	*	*	*	*
front	first element	*	*		
back	last element	*	*		
[ ]		*			*

	element access/modification				
insert	insert element(s)	*	*	*	*
push_back	insert new last element	*	*		
push_front	insert new first element		*		
erase	remove element(s)	*	*	*	*
pop_back	remove last element	*	*		
pop_front	remove last element		*		
Container Class Interface					

If the following discussion leaves something unclear (and it will) you can always write a small test program to investigate how some function or feature behaves.

20.4 Vectors

A vector is an array like container that improves on the C++ array types. In particular it is not neccessary to know how big you want the vector to be when you declare it, you can add new elements to the end of a vector using the ***push_back*** function. (In fact the ***insert*** function allows you insert new elements at any position of the vector, but this is a very inefficent operation −− if you need to do this often consider using a list instead).

Constructing Vectors

vector is a class template so that when declaring a vector object you have to state the type of the objects the vector is to contain. For example the following code fragment

```
vector<int> v1;
vector<string> v2;
vector<FiniteAutomaton> v3;
```

declares that v1 is a vector that holds integers, v2 a vector that holds strings and v3 holds objects of type FiniteAutomaton (presumably an user defined class type). These declarations do not say anything about how large the vectors are to be (implementations will use a default starting size) and you can grow them to as large as you require.

You can give an inital size to a vector by using a declaration like

```
vector<char> v4(26);
```

which says that v4 is to be vector of characters that initially has room for 26 characters. There is also a way to initailise a vector's elements. The declaration

```
vector<float> v5(100,1.0);
```

says that *v5* is a vector of 100 floating point numbers each of which has been initialised to 1.0.

Checking Up on Your Vector

Once you have created a vector you can find out the current number of elements it contains by using the *size* function. This function takes no arguments and returns an integer (strictly a value of type *size_type*, but this gets converted to an integer) which says how many elements there are in the vector. What will be printed out by the following small program?

```
<vector-size.cc>=
#include <iostream.h>
#include <vector.h>

void main()
{
  vector<int> v1;
  vector<int> v2(10);
  vector<int> v3(10,7);

  cout << "v1.size() returns " << v1.size() << endl;
  cout << "v2.size() returns " << v2.size() << endl;
  cout << "v3.size() returns " << v3.size() << endl;
}
```

To check on wether your vector is empty or not you can use the empty function. This takes no arguments and returns a boolean value, true if the vector is empty, false if it is not empty. What will be printed out by the following small program (true prints as 1 and false prints as 0)?

```
<vector-empty.cc>=
#include <iostream.h>
#include <vector.h>

void main()
{
  vector<int> v1;
  vector<int> v2(10);
  vector<int> v3(10,7);

  cout << "v1.empty() has value " << v1.empty() << endl;
  cout << "v2.empty() has value " << v2.empty() << endl;
  cout << "v3.empty() has value " << v3.empty() << endl;
}
```

Accessing Elements of a Vector

You can access a vector's elements using operator[]. Thus, if you wanted to print out all the elements in a vector you could use code like

```
vector<int> v;
// ...
for (int i=0; i<v.size(); i++)
     cout << v[i];
```

(which is very similar to what you might write for a builtin array).

You can also use operator[] to set the values of the elements of a vector.

```
vector<int> v;
// ...
for (int i=0; i<v.size(); i++)
      v[i] = 2*i;
```

The function front gives access to the first element of the vector.

```
vector<char> v(10,'a');
// ...
char ch = v.front();
```

You can also change the first element using front.

```
vector<char> v(10,'a');
// ...
v.front() = 'b';
```

The function back works the same as front but for the last element of the vector.

```
vector<char> v(10,'z');
// ...
char last = v.back();
v.back() = 'a';
```

Here is a simple example of the use of [].

```
<vector-access.cc>=
```

```
#include <vector.h>
#include <iostream.h>

void main()
{
  vector<int> v1(5);
  int x;
  cout << "Enter 5 integers (seperated by spaces):" << endl;
  for (int i=0; i<5; i++)
    cin >> v1[i];
  cout << "You entered:" << endl;
  for (int i=0; i<5; i++)
    cout << v1[i] << ' ';
  cout << endl;
}
```

Inserting and Erasing Vector Elements

Along with operator[] as described above there are a number of other ways to change or access the elements in a vector.

- push_back will add a new element to the end of a vector.
- pop_back will remove the last element of a vector.
- insert will insert one or more new elements, at a designated position, in the vector.
- erase will remove one or more elements from a vector between designated positions.

