C++ Programming HOW−TO

Table of Contents

Table of Contents

C++ Programming HOW−TO

Al Dev (Alavoor Vasudevan) alavoor@yahoo.com

v22.0, 05 Nov 2000

This document discusses methods to avoid memory problems in C++ and also will help you to program properly in C++ language. The information in this document 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

C++ is the most popular 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 an byte−code−interpreted language running on top of "virtual machine". Java runs faster with JIT compiler but is still slower than C++. And optimized C++ program is about **3 to 4 times faster** than Java using the JIT (Just−In−Time) compiler!! The memory management in Java is automated, so that 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. A neat feature of Java is that memory allocations are taken care of automatically. This howto will enable "C++" to "compete/imitate" with Java language in memory management.

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

1.1 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, weeks or months to debug memory related problems. Many times memory bugs will be "hiding" in the code for several months and can cause unexpected program crashes!! The memory bugs due to usage of **char *** in C/C++ is costing USA and Japan $2 billion every year in time lost in debugging and downtime of programs. If you use **char *** in C++ then it is a very costly affair especially if your programs have more than 50,000 lines of code.

Hence, the following techniques are proposed to overcome the faults of "C" language.

You should use the Java style String class given in this howto or STDlib string class whereever possible and **limit the usage** of char * to cases where you cannot use the String class. The rule is − give first preference to String class and if String class will not do the job then use the char * in C++.

Another technique is to use "C" language using "char *" and 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"** statement says that everything within the brace−surrounded block − in this case, everything in the header file and comp(), some_c_function() is compiled by a C compiler.

The **'String class'** utilises the constructor and destructor to automate the memory management and also provides many 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 lot of string manipulation functions. The **'string class'** and **'String class'** can limit the usage of **char *** datatype. Also, C++ programmers must be encouraged to use 'new', 'delete' features 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!!

1.2 COOP − C++ Object Oriented Programming−language

"C++" is the super−set of "C" and it is proposed that super−set of "C++" be named **COOP** which is acronym for " **C++ O** bject **O** riented **P** rogramming−language" pronounced as **"koooop"**. The "C" files have *.c as file extenstions, "C++" have *.cpp and it is proposed that "COOP" have *.coo as file extenstions. There is a need for "COOP" because I found some programs which were written in C++ but never used Object oriented techniques!! Also programs written in C++ are very hard to maitain as someone can write "C" like programs with C++!! The following can be the 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.
- Code written in COOP will be easy to maintain and is easily understandable/readable.
- Prevent writing "C" like programming in COOP, something which C++ currently allows.
- Syntax **very** close (as much as possible) to C++.
- Code written in "COOP" will be re−usable (thru components, modules, objects). Supports re−usable software components, thereby facilitating Rapid Application Development.
- No downward compatibility to "C" language.
- Borrow best ideas from Microsoft's C#, Java and Python.
- COOP is simple, robust, OOP, has bare mininum syntax (avoiding confusing, redundant, extra constructs)
- COOP is GNU/GPL and will be open−standard language.

In order implement this "COOP" 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 at http://msdn.microsoft.com/vstudio/nextgen/technology/csharpdownload.asp

COOP will compete with proprietary Microsoft C# and Sun's Java langaugues. Microsoft C# overview is at − http://msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp. Microsoft should have named C# as "COOP" because "coop" is a better name than C# and more easy to say and files can have *.coo as extension!! Also take a look at Connective C++ at http://www.quintessent.com/products/cc++.

1.3 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 greatness 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!!

---

2. String Class Variety

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 − socs http://www.socs.uts.edu.au/doc/gnuinfo/libg++/libg++_18.html mirror site gsi http://www.gsi.de/doc/gnu/libg++_toc.html mirror site techno http://www.techno.spb.ru/~xbatob/FAQ/GNU/libg++_toc.html mirror site utah http://www.math.utah.edu/docs/info/libg++_toc.html
- Qt String class at http://doc.trolltech.com/qstring.html mirror at http://www.cs.berkeley.edu/~dmartin/qt/qstring.html

---

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 site : http://aldev.webjump.com

---

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 a 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.

