C++ Programming HOW−TO

Table of Contents

Table of Contents

C++ Programming HOW−TO

Al Dev (Alavoor Vasudevan) alavoor@yahoo.com

v12.0, 10 July 2000

This document discusses methods to avoid memory problems in C++ and also will help you to program properly in C++ language. The information in this document applies to all the operating systems that is − Linux, MS DOS, BeOS, Apple Macintosh OS, Windows 95/98/NT/2000, OS/2, IBM OSes (MVS, AS/400 etc..), VAX VMS, Novell Netware, all flavors of Unix like Solaris, HPUX, AIX, SCO, Sinix, BSD, etc.. and to all other operating systems which support "C++" compiler (it means almost all the operating systems on this planet!).

---

1. Introduction

C++ is the most popular language and will be used for a long time in the future inspite of emergence of Java. C++ runs **extremely fast** and is in fact **10 to 20 times FASTER than** Java. Java runs very slow because it is an byte−code−interpreted language running on top of "virtual machine". Java runs faster with JIT compiler but is still slower than C++. And optimized C++ program is about **3 to 4 times faster** than Java using the JIT (Just−In−Time) compiler!! The memory management in Java is automated, so that programmers do not directly deal with memory allocations. This document attempts to automate the memory management in C++ to make it much more easy to use. A neat feature of Java is that memory allocations are taken care of automatically. This howto will enable "C++" to "compete/imitate" with Java language in memory management.

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

1.1 Problems facing the current C++ compilers

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

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

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

The usage of **char *** and **strcpy** causes *horrible* memory problems due to *"overflow"*, *"fence past errors"*, *"step−on−others−toe"* (hurting other variable's memory locations) or *"memory leaks"*. The memory problems are extremely hard to debug and are very time consuming to fix and trouble−shoot. Memory problems bring down the productivity of programmers. This document helps in increasing the productivity of programmers via different methods addressed to solve the memory defects in "C++". Memory related bugs are very tough to crack, and even experienced programmers take several days, weeks or months to debug memory related problems. Many times memory bugs will be "hiding" in the code for several months and can cause unexpected program crashes!! The usage of **char *** in C++ is costing USA and Japan $2 billion every year in time lost in debugging and downtime of programs. If you use **char *** in C++ then it is a very costly affair especially if your programs have more than 50,000 lines of code.

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

It is proposed that C++ compilers should prevent the programmers from using the **"char *"** , **"char[]"** datatypes and functions like **strcpy**, **strcat**, **strncpy**, **strncat**. The datatypes like char *, char[] and functions like strcpy, strcat are **evil** and must be completetly **BANNED** from usage in C++!! The **"char *"** is like *smallpox virus* and it must be eradicated from C++ world!! If you want to use "char *" as in some system functions than you should use "C" language. You would put all your "C" programs in a separate file and link to "C++" programs using the *linkage−specification* statement **extern "C"** −

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

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

The **extern "C"** statement says that everything within the brace−surrounded block − in this case, everything in the header file and comp(), some_c_function() is compiled by a C compiler.

Instead of using char * and char[] all the C++ programmers MUST use the '**String class**' which is given in this document and '**string class**' included in the STDLIB. The '**String class**' utilises the constructor and destructor to automate the memory management and also provides many functions like *ltrim*, *substring*, etc..

See also related '**string class**' in the C++ compiler. The **string class** is part of the standard GNU C++ library

and provides lot of string manipulation functions. The '**string class**' and '**String class**' can remove the need of **char *** datatype. Also, C++ programmers must be encouraged to use 'new', 'delete' features instead of using 'malloc' or 'free'.

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

The GNU C++ compiler MUST drop off the support of **char *, char[]** datatypes and in order to compile older programs using **char** datatype, the compiler should provide a additional option called "−fchar−datatype" to g++ command. Over the next 2 years all the C++ programs will use '**String class**' and '**string class**' and there will be no char * and char[]. The compiler should try to prevent bad programming practices!