Note that insert and erase are expensive operations on vectors. If you use them a lot then you should consider using the list data structure for which they are more efficient.

```
<vector-mod.cc>=
#include <iostream.h>
#include <vector.h>

void main()
{
  vector<int> v;

  for (int i=0; i<10; i++) v.push_back(i);
  cout << "Vector initialised to:" << endl;
  for (int i=0; i<10; i++) cout << v[i] << ' ' ;
  cout << endl;

  for (int i=0; i<3; i++) v.pop_back();
  cout << "Vector length now: " << v.size() << endl;
  cout << "It contains:" << endl;
  for (int i=0; i<v.size(); i++) cout << v[i] << ' ';
  cout << endl;

  int a1[5];
  for (int i=0; i<5; i++) a1[i] = 100;

  v.insert(& v[3], & a1[0],& a1[3]);
  cout << "Vector now contains:" << endl;
  for (int i=0; i<v.size(); i++) cout << v[i] << ' ';
  cout << endl;
```

```
  v.erase(& v[4],& v[7]);
  cout << "Vector now contains:" << endl;
  for (int i=0; i<v.size(); i++) cout << v[i] << ' ';
  cout << endl;
}
```

In the above a vector v has been declared then initialised using push_back. Then some elements have been trimmed off it's end using pop_back. Next an ordinary integer array has been created and then some of its elements inserted into v using insert. Finally erase has been used to remove elements from v. The functions used above take arguments as follows.

- push_back takes a single argument of the type of the elements held in the vector.
- pop_back takes no arguments. It is a mistake to use pop_back on an empty vector.
- insert has three forms:
    - insert(pos, T& x) which will insert the single element x at position pos in the vector.
    - insert(pos, start, end) which inserts a sequence of elements from some other container at position pos in the vector. The
    - sequence of elements is identified as starting at the start element and continuing to, but not including, the end element.
    - insert(pos, int rep, T& x) inserts rep copies of x at position pos in the vector.

As indicated in the code above the position pos should be the address of the element to insert at, whilst the start and end arguments are likewise also addresses. (The true story is that they are iterators −− see next subsection and following section).

- erase has two forms (pos, start and end have the same types as for the insert function):
    - erase(pos) which will remove the element at position pos in the vector.
    - insert(start,end) which will remove elements starting at postion start upto, but not including, the element at position end.

Vector Iterators

The simple way to step through the elements of a vector v is as we have done above:

```
for (int i=0; i<v.size(); i++) { ... v[i] ... }
```

Another way is to use iterators. An iterator can be thought of as a pointer into the container, incrementing the iterator allows you to step through the container. For container types other than vectors iterators are the only way to step through the container.

For a vector containing elements of type T:

```
vector<T> v;
```

an iterator is declared as follows:

```
vector<T>::iterator i;
```

Such iterators are constructed and returned by the functions begin() and end(). You can compare two iterators (of the same type) using == and !=, increment using ++ and dereference using *. [In fact vector iterators allow more operations on them − see next section for more information].

Here is an illustration of how to use iterators with vectors.

```
<vector-iterator.cc>=
#include <iostream.h>
#include <vector.h>

void main()
{
  vector<int> v(10);
 first is ``less'' than the second

  int j = 1;

  vector<int>::iterator i;

  // Fill the vector v with integers 1 to 10.
  i = v.begin();
  while (i != v.end())
  {
    *i = j;
    j++;
    i++;
  }

  // Square each element of v.
  for (i=v.begin(); i!=v.end(); i++) *i = (*i) * (*i);

  // Print out the vector v.
  cout << "The vector v contains: ";
  for (i=v.begin(); i!=v.end(); i++) cout << *i << ' ';
  cout << endl;

}
```

Note how *i can be used on the left−hand side of an assignment statement so as to update the element pointed at by i, and on the right−hand side to access the current value.

Comparing Vectors

You can compare two vectors using == and <. == will return true only if both vectors have the same number of elements and all elements are equal. The < functions performs a lexicographic comparison of the two vectors. This works by comparing the vectors element by element. Suppose we are comparing v1 and v2 (that is v1 < v2?). Set i=0. If v1[i] < v2[i] then return true, if v1[i] > v2[i] then return false, otherwise increment i (that is move on to the next element). If the end of v1 is reached before v2 return true, otherwise return false. Lexicographic order is also known as dictionary order. Some examples:

```
(1,2,3,4) < (5,6,7,8,9,10) is false.
```

```
(1,2,3) < (1,2,3,4) is true
(1,2,3,4) < (1,2,3) is false
(0,1,2,3) < (1,2,3) is true
```

The following code illustrates the third example above.

```
<vector-comp.cc>=
#include <vector.h>
#include <iostream.h>

void main()
{
  vector<int> v1;
  vector<int> v2;
  for (int i=0; i<4; i++) v1.push_back(i+1);
  for (int i=0; i<3; i++) v2.push_back(i+1);

  cout << "v1: ";
  for (int i=0; i<v1.size(); i++) cout << v1[i] << ' ';
  cout << endl;

  cout << "v2: ";
  for (int i=0; i<v2.size(); i++) cout << v2[i] << ' ';
  cout << endl;

  cout << "v1 < v2 is: " << (v1<v2 ? "true" : "false") << endl;
}
```

The comparison operators <= and >= also work.

20.5 Iterators and the STL

See the section STL References

20.6 Lists

See the section STL References

20.7 Sets

The set container type allows an user to store and retrieve elements directly rather than through an index into the container. The set container acts as a mathematical set in that it holds only distinct elements. However unlike a mathematical set, elements in a set container are held in (an user−supplied) order. In practise this is only a minor restriction on treating a set container as an implementation of the mathematical set abstract data type, and it allows for a much more efficent implementation than an unordered approach.

