

C++ Programming HOW-TO

Table of Contents

<u>C++ Programming HOW-TO</u>	1
Al Dev (Alavoor Vasudevan) alavoor[AT]yahoo.com	1
1. Introduction	1
2. Recommended C++ Compilers	1
3. String Class Varieties	1
4. Download String	1
5. Usage of String class	1
6. String.h file	2
7. The Standard C++ Library string class	2
8. File Class	2
9. Memory Allocation in C++	2
10. Pointers are problems	2
11. Debugging	2
12. IDE's and editors for C++	2
13. C++ Online Textbooks and Docs	2
14. C++ Coding Conventions	2
15. C++ Scripting Languages	3
16. Templates	3
17. STL References	3
18. Threads in C++	3
19. C++ Utilities	3
20. Other Formats of this Document	3
21. Translations To Other Languages	3
22. Copyright	3
23. Appendix A String Program Files	3
24. Appendix B C++ v/s Java	3
1. Introduction	4
1.1 Program in C++ ? C++ vs. Java/PHP	4
1.2 Which one Ada95, C, C++, Java or PHP?	5
1.3 Problems facing the current C++ compilers	6
2. Recommended C++ Compilers	7
2.1 Compilers for MS Windows 2000/NT/95/98/ME/XP	7
2.2 Compilers for UNIX systems and clones	8
3. String Class Varieties	8
3.1 Multiple Inheritance – Sample Custom String class	8
4. Download String	9
4.1 How Can I trust Al Dev's String Class?	9
5. Usage of String class	10
5.1 Operators	11
5.2 Functions	11
5.3 Renaming the String class	12
Case 1: Simple rename	12
Case 2: Resolve conflict	12
6. String.h file	13
6.1 StringBuffer.h	15
6.2 StringTokenizer.h	16
7. The Standard C++ Library string class	16
7.1 string by example	17

Table of Contents

7.2 Searching a string	18
7.3 A string tokenizer	19
8. File Class	20
9. Memory Allocation in C++	20
9.1 C++ Zap (Delete) function	21
9.2 Usage of <code>my_malloc</code> and <code>my_free</code>	22
9.3 Garbage Collector for C++	23
9.4 Anti-crash With <code>const</code>	24
9.5 Forget C	24
10. Pointers are problems	25
11. Debugging	26
11.1 Debug files	26
12. IDE's and editors for C++	26
12.1 IDE's	26
12.2 Editors	27
12.3 Other resources	27
13. C++ Online Textbooks and Docs	28
13.1 C++ Sites	28
13.2 C++ Tutorials	29
13.3 Useful links	29
13.4 C++ Quick-Reference	29
13.5 C++ Usenet Newsgroups	30
13.6 Java like API	30
14. C++ Coding Conventions	30
15. C++ Scripting Languages	31
15.1 PIKE & PHP (C/C++ Scripting Languages)	32
15.2 SoftIntegration Ch (C/C++ Scripting Language)	32
15.3 PHP (C++ Scripting Language)	32
16. Templates	32
17. STL References	34
17.1 Overview of the STL	34
17.2 Header Files	36
17.3 The Container Classes Interface	36
17.4 Vectors	37
Constructing Vectors	37
Checking Up on Your Vector	37
Accessing Elements of a Vector	38
Inserting and Erasing Vector Elements	40
Vector Iterators	41
Comparing Vectors	42
17.5 Iterators and the STL	43
17.6 Lists	43
17.7 Sets	43
Constructing Sets	43
What are Function Objects?	44
A Printing Utility	47
How Many Elements?	48
Checking the Equality of Sets	48

Table of Contents

Adding and Deleting Elements	49
Finding Elements	50
Set Theoretic Operations	51
17.8 Maps.....	54
17.9 STL Algorithms.....	54
18. Threads in C++.....	54
18.1 Threads Tutorial.....	54
18.2 Designing a Thread Class in C++.....	55
Introduction	55
Brief Introduction To Threads	55
Basic Approach	55
The Implementation	55
Using The Thread Class	57
Conclusion	57
19. C++ Utilities.....	57
19.1 Memory Tools.....	58
20. Other Formats of this Document.....	58
20.1 Acrobat PDF format	59
20.2 Convert linuxdoc to Docbook format	60
20.3 Convert to MS WinHelp format	60
20.4 Reading various formats	60
21. Translations To Other Languages.....	61
22. Copyright.....	61
23. Appendix A String Program Files.....	61
24. Appendix B C++ v/s Java.....	62

C++ Programming HOW-TO

AI Dev (Alavoor Vasudevan) [alavoor\[AT\]yahoo.com](mailto:alavoor[AT]yahoo.com)

v42.8, 03 Aug 2002

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 supplies C++ compilers (it means almost all the operating systems on this planet).

1. Introduction

- [1.1 Program in C++ ? C++ vs. Java/PHP](#)
- [1.2 Which one Ada95, C, C++, Java or PHP?](#)
- [1.3 Problems facing the current C++ compilers](#)

2. Recommended C++ Compilers

- [2.1 Compilers for MS Windows 2000/NT/95/98/ME/XP](#)
- [2.2 Compilers for UNIX systems and clones](#)

3. String Class Varieties

- [3.1 Multiple Inheritance – Sample Custom String class](#)

4. Download String

- [4.1 How Can I trust AI Dev's String Class?](#)

5. Usage of String class

- [5.1 Operators](#)
- [5.2 Functions](#)
- [5.3 Renaming the String class](#)

6. String.h file

- [6.1 StringBuffer.h](#)
- [6.2 StringTokenizer.h](#)

7. The Standard C++ Library string class

- [7.1 string by example](#)
- [7.2 Searching a string](#)
- [7.3 A string tokenizer](#)

8. File Class

9. Memory Allocation in C++

- [9.1 C++ Zap \(Delete\) function](#)
- [9.2 Usage of my_malloc and my_free](#)
- [9.3 Garbage Collector for C++](#)
- [9.4 Anti-crash With const](#)
- [9.5 Forget C](#)

10. Pointers are problems

11. Debugging

- [11.1 Debug files](#)

12. IDE's and editors for C++

- [12.1 IDE's](#)
- [12.2 Editors](#)
- [12.3 Other resources](#)

13. C++ Online Textbooks and Docs

- [13.1 C++ Sites](#)
- [13.2 C++ Tutorials](#)
- [13.3 Useful links](#)
- [13.4 C++ Quick-Reference](#)
- [13.5 C++ Usenet Newsgroups](#)
- [13.6 Java like API](#)

14. C++ Coding Conventions

15. C++ Scripting Languages

- [15.1 PIKE & PHP \(C/C++ Scripting Languages\)](#)
- [15.2 SoftIntegration Ch \(C/C++ Scripting Language\)](#)
- [15.3 PHP \(C++ Scripting Language\)](#)

16. Templates

17. STL References

- [17.1 Overview of the STL](#)
- [17.2 Header Files](#)
- [17.3 The Container Classes Interface](#)
- [17.4 Vectors](#)
- [17.5 Iterators and the STL](#)
- [17.6 Lists](#)
- [17.7 Sets](#)
- [17.8 Maps](#)
- [17.9 STL Algorithms](#)

18. Threads in C++

- [18.1 Threads Tutorial](#)
- [18.2 Designing a Thread Class in C++](#)

19. C++ Utilities

- [19.1 Memory Tools](#)

20. Other Formats of this Document

- [20.1 Acrobat PDF format](#)
- [20.2 Convert linuxdoc to Docbook format](#)
- [20.3 Convert to MS WinHelp format](#)
- [20.4 Reading various formats](#)

21. Translations To Other Languages

22. Copyright

23. Appendix A String Program Files

24. Appendix B C++ v/s Java

1. [Introduction](#)

(The latest version of this document is at <http://www.milkywaygalaxy.freesevers.com>. You may want to check there for changes).