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

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

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

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

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

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

---

2. Download String

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

- Go here and click on C++Programming howto.tar.gz file http://www.aldev.8m.com
- Mirror site : http://aldev.webjump.com

3. Usage of String class

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

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 H 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 = " Washington DC is the capital of USA ";

// 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);
}
```

3.1 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 eqaul to 'Bill Clinton' " << endl;

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

3.2 Functions

The functions provided by String class has the **same name** as that Java language's String class. The function names and the behaviour is **exactly** same as that of Java's string class!! This will facilitate portability of code between Java and C++ (you can cut and paste and do minimum changes to code).

The '**String class**' provides these **Java like** functions :−

- Current string length **length()**
- char charAt(int where);
- void getChars(int sourceStart, int sourceEnd, char target, int targetStart);
- char* toCharArray();
- bool equals(String str2); // See also == operator
- bool equals(char *str2); // See also == operator
- bool equalsIgnoreCase(String str2);
- bool regionMatches(int startIndex, String str2, int str2StartIndex, int numChars);
- bool regionMatches(bool ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars);
- String toUpperCase();
- String toLowerCase();
- bool startsWith(String str2);
- bool startsWith(char *str2);
- bool endsWith(String str2);
- bool endsWith(char *str2);
- int compareTo(String str2);
- int compareTo(char *str2);
- int compareToIgnoreCase(String str2);
- int compareToIgnoreCase(char *str2);
- int indexOf(char ch, int startIndex = 0);
- int indexOf(char *str2, int startIndex = 0);
- int indexOf(String str2, int startIndex = 0);
- int lastIndexOf(char ch, int startIndex = 0);
- int lastIndexOf(char *str2, int startIndex = 0);
- int lastIndexOf(String str2, int startIndex = 0);
- String substring(int startIndex, int endIndex = 0);
- String replace(char original, char replacement);
- String replace(char *original, char *replacement);
- String trim(); // See also overloaded trim()
- String concat(String str2); // See also operator +
- String concat(char *str2); // See also operator +
- String append(String str2) {return concat(str2);} // See also operator +

- String append(char *str2) {return concat(str2);} // See also operator +
- String append(int bb) {return (*this + bb);} // See also operator +
- String append(unsigned long bb) {return (*this + bb);} // See also operator +
- String append(float bb) {return (*this + bb);} // See also operator +
- String insert(int index, String str2);
- String insert(int index, char ch);
- String reverse(); // See also overloaded reverse()
- String deleteCharAt(int loc);
- String deleteStr(int startIndex, int endIndex); // Java's "delete()"

These are addional functions which are not available in Java.

- Left trim the string. Remove leading white−spaces − newlines, tabs **ltrim()**
- Right trim the string. Remove trailing white−spaces − newlines, tabs **rtrim()**
- Remove trailing and leading white−spaces **trim()**
- Remove trailing newlines **chop()**
- Change string to upper case **to_upper()**
- Change string to lower case **to_lower()**
- Truncate or round−off the float value **roundf(float input_val, short precision)**
- Truncate or round−off the double value **roundd(double input_val, short precision)**
- Find position, matching substr beginning from start **pos(char *substr, unsigned long start)**
- Explodes the string and returns the list in the list−head pointer explodeH **explode(char *seperator)**
- Implodes the strings in the list−head pointer explodeH and returns the String variable **implode(char *glue)**
- Joins the strings in the list−head pointer explodeH and returns the String variable **join(char *glue)**
- Repeat the input string n times **repeat(char *input, unsigned int multiplier)**
- Replace all occurences of string 'needle' with 'str' in the haystack 'val' **replace(char *needle, char *str)**
- Translate certain chars **str_tr(char *from, char *to)**
- Center the text string **center(int length, char padchar = ' ')**
- Formats the original string by placing 'number' of 'padchar' characters between each set of blank−delimited words. Leading and Trailing blanks are always removed. If 'number' is omitted or is 0, then all spaces are in the string are removed. The default number is 0 and default padchar ' ' **space(int number = 0, char padchar = ' ')**
- The result is string comprised of all characters between and including 'start' and 'end' **xrange(char start, char end)**
- Removes any characters contained in 'list'. The default character for 'list' is a blank ' ' **compress(char *list)**
- Deletes a portion of string of 'length' characters from 'start' position. If start is greater than the string length than string is unchanged **delstr(int start, int length)**
- The 'newstr' in inserted into val beginning at 'start'. The 'newstr' will be padded or truncated to 'length' characters. The default 'length' is string length of newstr **insert(char *newstr, int start = 0, int length = 0, char padchar = ' ')**
- The result is string of 'length' chars madeup of leftmost chars in val. Quick way to left justify a string **left(int length = 0, char padchar = ' ')**
- The result is string of 'length' chars madeup of rightmost chars in val. Quick way to right justify a string **right(int length = 0, char padchar = ' ')**
- The 'newstr' in overlayed into val beginning at 'start'. The 'newstr' will be padded or truncated to 'length' characters. The default 'length' is string length of newstr **overlay(char *newstr, int start = 0, int length = 0, char padchar = ' ')**
- Sub−string, extract a portion of string **substr(int start, int length = 0)**

- matches first match of regx **at(char *regx)**
- Returns string before regx **before(char *regx)**
- Returns string after regx **after(char *regx)**
- Returns true if string is NULL value **bool isnull()**
- Resets the string to NULL **clear()**

3.3 Miscellaneous Functions

Some miscellaneous String functions are given here, but **DO NOT USE** these, and instead use operators like '+', '+=', '==' etc.. These are 'private' members of the 'String' class.

- Copy string **str_cpy(char *bb)**
- Long integer converted to string **str_cpy(unsigned long bb)**
- Integer converted to string **str_cpy(int bb)**
- Float converted to string **str_cpy(float bb)**
- String concatenate a char * **str_cat(char *bb)**
- String concatenate a int **str_cat(int bb)**
- String concatenate a int **str_cat(unsigned long bb)**
- String concatenate a float **str_cat(float bb)**
- Is equal to String ? **bool equalto(const String & rhs, bool type = false)**
- Is equal to char* ? **bool equalto(const char *rhs, bool type = false)**

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 + " Honourable President Ronald Reagan " + 234.878;
cout << "The value of aa is : " << aa.val() << endl;
// The output aa will be '34 Honourable President Ronald Reagan 234.878'
```