Constructing Sets

Two template arguments are required to construct a set container −− the type of the objects the set is to contain and a function object that can compare two elements of the given type, that is:

```
set<T, Compare> s;
```

(The declaration *set < T > s* should also be possible −− it would use a default template argument less < T > as the second argument, but many C++ compilers (including g++) cannot as yet cope with default template arguments.)

For simple types **T** we can use the function object *less < T >* (without having to worry about what a ``function object'' is), for example all the following are legal set declarations.

```
set<int, less<int> > s1;
set<double, less<double> > s2;
set<char, less<char> > s3;
set<string, less<string> > s4;
```

(Note that the space between the two final >'s in the template is required − otherwise the compiler will interpret >> as the right shift operator.) In each of these cases the function object makes use of the operator < as defined for the the underlying type (that is *int, double, char* and *string*).

The following code declares a set of integers, then adds some integers to the set using the *insert* method and then prints out the set members by iterating through the set. You will note that the set's contents are printed out in ascending order even though they were added in no particular order.

```
<set-construct1.cc>=
#include <iostream.h>
#include <set.h>

void main()
{
  set<int, less<int> > s;
  set<int, less<int> >::iterator i;

  s.insert(4);
  s.insert(0);
  s.insert(-9);
  s.insert(7);
  s.insert(-2);
  s.insert(4);
  s.insert(2);

  cout << The set contains the elements: ;
  for (i=s.begin(); i!=s.end(); i++) cout << *i << ' ';
  cout << endl;
}
```

Note that 4 is added twice but only turns up once on the list of elements −− which is what one expects of a set.

What are Function Objects?

One of the nifty features of C++ is the ability to overload operators, so that one can have + mean whatever one likes for your newly designed class. One of the operators C++ allows you to overload is the function call operator () and this allows you to create classes whose instances can behave like functions in many ways. These are function objects.

Here is a simple example.

```
<function-object.cc>=
#include <iostream.h>

template<class T>
class square {
public:
  T operator()(T x) { return x*x; }
};
// This can be used with any T for which * is defined.

void main()
{
  // Create some function objects.
  square<double> f1;
  square<int> f2;

  // Use them.
  cout << 5.1^2 =  << f1(5.1) << endl;
  cout << 100^2 =  << f2(100) << endl;

  // The following would result in a compile time error.
  // cout << 100.1^2 =  << f2(100.1) << endl;

}
```

Function objects are used in a number of places in the STL. In particular they are used when declaring sets and maps.

The function object required for these purposes, let's suppose it is called *comp*, must satisfy the following requirements.

1. If comp(x,y) and comp(y,z) are true for objects x, y and z then comp(x,z) is also true.
2. comp(x,x) is false for every object x.

If for any particular objects x and y, both comp(x,y) and comp(y,x) are false then x and y are deemed to be equal.

This, in fact, is just the behaviour of the *strictly−less−than* relation (ie <) on numbers. The function object less < T > used above is defined in terms of a < operator for the type T. It's definition can be thought of as follows.

```
template<class T>
struct less {
```

```
  bool operator()(T x, T y) { return x<y; }
}
```

(The actual definition uses references, has appropriate const annotations and inherits from a template class binary_function.)

This means that if the type T has operator < defined for it then you can use less < T > as the comparator when declaring sets of T. You might still want to use a special purpose comparator if the supplied < operator is not appropriate for your purposes. Here is another example. This defines a simple class with a definition of operator < and a function object that performs a different comparison. Note that the overloaded < and () operators should be given const annotations so that the functions work correctly with the STL.

```
<set-construct2.cc>=
#include <iostream.h>
#include <set.h>

// This class has two data members. The overloaded operator< compares
// such classes on the basis of the member f1.
class myClass {
private:
  int f1;
  char f2;
public:
  myClass(int a, char b) : f1(a), f2(b) {}
  int field1() const { return f1; }
  char field2() const { return f2; }
  bool operator<(myClass y) const
        { return (f1<y.field1()); }
};

// This function object compares objects of type myClass on the basis
// of the data member f2.
class comp_myClass {
public:
  bool operator()(myClass c1, myClass c2) const
         { return (c1.field2() < c2.field2()); }
};

void main()
{
  set<myClass, less<myClass> > s1;
  set<myClass, less<myClass> >::iterator i;
  set<myClass, comp_myClass> s2;
  set<myClass, comp_myClass>::iterator j;

  s1.insert(myClass(1,'a'));
  s2.insert(myClass(1,'a'));
  s1.insert(myClass(1,'b'));
  s2.insert(myClass(1,'b'));
  s1.insert(myClass(2,'a'));
  s2.insert(myClass(2,'a'));

  cout << Set s1 contains: ;
  for (i=s1.begin(); i!=s1.end(); i++)
    { cout << ( << (*i).field1() << ,
                  << (*i).field2() << ) << ' ';
    }
```

```
  cout << endl;

  cout << Set s2 contains: ;
  for (j=s2.begin(); j!=s2.end(); j++)
    { cout << ( << (*j).field1() << ,
                  << (*j).field2() << ) << ' ';
    }
  cout << endl;

}
```

The set s1 contains (1,a) and (2,a) as comparison is on the data member f1, so that (1,a) and (1,b) are deemed the same element. The set s2 contains (1,a) and (1,b) as comparison is on the data member f2, so that (1,a) and (2,a) are deemed the same element.