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 Java-like String class, string tokenizer, memory functions and many other functions, which can be used in general C++ applications. C++ and Java is often used concurrently in many software projects. Programmers jump back and forth between C++ and Java will find this Java-like classes very helpful. Various examples are given which demonstrate the usage of this library and the Standard C++ Library.

This document is not a textbook on C++, and there are already several excellent on-line text books on the internet. Since C++ is being used for a long time there are very large number of C++ documents/articles/tutorials on Internet. If you are new to C++ and you never programmed in C++, then it is strongly suggested that you first either read an online C++ textbook given in chapter [C++ Online Textbooks](#) or you buy a C++ book from online bookstores such as [Amazon](#) or [barnes](#).

As someone said – *Leave the C/C++ programming to system engineers who write operating system, device drivers and fast response real-time programming, you should use Java/PHP-scripting as speed of the computers in year 2005 will be several billion times faster than computers of year 2002!!* Hardware is getting cheaper and faster.

1.1 Program in C++ ? C++ vs. Java/PHP

C++ is one of the most powerful languages and will be used for a long time in the future in spite of emergence of Java or PHP-scripting. Programs which need real-time ultra fast response use C/C++. C++ runs **extremely fast** and is in fact **10 to 20 times FASTER than** Java. Java is the "offspring" of C++. The only complaint against Java is – *"Java is GOD DAMN SLOW"*. Java byte-code is slower when running in a VM than the equivalent natively compiled code. Java runs faster with JIT (Just-In-Time) compiler, but it is still slower than C++. And optimized C/C++ program is about **3 to 4 times faster** than Java compiled to native code with JIT compiler or ahead-of-time compiler!! Then, why do people use Java? Because it is pure object oriented and 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 the 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.

When should you use C++ and when you should use Java/PHP?

Bottom line is, you use C++:

- If you are developing a program where speed and performance is very important.
- If the user base of your application or program is very large. Since C++ involves compile-link-debug cycle it is more time consuming to develop application in C++. You can justify the cost only if the user base of your program is large enough. Linking a large number of object files to create an executable takes time. (You can use archives, libraries or shared libraries to reduce linking time).

- If you have lot of experience programming in C++.

Use Java/PHP:

- If speed and performance is not important (relative to C/C++).
- Lower cost of development – There is no compile-link cycle, Java/PHP development is faster than C++.
- Need rapid development.
- You want no code maintenance nightmare. Maintaining C++ is more difficult than Java or PHP-scripting.
- Java and PHP-scripting is the future, hardware speed will be zooming with the introduction of molecular, atomic and sub-atomic scale computers. Future computers will be several trillion times faster than today's computer. Runtime performance of Java or PHP-script becomes less important as speed of hardware zooms in future. The computers you are using today (as of year 2002) are extremely slow and crawling and are not fast enough.

NOTE: There are lot of improvements in Java compilers (JIT and ahead-of-time). Java programs can be compiled with GNU GCJ <http://gcc.gnu.org/java>, GCJ is a portable, optimizing, ahead-of-time compiler for the Java programming language. It can compile – Java source code directly to native machine code, Java source code to Java bytecode (class files), and Java bytecode to native machine code.

The Java native code compiler "gnu GCJ" is very rapidly maturing and in near future everybody will be programming in Java instead of C++. Special optimizers in gnu GCJ compiler can make Java programs as fast as C++ programs. The gnu GCJ compiler is 2–3 years away in becoming the de facto compiler on all platforms (initially on Linux and then on MS Windows).

GCJ resources:

- Main site GNU GCJ <http://gcc.gnu.org/java>,
- Redhat rpm GNU GCJ <http://www.redhat.com/apps/download>. Go here and under the section "Find latest RPMs" search by keyword 'gcc-java' and 'libgcj'.
- Redhat GCJ Installation instructions <http://www.redhat.com/devnet/articles/gcj.pdf>. In this document see also the "Usage of gnu GCJ", it gives some examples on howto compile the Java programs using gcj (you MUST read the examples otherwise you will be confused).

1.2 Which one Ada95, C, C++, Java or PHP?

Language choice is very difficult. There are too many parameters – people, people skills, cost, tools, politics (even national politics) and influence of business-people/commercial companies. The best language based on technical merits does not get selected simply due to political decisions!

See the language comparison chart of David Wheeler at [Ada comparison chart](#). Ada got 93%, Java 72%, C++ 68% and C got 53%. C++ and Java are closer in points (only 4% difference). Development costs of Ada is half of C++ as per [Stephen F. Zeigler](#). Ada95 is available at –

- Ada home <http://www.gnuada.org>.
- Google [Ada index](#)

The C++ compiler is lot more complex than a C compiler and C++ programs may run bit slower than C programs. The C compiler is very mature and seasoned.

On some system, you can use compiler options, to optimize the code generated.

Nowadays, C is primarily used for low level systems programming to develop operating systems, device drivers and applications which must perform fast.

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

If you want to bypass the edit-compile-debug-compile cycle of C++ then see scripting languages like PHP which can be used for web development and for general purpose programming. Scripting languages like PHP and PERL enable rapid application development. PHP has some features of object-oriented programming. PHP is at <http://www.linuxdoc.org/HOWTO/PHP-HOWTO.html>.

1.3 Problems facing the current C++ compilers

Since C++ is a super-set of C, it has all the "bad" features of C. Manual allocation and deallocation of memory is tedious and error prone (see [Garbage Collector for C++](#)).

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", "memory corruption", "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 increase 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 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. Give preference in the following order –

1. Use references instead of pointers.
2. Java style String class (given in this HOWTO) or the string class from the Standard C++ Library.
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 <some_c_header.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**' utilizes the constructor and destructor features to automate memory management and provides access to functions like *ltrim*, *substring*, etc..

See also related [string class](#) in your C++ compiler. The **string class** is part of the Standard C++ Library 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'.

Both string classes do everything that **char *** or **char []** do. One of the added benefits is that you do not have to worry about the memory problems and memory allocation at all.

2. Recommended C++ Compilers

The current C++ standard adopted by ISO and ANSI was first finalized in 1997, this means that not all compilers are up to pace yet, and not supporting all features – it is extremely important that you get a standard compliant C++ compiler.

2.1 Compilers for MS Windows 2000/NT/95/98/ME/XP

Since MS Windows is quite popular for C++ development, the String class library given in this document works well and runs very well on all the versions of MS Windows i.e. MS Win XP/2000/NT/95/98/ME. The C++ compilers for MS Windows are:

- Redhat Cygwin GNU gcc compiler <http://www.cygwin.com> and mirror [Redhat cygwin](#).
- GNU BloodShed at <http://www.bloodshed.net/devcpp.html>
- Borland C++ compiler <http://www.borland.com/bcppbuilder/freecompiler>
- Microsoft Visual C++ compiler <http://msdn.microsoft.com/visualc>
- MSDOS C++ compiler <http://www.delorie.com/djgpp>
- <http://www.digitalmars.com>

The String class in this document is tested with all the above compilers. It works fine with MS Visual C++ compiler v6.0, Borland C++ v5.2, Borland C++ compiler v5.5.1 and Bloodshed compiler.