4. C++ Zap (Delete) command

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

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

```
/*
** Use do while to make it robust and bullet−proof macro.
```

```
** For example, if "do−while" is NOT used then results will be
** something else just as in −
** if (bbint == 4)
**              aa = 0
** else
**              zap(aptr);  // Problem!! aptr will be always set to NULL
*/

#define zap(x) do { delete(x); x = NULL; } while (0)
```

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

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

        zap(pLastname);
        zap(pJobDescription);
```

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

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

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

5. Pointers are problems

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

Avoid using pointers as much as possible and use references. Pointers are really a great pain. It is possible to write a application without using pointers.

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

6. 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 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*.

7. Debug files

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

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

See the sample Makefile .

8. C++ Online−Docs

Visit the following C++ sites :−

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

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

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

8.1 C++ Tutorials

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

8.2 C++ Coding Standards

Visit the C++ Coding Standards URLs

- C++ coding standard http://www.cs.umd.edu/users/cml/cstyle/CppCodingStandard.html
- Coding standards from Possibility http://www.possibility.com/Cpp/CppCodingStandard.html
- Coding standards from Ambysoft http://www.ambysoft.com/javaCodingStandards.html
- Rules and recommendations http://www.cs.umd.edu/users/cml/cstyle/
- Indent and annotate http://www.cs.umd.edu/users/cml/cstyle/indhill−annot.html
- Elemental rules http://www.cs.umd.edu/users/cml/cstyle/Ellemtel−rules.html
- C++ style doc http://www.cs.umd.edu/users/cml/cstyle/Wildfire−C++Style.html

8.3 C++ Quick−Reference

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

8.4 C++ Usenet Newsgroups

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

9. Memory Tools

Use the following memory debugging tools

- On linux contrib cdrom see mem_test*.rpm package
- On linux cdrom see ElectricFence*.rpm package
- Purify Tool from Rational Software Corp http://www.rational.com
- Insure++ Tool from Parasoft Corp http://www.parasoft.com
- Linux Tools at http://www.xnet.com/~blatura/linapp6.html#tools
- Search the Internet engines like Yahoo, Lycos, Excite, Mamma.com for keyword "Linux memory debugging tools".

---

10. Related URLs

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

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

- Vim color text editor for C++, C http://metalab.unc.edu/LDP/HOWTO/Vim−HOWTO.html
- C++ Beautifier HOWTO http://metalab.unc.edu/LDP/HOWTO/C−C++Beautifier−HOWTO.html
- CVS HOWTO for C++ programs http://metalab.unc.edu/LDP/HOWTO/CVS−HOWTO.html
- Linux goodies main site http://www.aldev.8m.com
- Linux goodies mirror site http://aldev.webjump.com

---

11. Other Formats of this Document

This document is published in 11 different formats namely − DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages and SGML.

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

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

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

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

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

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

This howto document is located at −

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

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

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

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

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

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

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

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

To read postscript document give the command −