A Printing Utility

The way we have printed out the sets in the previous examples is a little awkward so the following header file containing a simple overloaded version of *operator<<* has been written. It works fine for small sets with simple element types.

```
<printset.h>=
#ifndef _PRINTSET_H
#define _PRINTSET_H

#include <iostream.h>
#include <set.h>

template<class T, class Comp>
ostream& operator<<(ostream& os, const set<T,Comp>& s)
{
  set<T,Comp>::iterator iter = s.begin();
  int sz = s.size();
  int cnt = 0;

  os << {;
  while (cnt < sz-1)
    {
      os << *iter << ,;
      iter++;
      cnt++;
    }
  if (sz != 0) os << *iter;
  os << };

  return os;
}
#endif
```

The use here of << as an output routine for a set assumes that << has been defined for the set elements, and uses this to print a comma delimited list of the set elements wrapped in curly braces. It will be used without comment in the following examples.

How Many Elements?

You can determine if a set is empty or not by using the empty() method. You can find out how many elements there are in a set by using the size() method. These methods take no arguments, empty() returns true or false and size() returns an integer.

```
<set-size.cc>=
#include <iostream.h>
#include <set.h>
#include printset.h

void main()
{
  set<int, less<int> > s;

  cout << The set s is
       << (s.empty() ? empty. : non-empty.) << endl;
  cout << It has  << s.size() <<  elements. << endl;

  cout << Now adding some elements... << endl;

  s.insert(1);
  s.insert(6);
  s.insert(7);
  s.insert(-7);
  s.insert(5);
  s.insert(2);
  s.insert(1);
  s.insert(6);

  cout << The set s is now
       << (s.empty() ? empty. : non-empty.) << endl;
  cout << It has  << s.size() <<  elements. << endl;
  cout << s =  << s << endl;
}
```

Checking the Equality of Sets.

Two sets may be checked for equality by using ==. This equality test works by testing in order the corresponding elements of each set for equality using T::operator==.

```
<set-equality.cc>=
#include <iostream.h>
#include <set.h>
#include printset.h

void main()
{
  set<int, less<int> > s1, s2 ,s3;

  for (int i=0; i<10; i++)
  {
    s1.insert(i);
    s2.insert(2*i);
    s3.insert(i);
```

```
  }

  cout << s1 =  << s1 << endl;
  cout << s2 =  << s2 << endl;
  cout << s3 =  << s3 << endl;
  cout << s1==s2 is:  << (s1==s2 ? true. : false.) << endl;
  cout << s1==s3 is:  << (s1==s3 ? true. : false.) << endl;
}
```

It is also possible to compare two sets using <. The comparison s1 < s2 is true if the set s1 is lexicographically less than the set s2, otherwise it is false.

Adding and Deleting Elements

The way to add elements to a set is to use the insert method (as we have done above). The way to delete elements from a set is to use the erase method.

For a set holding elements of type T these methods come in following forms:

- **pair < iterator, bool> insert(T& x)**. This is the standard insert function. The return value may be ignored or used to test if the insertion succeeded (that is the element was not already in the set). If the insertion succeeded the boolean component will be true and the iterator will point at the just inserted element. If the element is already present the boolean component will be false and the iterator will point at the element x already present.
- **iterator insert(iterator position, T& x)**. This version of the insert function takes, in addition to the element to insert, an iterator stating where the insert function should begin to search. The returned iterator points at the newly inserted element, (or the already present element).
- **int erase(T& x)**. This version of the erase method takes an element to delete and returns 1 if the element was present (and removes it) or 0 if the element was not present.
- **void erase(iterator position)**. This version takes an iterator pointing at some element in the set and removes that element.
- **void erase(iterator first, iterator last)**. This verion takes two iterators pointing into the set and removes all the elements in the range [first,last] .