2.2 Compilers for UNIX systems and clones

In a GNU world, you will always be best off with GCC (GNU Compiler Collection), GCC is distributed with most Linux distributions, FreeBSD and most other UNIX clones. The GCC homepage is located at <http://gcc.gnu.org>. The latest version of GCC (3.0) is one of the most standards compliant compilers out there.

3. String Class Varieties

The string class is the one of the most vital objects in programming, and string manipulations are most extensively used. There is a lot of varieties of string classes. Of course, you can build your own string class by simply inheriting from these string classes –

- String class given in this document [Appendix A String.h](#)
- Standard C++ Library string class (ANSI/ISO string class at <http://www.msoe.edu/eecs/cese/resources/stl/string.htm> and http://www.sgi.com/tech/stl/basic_string.html)
- The external library Qt, has a 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 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.)

3.1 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 the standard C++ library string class and the String class presented in Appendix A.

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_ALDEV_
#define __STRING_MULTI_H_ALDEV_

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

#ifdef NOT_MSWINDOWS
#else
```

C++ Programming HOW-TO

```
using namespace std; // required for MS Visual C++ compiler Version 6.0
#endif

// 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 constructors
// 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 operat
    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_ALDEV_
```

4. [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 to <http://www.milkywaygalaxy.freeservers.com> and click on "Source code C++ Programming howto" ([Milkyway Galaxy site](#))
- Mirror sites are at – [angelfire](#), [geocities](#), [virtualave](#), [50megs](#), [theglobe](#), [NBCi](#), [Terrashare](#), [Fortunecity](#), [Freewebsites](#), [Tripod](#), [Spree](#), [Escalix](#), [Httpcity](#), [Freeservers](#).

4.1 How Can I trust AI Dev's String Class?

You may have question of mis-trust of the String class software. To build confidence, there is a scientific method to verify the functionality of AI Dev's String class. In modern days, computer scientists use the CPU power instead of human brain power to verify and validate the software. Human brain is too slow and hence it is better to use the computer's power to test and validate software.

The program [example_String.cpp](#) go here and click on 'Source code for C++'. (and also given in [Appendix A](#)) has regression test module which you can use to run the regression tests several millions of times automatically. After running the regression tests on the String class you can certify that the String class program is a **ROCK SOLID** and a **BULLET-PROOF** program.

I tested the String class with **repeat cycle = 50000** and it ran and completed the program without crash. While it is running I did not notice any memory leak. On Linux, I used /usr/bin/gtop, UNIX top command, KDEStart->System->KDE System Guard and KDEStart->System->Process management to monitor the cpu and memory usage.

I recommend that you start the regression test with **repeat cycle** equal to 10 million or greater. The greater the repeat cycle number the greater will be your confidence!! Start the test and go to lunch (or go drink gharam chai - "chai peeke auvo") and come back to see the results!!

5. [Usage of String class](#)

Take notice, this String class is not the same as the string class implemented in the Standard C++ Library. This special String class is a "home-made" String class, made to help Java programmers convert to C++. When you are more comfortable with C++, you should use the real string class provided in The Standard C++ Library.

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, similar to creating a baby human.";

// 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);
}
```

5.1 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;
```

5.2 Functions

The functions provided by String class have the **same name** as that of Java language's String class. The function names and the behaviour is **exactly** the 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 minimum 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 + " Can create infinite number of universes!! " + 234.878;
cout << "The value of aa is : " << aa.val() << endl;
```

```
// The output aa will be '34 Can create 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.

5.3 Renaming the String class

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 remaining 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](#) click on 'Source code for C++'.

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...
int 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 immediately undefines `String`. After `undef` the conflicting string class header file is included which defines the `"String"` class.

6. [String.h file](#)

C++ and Java are often used concurrently in many software projects. Programmers who jump back and forth between C++ and Java will find this string class very helpful.

In C++ (or any object oriented language), you just read the "class data-structure" (i.e. interface) to begin using that class. 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 '**interface**', which hides the implementation details.)

Given below is the sample **String.h** file and see also [Appendix A String.h](#)

The **String.h** has more than **200 string manipulation** functions but below only few functions are shown as sample.

```
// I compiled and tested this string class on Linux (Redhat 7.1) and
// MS Windows Borland C++ version 5.2 (win32). This should also work
// using MS Visual C++ compiler
class String
{
    public:
        String();
        virtual ~String();

        // Functions below imitate Java language's String object
        unsigned long length();
        char charAt(int where);
        void getChars(int sourceStart, int sourceEnd,
                     char target[], int targetStart);
        char* toCharArray();
        char* getBytes();

        bool equals(String str2);
        bool equals(char *str2);
        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);
```

C++ Programming HOW-TO

```
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();

String concat(String str2);
String concat(char *str2);
String concat(int bb);
String concat(unsigned long bb);
String concat(float bb);
String concat(double bb);

String reverse();
String deleteCharAt(int loc);
String deleteStr(int startIndex, int endIndex);

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 additional functions not in Java
////////////////////////////////////

String ltrim();
String rtrim();

////////////////////////////////////
// More than 200 string manipulation functions are provided (see the
// "Download String" section) but only few functions are shown here.
////////////////////////////////////
```

C++ Programming HOW-TO

```
void ensureCapacity(int capacity);
void setLength(int len);
void setCharAt(int where, char ch);

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(); }

// All Operators ...
String operator+ (const String rhs);
String operator+= (const String rhs);
String operator= (const String rhs);
bool operator== (const String rhs);
bool operator!= (const String rhs);
char operator [] (unsigned long Index) const;
ostream operator << (ostream Out, const String str2);
istream operator >> (istream In, String str2);
bool String::operator< (const char rhs);

////////////////////////////////////
// More than 200 string manipulation functions are provided (see the
// "Download String" section) but only few functions are shown here.
////////////////////////////////////
};
```

6.1 StringBuffer.h

C++ and Java are often used concurrently in many software projects. Programmers jump back and forth between C++ and Java will find this stringbuffer class very helpful.

```
//
// Author : Al Dev Email: alavoor[AT]yahoo.com
//

// 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();
        StringBuffer(char *aa);
        StringBuffer(int size);
        StringBuffer(String str);

        int capacity();
        StringBuffer append(String str2);
        // See also operator +
```

C++ Programming HOW-TO

```
        // { *this += str2; return *this; } // This is causing core dumps...

    StringBuffer append(char *str2);
    StringBuffer append(int bb);
    StringBuffer append(unsigned long bb) ;
    StringBuffer append(float bb) ;
    StringBuffer append(double bb) ;

    StringBuffer insert(int index, String str2);
    StringBuffer insert(int index, char ch);

    StringBuffer reverse();

    // Java's "delete()". Cannot use name delete in C++
    StringBuffer deleteStr(int startIndex, int endIndex);
    StringBuffer deleteCharAt(int loc);

    StringBuffer substring(int startIndex, int endIndex = 0);
    void assign(char *str);
};
```

6.2 StringTokenizer.h

C++ and Java is often used concurrently in many software projects. Programmers jump back and forth between C++ and Java will find this stringtokenizer class very helpful.