```
gv howto.ps
```

```
ghostscript howto.ps
```

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

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

---

12. 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.

---

13. Appendix A example_String.cpp

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

```
//*****************************************************************
// Copyright policy is GNU/GPL and it is requested  that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
//*****************************************************************

// To prevent memory leaks − a char class to manage character variables
// Always prefer to use  string class
// instead of char[] or char *
//

// To compile and test this program do −
// Assuming that libString.a is in the current directory
//              g++ example_String.cpp −L. −lString

#include <stdlib.h>  // for putenv
#include "String.h"
//#include <string>  // This is at /usr/include/g++−2/string
//#include <cstring>  // This is at /usr/include/g++−2/cstring and includes /usr/inlcude/strings.
//
void java_string_buffer();

////////////////////////////////////////////
// A example program to demo usage of String
// Note: In this example, I did not use memory
// manipulation functions like new, delete, malloc,
// strdup at all!! The String class takes care of
// it automatically !!
////////////////////////////////////////////

int main(int argc, char **argv)
{
        //java_string_buffer();
```

```
char p_name[1024];
sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
putenv(p_name);
print_total_memsize(); // in the beginning
String aa, bb, egg;
char  tmpaa[100];

//bb.str_cpy("  bbSTRing  ");
bb = "   bbSTRing   ";

// Testing the + operator
// aa + " rhs "; // You will not get any output here !!!
// You must directly use in fprintf as in below line −
fprintf(stdout, "1) aa.val() is :%sEOF\n", (aa + " my rhs " ).val());

// Testing the = operator
aa = " lhs " ;
fprintf(stdout, "2) With operator= aa.val() is :%sEOF\n", aa.val());

// Testing the + operator
// " lhs " + aa; // You will not get any output here !!!
// You must directly use in fprintf as in below line −
fprintf(stdout, "3) With lsh operator+, aa.val() is :%sEOF\n", (" my lhs " + aa ).val());

// ***************** Java like functions ********************
aa = "Some Value 2345";
fprintf(stdout, "4) aa.charAt() is :%c %sEOF\n", aa.charAt(3), aa.val());

aa = "Some Value 2345";
strcpy(tmpaa, "tmpaa value");
aa.getChars(3, 8, tmpaa, 2);
fprintf(stdout, "5) aa.getChars() is : %s %sEOF\n", tmpaa, aa.val());

aa = "Some Value 2345";
fprintf(stdout, "6) aa.toCharArray() is : %sEOF\n", aa.toCharArray());

aa = "Some2345";
if (aa.equals("Some2345"))
        fprintf(stdout, "7) aa.equals() is true : %sEOF\n", aa.val());
else
        fprintf(stdout, "7) aa.equals() is false : %sEOF\n", aa.val());

aa = "testinglettercase";
egg = "TestingLetterCase";
if (aa.equalsIgnoreCase(egg))
        fprintf(stdout, "8) egg equals aa (case insensitive) aa.val is :%sEOF\n", aa.val(
else
        fprintf(stdout, "8) egg not equals aa (case insensitive) aa.val is :%sEOF\n", aa.

aa = "kkktestinglettercase";
egg = "abtestingLetterCase";
if (aa.regionMatches(true, 3, egg, 2, 7))
        fprintf(stdout, "9) regionMatches is true aa.val is :%sEOF\n", aa.val());
else
        fprintf(stdout, "9) regionMatches is false aa.val is :%sEOF\n", aa.val());