Refer to Appendix A String.h for details about the String class 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'
```

5. C++ Zap (Delete) command

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

To make **delete** command even more cleaner, make a Zap() command. Define a zap() or delete9() command like this:

```
/*
** Use do while to make it robust and bullet-proof macro.
** Note: But not using do while works for some C++ pre-processors ..
#define zap(x) do {assert(x != NULL); delete(x); x = NULL; } while (0)
#define delete9(x) do {assert(x != NULL); delete(x); x = NULL; } while (0)
```

```
*/
// Put a 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!!
#define zap(x) { assert(x != NULL); delete(x); x = NULL; }
#define delete9(x) { assert(x != NULL); delete(x); x = NULL; }

// For delete array
#define zap_(x) { assert(x != NULL); delete [] (x); x = 0; }
#define delete_(x) { assert(x != NULL); delete [] (x); x = 0; }
```

Make sure that your C++ pre−processor is putting the surrounding brackets for delete. If it does not then use the do − while loop. Use −E option with GNU C++ to generate only the pre−processed file and view the output file to verify. Test using this −

```
bash$ g++ -E example_String.cpp > preprocess.out
bash$ vi preprocess.out
```

The zap() command 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);
        //delete9(pFirstname); // you can use either zap or delete9
```

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() command −− if something errors out on the above zap() command 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!!

6. 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 a 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

7. 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*.

8. 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 .

---

9. 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
- 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++ 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

- Teach Yourself C++ in 21 days online textbook http://guides.oernii.sk/c++/index.htm
- Effective C++ online textbook http://electra.lbl.gov/stuff/EffectiveCPP/EC/E.HTM
- C++ Online Textbook http://www.bruceeckel.com/DownloadSites
- 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

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

9.1 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

9.2 C++ Coding Standards

Visit the C++ Coding Standards URLs

- C++ coding standard http://www.cs.umd.edu/users/cml/cstyle/CppCodingStandard.html
- Coding standards from Possibility http://www.possibility.com/Cpp/CppCodingStandard.html
- Coding standards 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

9.3 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

9.4 C++ Usenet Newsgroups

- C++ newsgroups : comp.lang.c++.announce
- C++ newsgroups : comp.lang.c++.*

10. Memory Tools

Use the following memory debugging tools

- On linux contrib cdrom see mem_test*.rpm package
- On linux cdrom see ElectricFence*.rpm package
- 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".

11. 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 main site http://www.aldev.8m.com
- Linux goodies mirror site http://aldev.webjump.com

12. 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.

12.1 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).

12.2 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                  *
```

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.

12.3 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 a 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.

12.4 Iterators and the STL

This section to be written

12.5 Lists

This section to be written

12.6 Sets

The set container type allows a 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 (a 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;
}
```

12.7 Maps

This section not yet written.

12.8 STL Algorithms

This section not yet written.

13. Other Formats of this Document

This document is published in 11 different formats namely − DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages 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/
- Plain text format is in: ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://sunsite.unc.edu/pub/Linux/docs/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.

---

14. Copyright

---

15. Appendix A example_String.cpp

You can download all programs as a single tar.gz file from Download String . To get this file, in the web−browser, save this file as 'Text' type.

- example_String.cpp http://www.angelfire.com/nv/aldev/cpphowto/example_String.cpp
- String.h http://www.angelfire.com/nv/aldev/cpphowto/String.h
- String.cpp http://www.angelfire.com/nv/aldev/cpphowto/String.cpp
- my_malloc.cpp http://www.angelfire.com/nv/aldev/cpphowto/my_malloc.cpp
- my_malloc.h http://www.angelfire.com/nv/aldev/cpphowto/my_malloc.h
- 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
- StringTokenizer.cpp http://www.angelfire.com/nv/aldev/cpphowto/StringTokenizer.cpp

---