```
//
// Author : Al Dev Email: alavoor[AT]yahoo.com
//
// 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 delimAsToken);
        ~StringTokenizer();

        int    countTokens();
        bool   hasMoreElements();
        bool   hasMoreTokens();
        String nextElement(); // in Java returns type 'Object'
        String nextToken();
        String nextToken(String delimiters);
};
```

[7. The Standard C++ Library string class](#)

While the previously mentioned String class (note the uppercase S), is a good thing for people coming from Java, then you should take notice of the "real" string class provided by The Standard C++ Library.

The string class was made to overcome one of the greatest pitfalls in C; character arrays. While character arrays are extremely fast, they have many bad sides. Character arrays are the cause of many bugs, and parsing character arrays is very time consuming.

The string class brings a good interface for parsing and handling strings, and it's even STL compatible, so it can be used with all the general STL algorithms. Actually you could say that a string is a **vector<char>**. A container of chars, or an advanced array of chars.

Useful string references can be found at the following sites:

- SGI STL basic_string reference: http://www.sgi.com/tech/stl/basic_string.html.

7.1 string by example

Creating a string is easy:

```
#include <string>
#include <iostream>

using namespace std;

int main()
{
    string str("Hello World!"); // Or string str = "Hello World!";
    cout << str << endl;
}
```

This code will create a string called **str**, and put **Hello World!** into it. It is then being outputted to standard output by using cout.

(Note that I will skip the headers and the namespace from now on.)

Taking a substring of a string is also easy:

```
string str("Hello Universe!");
string start = str.substr(0, 5);
string end = str.substr(5);
```

This will put the first 6 characters into the string **start**, and the rest into **end**.

To get the size or length of a string, you would simply do this:

```
string str("How long is this string?");
cout << "Length of string is: " << str.size() << endl;
```

You can also use **length()** which works exactly the same.

7.2 Searching a string

Searching a string is much easier than using plain character arrays, the string class provides efficient member functions to search through the string. All member functions return `string::size_type`.

Member function	Purpose
<code>find()</code>	find the first position of the specified substring
<code>find_first_of()</code>	equal to <code>find()</code> , but finds the first position of any character specified
<code>find_last_of()</code>	equal to <code>findfirstof()</code> , but finds the last position of any character specified
<code>find_first_not_of()</code>	equal to <code>findfirstof()</code> , but returns the position of the first character not of those specified
<code>find_last_not_of()</code>	equal to <code>findlastof()</code> , but returns the last position of any characters not specified
<code>rfind()</code>	equal to <code>find()</code> , but searches backwards
string search member functions	

A very common thing to do, is to search a string for contents. This can be done by using **`find()`**

```
string str("Hello, can you find Ben?");
string::size_type position = str.find("Ben");
cout << "First occurrence of Ben was found at: " << position << endl;
```

This code makes a case sensitive search for **'Ben'** in the string, and puts the start position in the variable **'position'** of type `string::size_type`. Note that the return value is not an int, but a `string::size_type` which is a special implementation defined integral value.

The member function **`find_first_of()`** needs a practical introduction, consider this:

```
string s = "C++ is an impressive language.";
string::size_type pos = s.find_first_of(" .");

while (pos != string::npos) {
    cout << "Found space or dot at: " << pos << endl;
    pos = s.find_first_of(" .", pos + 1);
}
```

By using **`find_first_of()`**, we can search the string for **any** character of the first argument, here we decide to search for a space or a dot.

Try compiling the program and check the output.

7.3 A string tokenizer

A very common operation with strings, is to tokenize it with a delimiter of your own choice. This way you can easily split the string up in smaller pieces, without fiddling with the `find()` methods too much. In C, you could use `strtok()` for character arrays, but no equal function exists for strings. This means you have to make your own. Here is a couple of suggestions, use what suits your best.

The advanced tokenizer:

```
void Tokenize(const string& str,
             vector<string>& tokens,
             const string& delimiters = " ")
{
    // Skip delimiters at beginning.
    string::size_type lastPos = str.find_first_not_of(delimiters, 0);
    // Find first "non-delimiter".
    string::size_type pos      = str.find_first_of(delimiters, lastPos);

    while (string::npos != pos || string::npos != lastPos)
    {
        // Found a token, add it to the vector.
        tokens.push_back(str.substr(lastPos, pos - lastPos));
        // Skip delimiters. Note the "not_of"
        lastPos = str.find_first_not_of(delimiters, pos);
        // Find next "non-delimiter"
        pos = str.find_first_of(delimiters, lastPos);
    }
}
```

The tokenizer can be used in this way:

```
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

int main()
{
    vector<string> tokens;

    string str("Split me up! Word1 Word2 Word3.");

    Tokenize(str, tokens);

    copy(tokens.begin(), tokens.end(), ostream_iterator<string>(cout, " "));
}
```

The above code will use the `Tokenize` function, take the first argument `str` and split it up. And because we didn't supply a third parameter to the function, it will use the default delimiter " ", that is – a whitespace. All elements will be inserted into the vector `tokens` we created.

In the end we `copy()` the whole vector to standard out, just to see the contents of the vector on the screen.

Another approach is to let stringstream do the work. streams in C++ have the special ability, that they read until a whitespace, meaning the following code works if you only want to split on spaces:

```
#include <vector>
#include <string>
#include <sstream>

using namespace std;

int main()
{
    string str("Split me by whitespaces");
    string buf; // Have a buffer string
    stringstream ss(str); // Insert the string into a stream

    vector<string> tokens; // Create vector to hold our words

    while (ss >> buf)
        tokens.push_back(buf);
}
```

And that's it! The stringstream will use the output operator (`>>`) and put a string into **buf** everytime a whitespace is met, **buf** is then used to `push_back()` into the vector. And afterwards our vector **tokens** will contain all the words in **str**.

8. [File Class](#)

C++ and Java are often used concurrently in many software projects. Programmers jump back and forth between C++ and Java will find this File class very helpful.

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.

Note that these classes have some great functionality not supported in the Standard C++ Library, but don't confuse them with `fstreams(iostreams)`, which is the way you should perform many other operations on files.

- C++ File class is at `File.h` and `File.cpp` [File.cpp](#) click on 'Source code of C++'.
 - 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>
-

9. [Memory Allocation in C++](#)

In C, you use `malloc()`, `free()` and variants of `malloc()` to allocate and free memory, but these functions have their pitfalls. Therefore C++ introduced operators for handling memory, these operators are called **new** and **delete**. These operators allocates and frees memory from the heap (or sometimes called the free store) at

runtime.

In C++, you should always use **new** and **delete** unless you're really forced to use `malloc()` and `free()`. But be aware that you cannot mix the two. You cannot `malloc()` memory, and then delete it afterwards, likewise you can't "new" memory, and then free it with `free()`.

9.1 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
// Use zap instead of delete as this will be very clean
template <class T>
inline void zap(T & x)
{
    {assert(x != NULL);}
    delete x;
    x = NULL;
}

// In C++ the reason there are 2 forms of the delete operator is - because
// there is no way for C++ to tell the difference between a pointer to
// an object and a pointer to an array of objects. The delete operator
// relies on the programmer using "[" to tell the two apart.
// Hence, we need to define zaparr function below.
// To delete array of pointers
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. Please see the function `zap_example()` in [example_String.cpp](#) click on 'Source code of C++'.