//aa.str_cpy(bb.val());
aa = bb + "Some Value 2345";
egg = aa.toUpperCase();
fprintf(stdout, "10) egg.val is :%sEOF\n", egg.val());

aa = bb + "Some Value 2345";
```

```
egg = aa.toLowerCase();
fprintf(stdout, "11) egg.val is :%sEOF\n", egg.val());

aa = "Some Value 2345";
egg = "Some";
if (aa.startsWith("Some"))
//if (aa.startsWith(egg))
        fprintf(stdout, "12) aa.startsWith() is true :%sEOF\n", aa.val());
else
        fprintf(stdout, "12) aa.startsWith() is false :%sEOF\n", aa.val());

aa = "Some Value 2345";
egg = " 2345";
if (aa.endsWith(" 2345"))
//if (aa.endsWith(egg))
        fprintf(stdout, "13) aa.endsWith() is true :%sEOF\n", aa.val());
else
        fprintf(stdout, "13) aa.endsWith() is false :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
egg = "caabc";
if (aa.compareTo(egg) == 0)
        fprintf(stdout, "14) aa.compareTo() is zero :%sEOF\n", aa.val());
else
if (aa.compareTo(egg) > 0)
        fprintf(stdout, "14) aa.compareTo() is greater :%sEOF\n", aa.val());
else
if (aa.compareTo(egg) < 0)
        fprintf(stdout, "14) aa.compareTo() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
strcpy(tmpaa, "aabbb Some Value 2345");
if (aa.compareTo(tmpaa) == 0)
        fprintf(stdout, "15) aa.compareTo() is zero :%sEOF\n", aa.val());
else
if (aa.compareTo(tmpaa) > 0)
        fprintf(stdout, "15) aa.compareTo() is greater :%sEOF\n", aa.val());
else
if (aa.compareTo(tmpaa) < 0)
        fprintf(stdout, "15) aa.compareTo() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
//egg = "bbb Some Value 2345";
egg = "CCaabc";  // change values to caabc, aabc
if (aa.compareToIgnoreCase(egg) == 0)
        fprintf(stdout, "16) aa.compareToIgnoreCase() is zero :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(egg) > 0)
        fprintf(stdout, "16) aa.compareToIgnoreCase() is greater :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(egg) < 0)
        fprintf(stdout, "16) aa.compareToIgnoreCase() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
//strcpy(tmpaa, "bbb Some Value 2345");
strcpy(tmpaa, "CAABbb Some Value 2345"); // change value to caabb, aab
if (aa.compareToIgnoreCase(tmpaa) == 0)
        fprintf(stdout, "17) aa.compareToIgnoreCase() is zero :%sEOF\n", aa.val());
else
if (aa.compareToIgnoreCase(tmpaa) > 0)
        fprintf(stdout, "17) aa.compareToIgnoreCase() is greater :%sEOF\n", aa.val());
else
```

```
if (aa.compareToIgnoreCase(tmpaa) < 0)
        fprintf(stdout, "17) aa.compareToIgnoreCase() is less than :%sEOF\n", aa.val());

aa = "bbb Some Value 2345";
strcpy(tmpaa, "Some");
egg = "Value";
fprintf(stdout, "18) aa.indexOf('S') %d :%sEOF\n", aa.indexOf('S'),  aa.val());
fprintf(stdout, "18) aa.indexOf(tmpaa) %d :%sEOF\n", aa.indexOf(tmpaa),  aa.val());
fprintf(stdout, "18) aa.indexOf(egg) %d :%sEOF\n", aa.indexOf(egg),  aa.val());

aa = "bbb Some Value Some 2345";
strcpy(tmpaa, "Some");
egg = "Some";
fprintf(stdout, "19) aa.lastIndexOf('S') %d :%sEOF\n", aa.lastIndexOf('S'),  aa.val());
fprintf(stdout, "19) aa.lastIndexOf(tmpaa) %d :%sEOF\n", aa.lastIndexOf(tmpaa),  aa.val()
fprintf(stdout, "19) aa.lastIndexOf(egg) %d :%sEOF\n", aa.lastIndexOf(egg),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "20) aa.substring(5) %s :%sEOF\n",
                aa.substring(5).val(),  aa.val());