The following example illustrates these various forms.

```
<set-add-delete.cc>=
#include <iostream.h>
#include <set.h>
#include printset.h

void main()
{
  set<int, less<int> > s1;

  // Insert elements in the standard fashion.
  s1.insert(1);
  s1.insert(2);
```

```
  s1.insert(−2);

  // Insert elements at particular positions.
  s1.insert(s1.end(), 3);
  s1.insert(s1.begin(), −3);
  s1.insert((s1.begin()++)++, 0);

  cout << s1 =  << s1 << endl;

  // Check to see if an insertion has been successful.
  pair<set<int, less<int> >::iterator,bool> x = s1.insert(4);
  cout << Insertion of 4  << (x.second ? worked. : failed.)
       <<  endl;
  x = s1.insert(0);
  cout << Insertion of 0  << (x.second ? worked. : failed.)
       <<  endl;

  // The iterator returned by insert can be used as the position
  // component of the second form of insert.
  cout << Inserting 10, 8 and 7. << endl;
  s1.insert(10);
  x=s1.insert(7);
  s1.insert(x.first, 8);

  cout << s1 =  << s1 << endl;

  // Attempt to remove some elements.
  cout << Removal of 0  << (s1.erase(0) ? worked. : failed.)
       << endl;
  cout << Removal of 5  << (s1.erase(5) ? worked. : failed.)
       << endl;

  // Locate an element, then remove it. (See below for find.)
  cout << Searching for 7. << endl;
  set<int,less<int> >::iterator e = s1.find(7);
  cout << Removing 7. << endl;
  s1.erase(e);

  cout << s1 =  << s1 << endl;

  // Finally erase everything from the set.
  cout << Removing all elements from s1. << endl;
  s1.erase(s1.begin(), s1.end());
  cout << s1 =  << s1 << endl;
  cout << s1 is now  << (s1.empty() ? empty. : non−empty.)
       << endl;
}
```

Finding Elements

We mention two member functions that can be used to test if an element is present in a set or not.

- **iterator find(T& x)**. This searches for the element x in the set. If x is found it returns an iterator pointing at x otherwise it returns end().
- **int count(T& x)**. This returns 1 if it finds x in the set and 0 otherwise. (The count function for multisets returns the number of copies of the element in the set which may be more than 1. Hence, I guess, the name of the function.)

The use of find has been illustrated above. We could use count to write a simple template based set membership function. (This should also provide a version that takes a reference to the argument x.)

```
<setmember.h>=
#ifndef _SETMEMBER_H
#define _SETMEMBER_H
#include <set.h>

template<class T, class Comp>
bool member(T x, set<T,Comp>& s)
{
 return (s.count(x)==1 ? true : false);
}
#endif

Which might be used as follows.

<set−membership.cc>=
#include <iostream.h>
#include <set.h>
#include printset.h
#include setmember.h

void main()
{
  set<int, less<int> > s;
  for (int i= 0; i<10; i++) s.insert(i);
  cout << s =  << s << endl;
  cout << 1 is  << (member(1,s) ?  : not) <<  a member of s
       <<  endl;
  cout << 10 is  << (member(10,s) ?  : not) <<  a member of s
       <<  endl;
}
```

Set Theoretic Operations

The STL supplies as generic algorithms the set operations includes, union, intersection, difference and symmetric diffference. To gain access to these functions you need to include algo.h. (In what follows iter stands for an appropriate iterator).

- bool includes(iter f1,iter l1,iter f2,iter l2).

  This checks to see if the set represented by the range [f2,l2] is included in the set [f1,l1]. It returns true if it is and false otherwise. So to check to see if one set is included in another you would use

  includes(s1.begin(), s1.end(), s2.begin(), s2.end())

  The includes function checks the truth of 3#3 (that is of 4#4). This function assumes that the sets are ordered using the comparison operator <. If some other comparison operator has been used this needs to be passed to includes as an extra (function object) argument after the other arguments.
- iter set_union(iter f1,iter l1,iter f2,iter l2,iter result).

> This forms the union of the sets represented by the ranges [f1,l1] and [f2,l2]. The argument result is an output iterator that points at the start of the set that is going to hold the union. The return value of the function is an output iterator that points at the end of the new set.

The fact that the result argument is an output iterator means that you cannot use set_union in the following, natural, fashion:

```
      set<int, less<int> > s1, s2, s3;
      // Add some elements to s1 and s2 ...
      // Then form their union. (This does not work!)
      set_union(s1.begin(), s1.end(),
                s2.begin(), s2.end(),
                s3.begin());
```

The reason is that begin() (also end()) when used with sets (or maps) returns a (constant) input iterator. This type of iterator allows you to access elements of the set for reading but not writing. (And this is a Good Thing since if you could assign to a dereferenced iterator (as in (*i)= ...) then you could destroy the underlying order of the set.)

The solution is to use an insert iterator based on the set type. This, basically, converts an assignment (*i)=value (which is illegal) into a (legal) insertion s.insert(i,value) (where s is the set object that the iterator i is pointing into). It is used as follows:

```
      // Typedef for convenience.
      typedef set<int, less<int> > intSet;
      intSet s1, s2, s3;
      // Add some elements to s1 and s2 ...
      // Then form their union.
      set_union(s1.begin(), s1.end(),
                s2.begin(), s2.end(),
                insert_iterator<intSet>(s3,s3.begin()) );
```