```
// See zap_example() in example_String.cpp
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 repetitive 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](#) click on 'Source code of C++'.

9.2 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*.

9.3 Garbage Collector for C++

In C/C++ Garbage Collection is not a standard feature and hence allocating and freeing storage explicitly is difficult, complicated and is error-prone. The **Garbage Collection (GC)** is not part of the C++ standard because there are just so many ways how one could implement it; there are many GC techniques, and deciding to use a particular one would not be good for certain programs. Computer scientists had designed many GC algorithms, each one of them catering to a particular problem domain. There is no one single generic GC which will tackle all the problem domains. As a consequence, GC is not part of C++ standard, they just left it out. Still, you always have the choice of many freely available C++ libraries that do the job for you.

Visit these sites:

- [Garbage Collection](#)
- [Memory management](#)
- Java-like library – Hans-J. Boehm's garbage collector at [Hans-Boehm](#) works quite well under Linux, Solaris and Windows.

9.4 Anti-crash With const

You must use **const** at every opportunity. The **const** is your most powerful anti-crash weapon. An additional benefit is that it makes your code self-documenting. For instance, look at this:

```
const char *add2strings(const char *sz1, const char *sz2);
```

Such a declaration guarantee's that no matter what wierd things go on within the function, it can't harm the application programmer's two strings sz1 and sz2. If any memory corruption occurs, it will be to variables within the function's scope. This, of course, greatly reduces side effect bugs.

Furthermore, the declaration of the return pointer as **const** means that the application programmer can't "reach inside" the function to corrupt its scope. For instance, if the return value is a static array of 40 characters, if the return wasn't static the application programmer could do this:

```
char *pchName = add2strings("James", "Bond 007");
strcat(pchName, " jumps out of the plane and parachutes down");
cout << "I just corrupted an internal variable of add2strings. ";
cout << "Will I see this message?\n";
cout << "Will it crash later in the program? Who knows!\n";
```

Fortunately, because add2strings() returns a **const** pointer, you'll get a compiler error on this:

```
char *pchName = add2strings("Big galaxies", "Big stars");
```

Even if you declared pchName as a **const** char *, the moment you modified its contents with strcat() you'd get a compile error. The **const** keyword helps the programmer keep any errors localized, thus greatly reducing the likelihood of side-effect errors.

9.5 Forget C

Forget you ever knew printf(). Use cout <<. Of course, scanf() was flirting with disaster even in the rough and ready C days. Use cin >>, or make your own input routines or classes. What about that wonderful sprintf()? Instead of this:

```
#include <iostream.h>
#include <stdio.h>
/* . . . */
char myaString[100];
sprintf(myaString, "%s %s %c", myaLname, myaFname, *myaMname);
cout << myaString << '\n';
```

Try This:

```

#include <iostream.h>
#include <sstream.h>
/* . . . */
ostringstream ost;
ost << myaLname << ", " << myaFname << " " << *myaMname;
ost.put(0); //null terminate the string cout
cout << ost.str() << '\n';

```

10. [Pointers are problems](#)

Pointers are not required for general purpose programming. In modern languages like Java there is no support for pointers (Java internally uses 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
-

11. [Debugging](#)

Finding the exact source to a bug can be a troublesome process, however there is several techniques used for debugging:

- Printing to standard out – for simple cases, print the values of several variables, see what they contain – and find out where exactly your program crashes
- Using a debugger, a debugger lets you set breakpoints, and make backtraces in your code, while it's running. Most IDEs come with a debugger, for GNU systems there is gdb.
- Use compiler features, on most compilers you can enable more warnings, for example on g++, use `-Wall`

Sites to help debugging:

- Debugging C and C++ in a UNIX environment: <http://www.liacs.nl/~jdassen/onderwijs/stuva/debug/debug.html>
- MPatrol – a useful memory debugging tool: <http://www.cbmamiga.demon.co.uk/mpatrol>
- NJAMD – another useful memory debugging tool: <http://sourceforge.net/projects/njamd/>
- LeakTracer – a simple yet powerful tool to find memory leaks: <http://www.andreasen.org/LeakTracer/>

11.1 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 a sample which uses `debug.h` and debug functions.

See the sample [Makefile](#) .

12. [IDE's and editors for C++](#)

When programming C++, it is a good idea to used to an editor or an IDE. Most programmers have their own favourites, and it's a religious discussion on which is better.

You can choose to use an IDE (Integrated Development Environment), which is an application with embedded editor, compiler, documentation and more. And then there is the stand-alone editors, which some people like better.

12.1 IDE's

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

- The "top rated" Dev-C++ is an full-featured Integrated Development Environment (IDE) for both Win32 and Linux. It uses GCC, Mingw or Cygwin as compiler and library set. It is at <http://www.bloodshed.net/devcpp.html> and at [mirror-site](#)
- KDE KDevelop [Kdevelop](#)
- Blatura site [C++ Tools](#)
- Amulet [Amulet](#)
- App Dev suite [Angoss](#)
- Make replacement [Brass](#)
- S/W product metrics [CCC](#)
- Project management, 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](#)

12.2 Editors

The problem with IDE's is many times their editors have a big lack of functionality. Therefore many people want a powerful editor alone, and then supply compiler next to it.

Of powerful editors, vim and emacs can be mentioned. They are both available for most platforms – and they support syntax highlighting and other things which will make you more efficient.

Other editors include UltraEdit(win32 only) and EditPlus(win32 only).

- Vim online at <http://vim.sourceforge.net/>
- Vim color text editor for C++, C <http://www.linuxdoc.org/LDP/HOWTO/Vim-HOWTO.html>
- Emacs at <http://www.gnu.org/software/emacs/>
- EditPlus for Windows at <http://www.editplus.com/>
- UltraEdit for Windows at <http://www.ultraedit.com/>

12.3 Other resources

- C++ Beautifier HOWTO <http://www.linuxdoc.org/LDP/HOWTO/C-C++Beautifier-HOWTO.html>
 - Source code control system for C++ programs (CVS HOWTO) <http://www.linuxdoc.org/LDP/HOWTO/CVS-HOWTO.html>
 - Linux goodies main site is at <http://24.221.230.253> and secondary site at <http://www.milkywaygalaxy.freesevers.com> Mirror sites are at – [angelfire](#), [geocities](#), [virtualave](#), [Fortunecity](#), [Freewebsites](#), [Tripod](#), [101xs](#), [50megs](#),
-

13. C++ Online Textbooks and 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 Google, Yahoo, Lycos, Excite etc..

- "C++ Annotations" online book main site: [Annotations](#)
- "Teach Yourself C++ in 21 days" online textbook [Teach C++](#)
- C++ Textbook by Bruce Eckel [Thinking in C++](#)
- C++ Open books: [Panorama](#) and click on Open Books.
- "Who's Afraid of C++?" online textbook: [Steveheller](#)
- "Introduction to Object Oriented Programming" an ebook [C++ OOP](#)
- C++ in Hypertext [C++ Hypertext](#)
- Object Oriented Systems [OOP article](#)

- C++ Language Reference from cplusplus.com <http://www.cplusplus.com/ref>
- C++ stdlib Reference for commands like atol, atoi <http://www.cplusplus.com/ref/cstdlib>
- C++ Documentation from cplusplus.com <http://www.cplusplus.com>
- Common C++ Pitfalls to be avoided <http://www.horstmann.com/cpp/pitfalls.html>