aa = "bbb Some Value Some 2345";
strcpy(tmpaa, "Some");
egg = "Some";
fprintf(stdout, "20) aa.replace('S', 'V') %s :%sEOF\n",
                aa.replace('S', 'V').val(),  aa.val());
fprintf(stdout, "20) aa.replace(Som, Vzz) %s :%sEOF\n",
                aa.replace("Som", "Vzz").val(),  aa.val());

aa = "   bbb Some Value Some 2345   ";
fprintf(stdout, "21) aa.trim() :%sEOF val() :%sEOF\n",
                aa.trim().val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "21) aa.concat() %s val :%sEOF\n",
                aa.concat("add one").val(),  aa.val());

//aa = "bbb Some Value Some 2345";
//fprintf(stdout, "21) aa.append() %s val :%sEOF\n",
//              aa.append("add append").val(),  aa.val());

aa = "bbb Some Value Some 2345";
egg = "jjjj";
fprintf(stdout, "21) aa.insert(5, egg) %s val :%sEOF\n",
                aa.insert(5, egg).val(),  aa.val());
fprintf(stdout, "21) aa.insert(5, ch) %s val :%sEOF\n",
                aa.insert(5, 'M').val(),  aa.val());

aa = "12345678";
fprintf(stdout, "46) aa.reverse()=%s aa.val is :%sEOF\n", aa.reverse().val(), aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "21) aa.deleteCharAt(4) %s val :%sEOF\n",
                aa.deleteCharAt(4).val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "22) aa.deleteStr(3,5) %s val :%sEOF\n",
                aa.deleteStr(3,5).val(),  aa.val());

// ***************** end Java like functions ********************

aa = "bbb Some Value Some 2345";
```

```
fprintf(stdout, "23) aa.str_tr(bomekk, BOME) %s val :%sEOF\n",
                aa.tr("bomekk", "BOME").val(),  aa.val());

aa = "bbb Some Value Some 2345";
aa = "$1,934 100%.234";
fprintf(stdout, "24) aa.compress() %s val :%sEOF\n",
                aa.compress("$,%").val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "25) aa.xrange('a', 'j') %s val :%sEOF\n",
                aa.xrange('a', 'j').val(),  aa.val());
fprintf(stdout, "25) aa.xrange('1', '8') %s val :%sEOF\n",
                aa.xrange('1', '8').val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "26) aa.center(15) %s val :%sEOF\n",
                aa.center(15).val(),  aa.val());
fprintf(stdout, "26) aa.center(15, '*') %s val :%sEOF\n",
                aa.center(15, '*').val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "27) aa.space(3) %s val :%sEOF\n",
                aa.space(3).val(),  aa.val());

aa = "      Some Value Some 2345";
fprintf(stdout, "28) aa.left() %s val :%sEOF\n",
                aa.left().val(),  aa.val());
fprintf(stdout, "28) aa.left(18) %s val :%sEOF\n",
                aa.left(18).val(),  aa.val());

aa = "  2345    ";
fprintf(stdout, "29) aa.right():%s val :%sEOF\n",
                aa.right().val(),  aa.val());
fprintf(stdout, "29) aa.right(5):%s val :%sEOF\n",
                aa.right(5).val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "30) aa.overlay(12345678, 4, 10, *):%s val :%sEOF\n",
                aa.overlay("12345678", 4, 10, '*').val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "31) aa.at(Som) %s :%sEOF\n",
                aa.at("Som").val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "32) aa.before(Som) %s :%sEOF\n",
                aa.before("Skkkom").val(),  aa.val());

aa = "bbb Some Value Some 2345";
fprintf(stdout, "33) aa.after(Som) %s :%sEOF\n",
                aa.after("Som").val(),  aa.val());

aa = "   bb some value   ";
aa.ltrim(true);
fprintf(stdout, "34) aa.val is :%sEOF\n", aa.val());

aa = "   bb some value   ";
aa.rtrim(true);
fprintf(stdout, "35) aa.val() is :%sEOF\n", aa.val());

aa = "   bb some value   ";
aa.trim(true);
```

```
fprintf(stdout, "36) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = aa + " testing newlines \n\n\n\n";
aa.chopall();
fprintf(stdout, "37) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = aa + " rhs ";
fprintf(stdout, "38) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = " lhs " + aa;
fprintf(stdout, "39) aa.val() is :%sEOF\n", aa.val());