Here is an example illustrating all these operations.

```
<set-theory.cc>=
#include <iostream.h>
#include <set.h>
#include <algo.h>
#include <iterator.h>
#include printset.h

void main()
{
  typedef set<int, less<int> > intSet;

  intSet s1, s2, s3, s4;

  for (int i=0; i<10; i++)
  { s1.insert(i);
    s2.insert(i+4);
  }
```

```
  for (int i=0; i<5; i++) s3.insert(i);

  cout << s1 =  << s1 << endl;
  cout << s2 =  << s2 << endl;
  cout << s3 =  << s3 << endl;

  // Is s1 a subset of s2?
  bool test = includes(s2.begin(),s2.end(),s1.begin(),s1.end());
  cout << s1 subset of s2 is  << (test ? true. : false.) << endl;

  // Is s3 a subset of s1?
  test = includes(s1.begin(),s1.end(),s3.begin(),s3.end());
  cout << s3 subset of s1 is  << (test ? true. : false.) << endl;

  // Form the union of s1 and s2.
  set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
                 insert_iterator<intSet>(s4,s4.begin()) );
  cout << s1 union s2 =  << s4 << endl;

  // Erase s4 and form intersection of s1 and s2. (If we don't erase
  // s4 then we will get the previous contents of s4 as well).
  s4.erase(s4.begin(),s4.end());
  set_intersection(s1.begin(), s1.end(), s2.begin(), s2.end(),
                 insert_iterator<intSet>(s4,s4.begin()) );
  cout << s1 intersection s2 =  << s4 << endl;

  // Now set difference.
  s4.erase(s4.begin(),s4.end());
  set_difference(s1.begin(), s1.end(), s2.begin(), s2.end(),
                 insert_iterator<intSet>(s4,s4.begin()) );
  cout << s1 minus s2 =  << s4 << endl;

  // Set difference is not symmetric.
  s4.erase(s4.begin(),s4.end());
  set_difference(s2.begin(), s2.end(), s1.begin(), s1.end(),
                 insert_iterator<intSet>(s4,s4.begin()) );
  cout << s2 minus s1 =  << s4 << endl;

  // Finally symmetric difference.
  s4.erase(s4.begin(),s4.end());
  set_symmetric_difference(s1.begin(), s1.end(), s2.begin(), s2.end(),
                 insert_iterator<intSet>(s4,s4.begin()) );
  cout << s1 symmetric_difference  s2 =  << s4 << endl;

  // Which is symmetric!
  s4.erase(s4.begin(),s4.end());
  set_symmetric_difference(s2.begin(), s2.end(), s1.begin(), s1.end(),
                 insert_iterator<intSet>(s4,s4.begin()) );
  cout << s2 symmetric_difference  s1 =  << s4 << endl;
}
```

20.8 Maps

See the section STL References

20.9 STL Algorithms

See the section

---

21. Threads in C++

- IBM pthread User Guide, Thread concepts, API reference http://www.as400.ibm.com/developer/threads/uguide/document.htm and mirror site is at IBM main site
- QpThread Library for C++ provides object oriented framework in C++ for threads and Unix signals on top of system level threads (currently POSIX Threads) http://lin.fsid.cvut.cz/~kra/index.html
- ThreadJack supports Java−like multi−thread programming model with platform independent C++ class library http://www.esm.co.jp/divisions/open−sys/ThreadJack/index−e.html and here is the download−site
- APE is the "APE Portable Environment" and class libraries for writing portable threaded servers in C++, under UNIX (pthread) and Win32 API's. APE provides portable class abstraction for threads, sockets, file handling, and synchronization objects. The goal of APE is to make writing threaded servers in C++ both practical and convient, even for small and simple projects, and hence simplicity and low runtime overhead are design goals http://www.voxilla.org/projects/projape.html
- Portabale Thread Lib http://www.media.osaka−cu.ac.jp/~k−abe/PTL
- Thread−Recyling in C++ http://www.sigs.de/html/kuhlmann.html

21.1 Threads Tutorial

- You can download all the tutorials in one file from aldev−site
- Threads tutorial is at http://www.math.arizona.edu/swig/pthreads/threads.html
- HERT tutorial at http://www.hert.org/docs/tutorials, go here to search for "Threads".
- Intro to threads at linuxjournal
- North Arizona Univ NAU
- Posix threads Acctcom multi−thread

21.2 Designing a Thread Class in C++

This section is written by Ryan Teixeira and the document is located here .

Introduction

Multi threaded programming is becomming ever more popular. This section presents a design for a C++ class that will encapsulate the threading mechanism. Certain aspects of thread programming, like mutexes and semaphores are not discussed here. Also, operating system calls to manipulate threads are shown in a generic form.

Brief Introduction To Threads

To understand threads one must think of several programs running at once. Imagine further that all these programs have access to the same set of global variables and function calls. Each of these programs would represent a thread of execution and is thus called a thread. The important differentiation is that each thread does not have to wait for any other thread to proceed. All the threads proceed simultaneously. To use a metaphor, they are like runners in a race, no runner waits for another runner. They all proceed at their own rate.

Why use threads you might ask. Well threads can often improve the performance of an application and they do not incur significant overhead to implement. They effectively give good bang for a buck. Imagine an image server program that must service requests for images. The program gets a request for an image from another program. It must then retieve the image from a database and send it to the program that requested it. If the server were implemented in a single threaded approach, only one program could request at a time. When it was busy retrieving an image and sending it to a requestor, it could not service other requests. Of course one could still implement such a system without using threads. It would be a challenge though. Using threads, one can very naturally design a system to handle multiple requests. A simple approach would be to create a thread for each request received. The main thread would create this thread upon receipt of a request. The thread would then be responsible for the conversation with the client program from that point on. After retrieving the image, the thread would terminate itself. This would provide a smooth system that would continue to service requests even though it was busy serviceing other requests at the same time.