- Porting C++ to Java [PortingC](#)
- C/C++ Journals [UtahJournals](#)
- Yahoo C++ category site [CCyahoo](#)
- C Library Reference Guide [c_guide](#)
- Online textbooks C++/Java [FreeLib](#)
- "C++ In Action" by Bartosz Milewski at <http://www.relisoft.com/book/index.htm>
- Amusing examples of how not to write code. "How to write unmaintainable code" at <http://mindprod.com/unmain.html>

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

- Great Web reference site [WebRef](#)
- Many Java books [JBooks](#)
- Introduction to Java V3.0 [JavaNotes](#) mirror [JavaNotes](#)
- Thinking in Java: [Thinking Java](#)
- John Hopkins Univ – Java resources [Hall](#)
- online Java tutorial [Chortle](#)
- Practical guide for Java [SunBooks](#)
- Java [Soton](#)

13.1 C++ Sites

Visit the following C++ sites :-

- C++ STL basic string class documentation is at http://www.sgi.com/tech/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://gcc.gnu.org>
- 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/cplusplus.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 Google, 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.google.com>
- <http://www.yahoo.com>
- <http://www.lycos.com>
- <http://www.infoseek.com>
- <http://www.excite.com>
- <http://www.mamma.com>

13.2 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>
- Cplusplus.com Tutorial <http://www.cplusplus.com/doc/tutorial>
- C++ Tutorial IISc, India
<http://www.csa.iisc.ernet.in/Documentation/Tutorials/StyleGuides/cplusplus-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/cplusplus.htm>
- C++ Tutorial IOstreams <http://osiris.sunderland.ac.uk/~cs0pdu/pub/com365/Sched3/iocpp.html>

13.3 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](http://www.snippets.org) portable C code contains over 360 files.
- Mathtools at <http://www.mathtools.net> is a technical computing portal for all scientific and engineering needs. The portal is free and contains over 20,000 useful links to technical computing programmers, covering C/C++, Java, Excel, MATLAB, Fortran and others.

13.4 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.halpernwrightsoftware.com/stdlib-scratch/quickref.html>
- C++ STL from halper <http://www.halpernwrightsoftware.com/stdlib-scratch/quickref.html>

13.5 C++ Usenet Newsgroups

- C++ newsgroups : comp.lang.c++.announce
- C++ newsgroups : [comp.lang.c++.*](http://comp.lang.c++.)
- C++ newsgroups : <http://marshall-cline.home.att.net/cpp-faq-lite>

13.6 Java like API

Visit the following sites for Java like API for C++

- Java utilities in C++ <http://www.pulsar.org/users/ej/archive/ooop>
- PhD Thesis book Java API in C++ <http://www.pulsar.org/archive/phd/ejphd>
- Java-like library Jakelib at <http://www.jakelib.org>. The version 2 of Jakelib uses Boehm's gc and is a lot more Java-like. Written by [Florian](#).

14. [C++ Coding Conventions](#)

Coding convention is very essential for readability and maintenance of programs. And it also greatly improves the productivity of the programmer. Coding convention is required for good 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()**

Uniform world-wide coding convention for C++ language will help better programming.

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
};
```

C++ Programming HOW-TO

```
        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://www.parashift.com/c++-faq-lite/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/joodes/ToddHoff.html>
- Arctic Labs coding standards <http://www.arcticlabs.com/codingstandards>

See also

- For rapid navigation with ctags [Vim color text editor](#)
 - To improve productivity see [C++ Beautifier HOWTO](#)
-

15. [C++ Scripting Languages](#)

The major disadvantage of C++ is that you must recompile and link the object files to create an executable anytime you make a small change. The compile/link/debug cycles take away a lot of time and is quite unproductive. Since modern CPU's and RAM are becoming extremely fast and cheap, it is sometimes better to spend more money on hardware and use scripting languages for development.

15.1 PIKE & PHP (C/C++ Scripting Languages)

The scripting languages like PHP or 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 languages like PHP or PIKE will **EXPLODE in popularity**. PHP or PIKE will become most widely used scripting language as it is object oriented and it's syntax is very identical to that of C/C++.

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

PHP is exploding in popularity for general purpose programming and for web development. PHP may become the most widely used scripting language in near future. PHP is at <http://www.linuxdoc.org/HOWTO/PHP-HOWTO.html>.

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.

15.2 SoftIntegration Ch (C/C++ Scripting Language)

If you want commercial scripting language, get the 'Ch scripting' product from SoftIntegration corporation at <http://www.softintegration.com>.

The scripting language environment called Ch is a superset of C with high-level extensions, and salient features from C++ and other languages so that users can learn the language once and use it anywhere for almost any programming purposes. This C-compatible scripting language environment is also a middleware serving as crucial software infrastructure for running portable applications in heterogeneous platforms. The portable Ch code can be deployed safely over the internet or intranets to run anywhere ranging from supercomputers, workstations, PCs, Palm Pilots, PDA, to non-traditional computing devices such as CNC machines, robots, TVs, refrigerators, among others.

15.3 PHP (C++ Scripting Language)

PHP is hypertext-preprocessor scripting language and is very rapidly evolving and adopting object oriented features. It has the "class" keyword through which it tries to implement object oriented scripting. May be in near future PHP will mature rapidly to become a robust scripting language for object oriented projects. In future it will tackle both the web applications and general purpose applications. Why have different scripting languages for web and general applications, instead just use PHP for both. PHP is at <http://www.linuxdoc.org/HOWTO/PHP-HOWTO.html>.

16. [Templates](#)

Templates is a feature in C++ which enables generic programming, with templates code-reuse becomes much easier.

Consider this simple example:

```
#include <string>
#include <iostream>

void printstring(const std::string& str) {
    std::cout << str << std::endl;
}

int main()
{
    std::string str("Hello World");
    printstring(str);
}
```

Our `printstring()` takes a `std::string` as its first argument, therefore it can only print strings. Therefore, to print a character array, we would either overload the function or create a function with a new name.

This is bad, since the implementation of the function is now duplicated, maintainability becomes harder.

With templates, we can make this code re-usable, consider this function:

```
template<typename T>
void print(const T& var) {
    std::cout << var << std::endl;
}
```

The compiler will automatically generate the code for whatever type we pass to the `print` function. This is the major advantage of templates. Java doesn't have templates but Java uses inheritance to achieve generic programming. For example in Java:

```
public static void printstring( Object o ) {
    System.out.println( o );
}
You can pass in any object you want.
```

References:

- 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++: go to <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>
-

17. STL References

Please visit the following sites for STL:

- Very good introduction to iterators <http://www.cs.trinity.edu/~joldham/1321/lectures/iterators/>
- Introduction to STL SGI http://www.sgi.com/tech/stl/stl_introduction.html
- Mumits STL Newbie guide (a bit outdated)
http://www.xraylith.wisc.edu/~khan/software/stl/STL_newbie.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
- Marian Corcoran's STL FAQ. <ftp://butler.hpl.hp.com/stl/stl.faq>

STL tutorials:

- Phil Ottewell's STL Tutorial – <http://www.yrl.co.uk/~phil/stl/stl.htmlx>
- Good, but outdated doc – <http://www.decompile.com/html/tut.html> Mirror:
<http://mip.ups-tlse.fr/~grundman/stl-tutorial/tutorial.html>
- The Code Project, introduction to C++/STL/MFC
<http://www.codeproject.com/cpp/stl/introduction.asp>
- C++ Standard Template Library, another great tutorial, by Mark Sebern
<http://www.msoe.edu/eecs/cese/resources/stl/index.htm>
- 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>

Main STL sites:

- C++ STL from SGI <http://www.sgi.com/tech/stl>
- C++ STL from RPI univ <http://www.cs.rpi.edu/projects/STL/htdocs/stl.html>
- C++ STL site [ODP for STL](#) and the [mirrorsite](#)
- STL for C++ Programmers <http://userwww.econ.hvu.nl/~ammeraal/stlcpp.html>
- C++ STL from halper <http://www.halpernwrightsoftware.com/stdlib-scratch/quickref.html>

17.1 Overview of the STL

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

- Containers. There are two types:
 - ◆ Sequential. This group comprises the vector, list and deque 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 efficiently 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 liberty 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 `<utility>`. It crops up in a number of places but particularly when using the set and map classes.