// Sample addition of numbers
//aa = (String) 9989 + "kkk" + 33 ;
aa = 9999;
fprintf(stdout, "40) aa.val() is :%sEOF\n", aa.val());

aa = bb;
aa = " lhs " + aa + " rhs " + " 9989 " + " 33 ";
fprintf(stdout, "41) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = bb + "alkja " + " 99djd " ;
fprintf(stdout, "42) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = (String) "alkja " + " 99djd " ;
fprintf(stdout, "43) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa += (String) " al dev test kkk... " + " al2 slkj" + " al3333 ";
fprintf(stdout, "44) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = aa + " add aa " + aa + aa + aa + " 1111 " + " 2222 " + aa + aa + aa + " 3333 ";
fprintf(stdout, "45) aa.val() is :%sEOF\n", aa.val());

aa = "12345678";
aa.reverse(true);
fprintf(stdout, "46) aa.val() is :%sEOF\n", aa.val());

aa = " AA value ";
aa = aa + " add aa " + aa + 1111 +" "+ 2222 + " " + 3.344 + aa;
fprintf(stdout, "47) aa.val() is :%sEOF\n", aa.val());

aa.roundd(123456.0123456789012345, 13);
fprintf(stdout, "48) double aa.val() is :%sEOF\n", aa.val());

aa.roundf(123456.0123456789, 13);
fprintf(stdout, "49) float aa.val() is :%sEOF\n", aa.val());

// Test equal to operators
aa = " AA value ";
String cc(" AA value ");
if (aa == cc)
        fprintf(stdout, "50)aa=%s and cc=%s are equal!!\n", aa.val(), cc.val());
else
        fprintf(stdout, "51)aa=%s and cc=%s are NOT equal!!\n", aa.val(), cc.val());
cc = "CC";
```

```
if (aa == cc)
        fprintf(stdout, "52)aa=%s and cc=%s are equal!!\n", aa.val(), cc.val());
else
        fprintf(stdout, "53)aa=%s and cc=%s are NOT equal!!\n", aa.val(), cc.val());
if (aa == " AA value ")
        fprintf(stdout, "54)aa=%s and string are equal!!\n", aa.val());
else
        fprintf(stdout, "55)aa=%s and string are NOT equal!!\n", aa.val());
if (aa == " AA valuexxx ")
        fprintf(stdout, "56)aa=%s and string are equal!!\n", aa.val());
else
        fprintf(stdout, "57)aa=%s and string are NOT equal!!\n", aa.val());

aa = " AA bb value 12345678 ";
fprintf(stdout, "58) aa.length() is :%ldEOF\n", aa.length());

aa = " AA bb value 12345678 ";
fprintf(stdout, "59) aa.repeat(BA, 4).val=%s aa.val() is :%sEOF\n",
                aa.repeat("BA", 4).val(), aa.val());

aa = "";
aa = "aa";
if (aa.isNull())
        fprintf(stdout, "60) aa.isNull() result=true%sEOF\n", aa.val());
else
        fprintf(stdout, "60) aa.isNull() result=false%sEOF\n", aa.val());

aa = " some value aa";
aa.clear();
fprintf(stdout, "61) aa.clear() %sEOF\n", aa.val());

aa = "  abcd efg  hijk  lmno     ";
fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                aa.token().val(),  aa.val());
fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                aa.token().val(),  aa.val());
fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                aa.token().val(),  aa.val());
fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                aa.token().val(),  aa.val());
fprintf(stdout, "62) aa.token():%s val :%sEOF\n",
                aa.token().val(),  aa.val());

aa = " 2345 ";
if (aa.isInteger()) // is true
        fprintf(stdout, "63) aa is a integer val :%sEOF\n", aa.val());
else
        fprintf(stdout, "63) aa is NOT a integer val :%sEOF\n", aa.val());

aa = " 23.045  ";
if (aa.isNumeric()) // is true
        fprintf(stdout, "64) aa is a numeric val :%sEOF\n", aa.val());
else
        fprintf(stdout, "64) aa is NOT a numeric val :%sEOF\n", aa.val());

aa = " 23045  ";
fprintf(stdout, "65) aa.int_value()=%d val :%sEOF\n",
                aa.int_value(), aa.val());

aa = " 230.45  ";
fprintf(stdout, "66) aa.double_value()=%f val :%sEOF\n",
                aa.double_value(), aa.val());
```