Basic Approach

The create a thread, you must specify a function that will become the entry point for the thread. At the operating system level, this is a normal function. We have to do a few tricks to wrap a C++ class around it because the entry function cannot be a normal member function of a class. However, it can be a static member function of a class. This is what we will use as the entry point. There is a gotcha here though. Static member functions do not have access to the this pointer of a C++ object. They can only access static data. Fortunately, there is way to do it. Thread entry point functions take a void * as a parameter so that the caller can typecast any data and pass in to the thread. We will use this to pass this to the static function. The static function will then typecast the void * and use it to call a non static member function.

The Implementation

It should be mentioned that we are going to discuss a thread class with limited functionality. It is possible to do more with threads than this class will allow.

```
class Thread
{
   public:
      Thread();
```

```
      int Start(void * arg);
   protected:
      int Run(void * arg);
      static void * EntryPoint(void*);
      virtual void Setup();
      virtual void Execute(void*);
      void * Arg() const {return Arg_;}
      void Arg(void* a){Arg_ = a;}
   private:
      THREADID ThreadId_;
      void * Arg_;

};

Thread::Thread() {}

int Thread::Start(void * arg)
{
   Arg(arg); // store user data
   int code = thread_create(Thread::EntryPoint, this, & ThreadId_);
   return code;
}

int Thread::Run(void * arg)
{
   Setup();
   Execute( arg );
}

/*static */
void * Thread::EntryPoint(void * pthis)
{
   Thread * pt = (Thread*)pthis;
   pthis->Run( Arg() );
}

virtual void Thread::Setup()
{
        // Do any setup here
}

virtual void Thread::Execute(void* arg)
{
        // Your code goes here
}
```

It is important to understand that we are wrapping a C++ object around a thread. Each object will provide an interface to a single thread. The thread and the object are not the same. The object can exist without a thread. In this implementation, the thread does not actually exist until the Start function is called.

Notice that we store the user argument in the class. This is necessary because we need a place to store it temporarily until the thread is started. The operating system thread call allows us to pass an argument but we have used it to pass the this pointer. So we store the real user argument in the class itself and when the execute function is called it can get access to the argument.

Thread(); This is the constructor.

int Start(void * arg); This function provides the means to create the thread and start it going. The argument arg provides a way for user data to be passed into the thread. Start() creates the thread by calling the operating system thread creation function.

int Run(void * arg); This is a protected function that should never be tampered with.

static void * EntryPoint(void * pthis); This function serves as the entry point to the thread. It simply casts pthis to Thread * and

virtual void Setup(); This function is called after the thread has been created but before Execute() is called. If you override this function, remember to call the parent class Execute().

virtual void Execute(void *); You must override this function to provide your own functionality.

Using The Thread Class

To use the thread class, you derive a new class. you override the Execute() function where you provide your own functionality. You may override the Setup() function to do any start up duties before Execute is called. If you override Setup(), remember to call the parent class Setup().

Conclusion

This section presented an implementation of a thread class written in C++. Of course it is a simple approach but it provides a sound foundation upon which to build a more robust design.

If you have comments or suggestions, email to

---

22. C++ Utilities

Visit the following sites for C++ Utilities

- C++ Binary File I/O http://www.angelfire.com/nv/aldev/cpphowto/cpp_BinaryFileIO.html and at mirror site
- Portability Guide http://www.angelfire.com/nv/aldev/cpphowto/cpp_PortabilityGuide.html and at mirror site
- Snippets collections of C++ routines http://www.angelfire.com/nv/aldev/cpphowto/cpp_Snippets.html and at mirror site and at snippets site
- escape ISB for C++ − Provides information on how to develop and program distributed, object−based applications in C++ for Windows and Unix using the Netscape Internet Service Broker http://docs.iplanet.com/docs/manuals/enterprise/cpluspg/contents.htm
- Common c++ http://www.voxilla.org/projects/projape.html
- Large List of free C++ libs http://www.thefreecountry.com/developercity/freelib.html

* C++ Tools http://development.freeservers.com
* C++ Tools CUJ http://www.cuj.com/code
* C++libs Univ of vaasa http://garbo.uwasa.fi/pc/c−lang.html

---

23. Other Formats of this Document

This document is published in 12 different formats namely − DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages, single HTML file and SGML.

* You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from − ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/other−formats/ and http://www.linuxdoc.org/docs.html#howto
* Plain text format is in: ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO and http://www.linuxdoc.org/docs.html#howto
* Single HTML file format is in: http://www.linuxdoc.org/docs.html#howto
* Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO and http://www.linuxdoc.org/docs.html#howto Any help from you to translate to other languages is welcome.