17.2 Header Files

To use the various bits of the STL you have to `#include` the appropriate header files. If your compiler is not standard compliant, this may differ, but a standard compliant compiler (like g++), would have these:

- `<vector>` for the vector type.
- `<list>` for the list type.
- `<set>` for the set type.
- `<map>` for the map type.
- `<algorithm>` for access to the generic algorithms.

Note that headers in the Standard C++ Library are without a `.h` suffix. If you use an old or poor compiler, the above headers might fail, if then, you can try the version with the `.h` suffix, or better yet; get another compiler.

17.3 The Container Classes Interface

The container classes have many member functions that have the same names. These functions provide the same (or very similar) interface 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
<code>==</code>	comparison	*	*	*	*
<code><</code>	comparison	*	*	*	*
<code>begin</code>	iterator	*	*	*	*
<code>end</code>	iterator	*	*	*	*
<code>size</code>	no. of elements	*	*	*	*
<code>empty</code>	is container empty	*	*	*	*
<code>front</code>	first element	*	*		
<code>back</code>	last element	*	*		
<code>[]</code>	element access/modification	*			*
<code>insert</code>	insert element(s)	*	*	*	*
<code>push_back</code>	insert new last element	*	*		
<code>push_front</code>	insert new first element		*		
<code>erase</code>	remove element(s)	*	*	*	*
<code>pop_back</code>	remove last element	*	*		
<code>pop_front</code>	remove first 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.

17.4 Vectors

A vector is an array like container that improves on the C++ array type. In particular it is not necessary 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 to insert new elements at any position of the vector, but this is a very inefficient 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 initial 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 initialise 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.cpp>=
#include <iostream>
#include <vector>

using namespace std;

int 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 whether 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.cpp>=
#include <iostream>
#include <vector>

using namespace std;

int 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 built-in 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.cpp>=
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    vector<int> v1(5);
    int x;
    cout << "Enter 5 integers (separated 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.cpp>=
#include <iostream>
#include <vector>

using namespace std;

int 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.
 - ◆ `erase(start,end)` which will remove elements starting at position `start` up to, 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.cpp>=
#include <iostream>
#include <vector>

using namespace std;

int 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 true.
(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.cpp>=
#include <vector>
#include <iostream>

using namespace std;

int 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.

17.5 Iterators and the STL

See the section [STL References](#)

17.6 Lists

See the section [STL References](#)

17.7 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 practice 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 efficient 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 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.cpp>=
#include <iostream>
#include <set>

using namespace std;

int 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.cpp>=
#include <iostream>

using namespace std;

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

int 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.cpp>=
#include <iostream>
#include <set>

using namespace std;

// 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()); }
};

int 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>
#include <set>

template<class T, class Comp>
std::ostream& operator<<(std::ostream& os, const std::set<T,Comp>& s)
{
    std::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.cpp>=
#include <iostream>
#include <set>
#include "printset.h"

using namespace std;

int 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.cpp>=
#include <iostream>
#include <set>
#include "printset.h"

using namespace std;

int 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 version 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.cpp>=
#include <iostream>
#include <set>
#include "printset.h"

using namespace std;

int 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>

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

Which might be used as follows.

```
<set-membership.cpp>=
#include <iostream>
#include <set>
#include "printset.h"
#include "setmember.h"

using namespace std;

int 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 difference. 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())
```

C++ Programming HOW-TO

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.cpp>=
#include <iostream>
#include <set>
#include <algorithm>
#include <iterator>
#include "printset.h"

using namespace std;
```

```

int 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;
}

```

17.8 Maps

See the section [STL References](#)

17.9 STL Algorithms

See the section [STL References](#)

18. [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 convenient, 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-Recycling in C++ <http://www.sigs.de/html/kuhlmann.html>

18.1 Threads Tutorial

- You can download all the tutorials in one file from <http://www.milkywaygalaxy.freesevers.com> and click on "Source code C++ Programming howto".
- 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".
- Introduction to threads at [linuxjournal](#)
- North Arizona Univ [NAU](#)
- POSIX threads [Acctcom multi-thread](#)

18.2 Designing a Thread Class in C++

This section is written by [Ryan Teixeira](#) and the document is located [here](#) .

Introduction

Multi threaded programming is becoming 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 retrieve 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 servicing other requests at the same time.

Basic Approach

To 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 [Ryan Teixeira](mailto:Ryan.Teixeira)

19. [C++ Utilities](#)

Visit the following sites for C++ Utilities

- Portable C++ utilities from <http://www.boost.org>. The Boost web site provides free peer-reviewed portable C++ source libraries. The emphasis is on libraries which work well with the C++ Standard Library. One goal is to establish "existing practice" and provide reference implementations so that the Boost libraries are suitable for eventual standardization.
- The smart pointer library from http://www.boost.org/libs/smart_ptr/index.htm includes five smart pointer class templates. Smart pointers ease the management of memory dynamically allocated with C++ new expressions. In addition, scoped_ptr can ease the management of memory dynamically allocated in other ways.
- C++ Binary File I/O http://www.angelfire.com/country/aldev0/cpphowto/cpp_BinaryFileIO.html
- Portability Guide http://www.angelfire.com/country/aldev0/cpphowto/cpp_PortabilityGuide.html
- Snippets collections of C++ routines

http://www.angelfire.com/country/aldev0/cpphowto/cpp_Snippets.html 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/cplusplus/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++ CUJ articles <http://www.cuj.com/articles>
- C++libs Univ of vaasa <http://garbo.uwasa.fi/pc/c-lang.html>

19.1 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 Google, Yahoo, Lycos, Excite, Mamma.com for keyword "Linux memory debugging tools".
-

20. [Other Formats of this Document](#)

This document is published in 14 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, SGML (linuxdoc format), SGML (Docbook format), MS WinHelp format.

This howto document is located at –

- <http://www.linuxdoc.org> and click on HOWTOs and search for howto document name using CTRL+f or ALT+f within the web-browser.

You can also find this document at the following mirrors sites –

- <http://www.caldera.com/LDP/HOWTO>
 - <http://www.linux.ucla.edu/LDP>
 - <http://www.cc.gatech.edu/linux/LDP>
 - <http://www.redhat.com/mirrors/LDP>
 - Other mirror sites near you (network-address-wise) can be found at <http://www.linuxdoc.org/mirrors.html> select a site and go to directory /LDP/HOWTO/xxxxx-HOWTO.html
 - You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from – <ftp://www.linuxdoc.org/pub/linux/docs/HOWTO/other-formats/> and <http://www.linuxdoc.org/docs.html#howto>
 - Plain text format is in: <ftp://www.linuxdoc.org/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>
- Single HTML file can be created with command (see man sgml2html) – `sgml2html –split 0 xxxxhowto.sgml`
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in <ftp://www.linuxdoc.org/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 xxxxhowto.sgml` (to generate html file)
- `sgml2html –split 0 xxxxhowto.sgml` (to generate a single page html file)
- `sgml2rtf xxxxhowto.sgml` (to generate RTF file)
- `sgml2latex xxxxhowto.sgml` (to generate latex file)

20.1 Acrobat PDF format

PDF file can be generated from postscript file using either acrobat **distill** or **Ghostscript**. And postscript file is generated from DVI which in turn is generated from LaTeX file. You can download distill software from <http://www.adobe.com>. Given below is a sample session:

```
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.

20.2 Convert linuxdoc to Docbook format

This document is written in linuxdoc SGML format. The Docbook SGML format supersedes the linuxdoc format and has lot more features than linuxdoc. The linuxdoc is very simple and is easy to use. To convert linuxdoc SGML file to Docbook SGML use the program **ld2db.sh** and some PERL scripts. The `ld2db` output is not 100% clean and you need to use the **clean_ld2db.pl** PERL script. You may need to manually correct few lines in the document.

- Download `ld2db` program from <http://www.dcs.gla.ac.uk/~rrt/docbook.html> or from [Milkyway Galaxy site](#)
- Download the `cleanup_ld2db.pl` PERL script from from [Milkyway Galaxy site](#)

The `ld2db.sh` is not 100% clean, you will get lots of errors when you run

```
bash$ ld2db.sh file-linuxdoc.sgml db.sgml
bash$ cleanup.pl db.sgml > db_clean.sgml
bash$ gvim db_clean.sgml
bash$ docbook2html db.sgml
```

And you may have to manually edit some of the minor errors after running the PERL script. For e.g. you may need to put closing tag `</Para>` for each `<Listitem>`

20.3 Convert to MS WinHelp format

You can convert the SGML howto document to Microsoft Windows Help file, first convert the sgml to html using:

```
bash$ sgm12html xxxxhowto.sgml      (to generate html file)
bash$ sgm12html -split 0  xxxxhowto.sgml (to generate a single page html file)
```

Then use the tool [HtmlToHlp](#). You can also use `sgml2rtf` and then use the RTF files for generating winhelp files.

20.4 Reading various formats

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 –

C++ Programming HOW-TO

```
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.

21. [Translations To Other Languages](#)

- Translation to Polish is at <http://www.3miasto.net/~chq/c/howto/book1.htm> thanks to Darek Ostolski [Darek Ostolski](#)
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in <ftp://www.linuxdoc.org/pub/linux/docs/HOWTO> and <http://www.linuxdoc.org/docs.html#howto>

Any help from you to translate to other languages is welcome.

22. [Copyright](#)

Copyright policy is GNU/GPL as per LDP (Linux Documentation project). LDP is a GNU/GPL project. Additional requests are that you retain the author's name, email address and this copyright notice on all the copies. If you make any changes or additions to this document then you please intimate all the authors of this document. Brand names mentioned in this document are property of their respective owners.

23. [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
 - ◆ StringBuffer.h
 - ◆ StringTokenizer.h
 - ◆ StringRW.h
 - ◆ string_multi.h
 - ◆ example_String.cpp [example_String.cpp](#) click on 'Source code of C++ howto'.
 - File manipulation class, only length() function is implemented..
 - ◆ File.h
 - ◆ File.cpp [File.cpp](#) click on 'Source code of C++ howto'.
 - The zap() implemented here ..
 - ◆ my_malloc.h
 - ◆ my_malloc.cpp [my_malloc.cpp](#) click on 'Source code of C++ howto'.
 - Implementation of String class...
 - ◆ String.cpp [String.cpp](#) click on 'Source code of C++ howto'.
 - ◆ StringTokenizer.cpp [StringTokenizer.cpp](#) click on 'Source code of C++ howto'.
 - ◆ StringBuffer.cpp [StringBuffer.cpp](#) click on 'Source code of C++ howto'.
 - ◆ StringRW.cpp [StringRW.cpp](#) click on 'Source code of C++ howto'.
 - Debug facilities ..
 - ◆ debug.h
 - ◆ debug.cpp [debug.cpp](#) click on 'Source code of C++ howto'.
 - ◆ Makefile.unx
 - Sample Java file for testing the functionalities of String class ..
 - ◆ string.java
-

24. [Appendix B C++ v/s Java](#)

The students in Universities can debate the pros and cons of C++ and Java. Just for a debate sake, given below are some points discussing C++ versus Java:

1. **Generic Programming:** C++ has an extremely powerful template meta language, that allows a programmer to reuse many algorithms without paying neither performance, nor conveniences like type safety, for it. The somewhat equivalent for java, which is "all classes inherit from Object", so that you can have vectors, lists, cloneable, etc., is at the same time cumbersome and dangerous, because it forces programmers to duplicate code possibly introducing nasty minor bugs –the ones that results in intermitent failures, the worsts of all– doing that re–implementations. Conclusions are that :

- ◆ The issue is being addressed right now (next version of Java spec will have some genericity), justifying C++ for having it in since quite a while, and
- ◆ Language designers wanted to avoid compiler complications due to meta programming, praising C++ for being brave. When would we see things like "expression templates" in Java ? Perhaps never...

2. **Multiple Inheritance:** Java doesn't allow multiple inheritance: While it allows a class to implement as many interfaces as it wants, a new class can not make use of "arquetypical" implementations of the interfaces. What happens all the time, is that the programmer simply produces such implementation of the interface, and then copies and pastes that into each new class that implements it. The problem with that is the same as above: minor bugs leading to intermitent crashes, and also the problem that once you decide to change the interface.... you have to browse in all your code for each implementation to adapt it to the change.... that is the most anti Object Oriented practice you could find. Instead, in C++ you simply inherit some virtual methods from the "interface class", and just re-implement the really important pure virtual methods of the base class (the real interface). In that way, you can provide directly in the base class some services (methods) for implementers of its pure virtual ones. Some computer intellectuals say that multiple inheritance is an error. It is a very important tool for describing the reality to the computer (design).

3. **Operator overloading:** Suppose you are developing some numerical recipes, and you work with matrices, where you have to transpose, invert, add, multiply by scalar, etc. So, honestly, what do you prefer?: (A, B, C, D matrices and scale a double):

```
A = scale * B^-1 * (C + D) * ~B
(in C++ overloading * twice, +, ^, and ~ operators)
to this Java equivalent :
A.assign(B.invert().scalar_product(scale).product(C.add(D)).product(B.transpose()))
```

By the way, these are real life examples.

4. This differentiates C++ from Java: C++ is a very rich language, and yet, easily extensible (but not mutable, as Stroustrup says), but in the other hand, Java is a very poor language (although it is very useful), and not extensible at all. If you need something outside what normal Java can provide, you have no alternative – use JNI. Consequently, many many important programming techniques or design patterns can be naturally expressed in C++ only.

5. Some more points to come in future..