The document is written using a tool called "SGML−Tools" which can be got from − http://www.sgmltools.org Compiling the source you will get the following commands like

* sgml2html C++Programming−HOWTO.sgml (to generate html file)
* sgml2rtf C++Programming−HOWTO.sgml (to generate RTF file)
* sgml2latex C++Programming−HOWTO.sgml (to generate latex file)

LaTeX documents may be converted into PDF files simply by producing a Postscript output using **sgml2latex** (and dvips) and running the output through the Acrobat **distill** (http://www.adobe.com) command as follows:

```
bash$ man sgml2latex
bash$ sgml2latex filename.sgml
bash$ man dvips
bash$ dvips -o filename.ps filename.dvi
bash$ distill filename.ps
bash$ man ghostscript
bash$ man ps2pdf
bash$ ps2pdf input.ps output.pdf
bash$ acroread output.pdf &
```

Or you can use Ghostscript command **ps2pdf**. ps2pdf is a work−alike for nearly all the functionality of Adobe's Acrobat Distiller product: it converts PostScript files to Portable Document Format (PDF) files. **ps2pdf** is implemented as a very small command script (batch file) that invokes Ghostscript, selecting a special "output device" called **pdfwrite**. In order to use ps2pdf, the pdfwrite device must be included in the makefile when Ghostscript was compiled; see the documentation on building Ghostscript for details.

This howto document is located at −

- http://sunsite.unc.edu/LDP/HOWTO/C++Programming−HOWTO.html

Also you can find this document at the following mirrors sites −

- http://www.caldera.com/LDP/HOWTO/C++Programming−HOWTO.html
- http://www.WGS.com/LDP/HOWTO/C++Programming−HOWTO.html
- http://www.cc.gatech.edu/linux/LDP/HOWTO/C++Programming−HOWTO.html
- http://www.redhat.com/linux−info/ldp/HOWTO/C++Programming−HOWTO.html
- Other mirror sites near you (network−address−wise) can be found at http://sunsite.unc.edu/LDP/hmirrors.html select a site and go to directory /LDP/HOWTO/C++Programming−HOWTO.html

In order to view the document in dvi format, use the xdvi program. The xdvi program is located in tetex−xdvi*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Publishing | TeX menu buttons. To read dvi document give the command −

```
xdvi −geometry 80x90 howto.dvi
man xdvi
```

And resize the window with mouse. To navigate use Arrow keys, Page Up, Page Down keys, also you can use 'f', 'd', 'u', 'c', 'l', 'r', 'p', 'n' letter keys to move up, down, center, next page, previous page etc. To turn off expert menu press 'x'.

You can read postscript file using the program 'gv' (ghostview) or 'ghostscript'. The ghostscript program is in ghostscript*.rpm package and gv program is in gv*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Graphics menu buttons. The gv program is much more user friendly than ghostscript. Also ghostscript and gv are available on other platforms like OS/2, Windows 95 and NT, you view this document even on those platforms.

- Get ghostscript for Windows 95, OS/2, and for all OSes from http://www.cs.wisc.edu/~ghost

To read postscript document give the command −

```
gv howto.ps
ghostscript howto.ps
```

You can read HTML format document using Netscape Navigator, Microsoft Internet explorer, Redhat Baron Web browser or any of the 10 other web browsers.

You can read the latex, LyX output using LyX a X−Windows front end to latex.

24. Copyright

25. Appendix A String Program Files

You can **download all programs as a single tar.gz** file from Download String and give the following command to unpack

```
bash$ man tar
bash$ tar ztvf C++Programming-HOWTO.tar.gz
This will list the table of contents

bash$ tar zxvf C++Programming-HOWTO.tar.gz
This will extract the files
```

- Read the header file first and then see the example cpp program
    - String.h http://www.angelfire.com/nv/aldev/cpphowto/String.h
    - string_multi.h http://www.angelfire.com/nv/aldev/cpphowto/string_multi.h
    - example_String.cpp http://www.angelfire.com/nv/aldev/cpphowto/example_String.cpp
- File manipulation class, only length() function is implemented..
    - File.h http://www.angelfire.com/nv/aldev/cpphowto/File.h
    - File.cpp http://www.angelfire.com/nv/aldev/cpphowto/File.cpp
- The zap() implemented here ..
    - my_malloc.h http://www.angelfire.com/nv/aldev/cpphowto/my_malloc.h
    - my_malloc.cpp http://www.angelfire.com/nv/aldev/cpphowto/my_malloc.cpp
- Implementation of string class...
    - String.cpp http://www.angelfire.com/nv/aldev/cpphowto/String.cpp
    - StringTokenizer.cpp http://www.angelfire.com/nv/aldev/cpphowto/StringTokenizer.cpp
- Debug facilities ..
    - debug.h http://www.angelfire.com/nv/aldev/cpphowto/debug.h
    - debug.cpp http://www.angelfire.com/nv/aldev/cpphowto/debug.cpp
    - Makefile http://www.angelfire.com/nv/aldev/cpphowto/Makefile
- Sample java file for testing the functionalities of String class ..
    - string.java http://www.angelfire.com/nv/aldev/cpphowto/string.java