```
        aa = " testing abcdefg";
        aa.chop();
        fprintf(stdout, "68) aa.chop() aa.val is :%sEOF\n", aa.val());

        aa = " str1 str2 string3 abcdefg joe john hardy  ";
        String *strlist;
        int strcount = 0;
        strlist = aa.explode(strcount);
        for (int ii = 0; ii <= strcount; ii++)
        {
                fprintf(stdout, "69) strlist[%d] is :%sEOF\n",
                                ii, strlist[ii].val());
        }

        aa = " some aa ";
        cout << "\n\nPlease enter a line and hit return key : ";
        aa.getline();
        fprintf(stdout, "70) aa.getline() is :%sEOF\n", aa.val());

        aa = " some aa ";
        cout << "71) Testing << operator aa is : " << aa << endl;

        aa = " some aa ";
        cout << "\n\n73) Testing >> operator. Enter value for aa : ";
        cin >> aa;
        cout << "73) Testing >> operator aa is : " << aa << endl;

        // You can use aa.val() like a 'char *' variable in programs !!
        aa = " str1 str2 string3 abcdefg joe john hardy  ";
        fprintf(stdout, "\n\n74) Test case using aa.val() as 'char []' variable... ");
        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]);
        }
        fprintf(stdout, "\n");

        // Using pointers on 'char *' ...
        fprintf(stdout, "\n\n75) Test case using aa.val() as 'char *' pointers... ");
        aa = " str1 str2 string3 abcdefg joe john hardy  ";
        for (char *tmpcc = aa.val(); *tmpcc != 0; tmpcc++)
        {
                fprintf(stdout, "aa.val()=%c ", *tmpcc);
        }
        fprintf(stdout, "\n");

        print_total_memsize(); // in the end

        java_string_buffer();
        exit(0);
}

// Sample code to demo imitation of Java's StringBuffer Object
void java_string_buffer()
{
        String str1 = "ABCD EFGHI";
        cout << "\nAssigned value to str1 " << endl;
        StringBuffer aa;
        StringBuffer bb(30);
        StringBuffer cc(str1);
```

```
cout << "\n StringBuffer aa.length() : " << aa.length() << endl;
cout << "\n StringBuffer aa.capacity() : " << aa.capacity() << endl;
cout << "\n StringBuffer aa.ensureCapacity(28) : " << endl;
aa.ensureCapacity(28);
cout << "\n StringBuffer aa.setLength(38) : " << endl;
aa.setLength(38);

// In Java, you use new, in C++ remove new
// StringBuffer dd = new StringBuffer("some value for string buffer");
StringBuffer dd = StringBuffer("some value for string buffer");
cout << "\n StringBuffer dd.charAt(3) : " << dd.charAt(3) << endl;

dd.setCharAt(3, 'K');
cout << "\n StringBuffer setCharAt(3) : " << dd.charAt(3) << endl;

char ee[100];
memset(ee, 0, 100);
strcpy(ee, "1111111111111111111111111111111111111");
dd.getChars(2, 12, ee, 3);
cout << "\n StringBuffer getChars(2, 12, ee, 3) : " << ee << endl;

dd.append(str1);
cout << "\n StringBuffer append() : " << dd << endl;

dd.append("12345678");
cout << "\n StringBuffer append() : " << dd << endl;

dd.append(9);
cout << "\n StringBuffer append() : " << dd << endl;

dd.append(7.876);
cout << "\n StringBuffer append() : " << dd << endl;

dd.setLength(1);
dd.append(" some value for dd");
dd.insert(4, str1);
cout << "\n StringBuffer insert() : " << dd << endl;

dd.reverse();
cout << "\n StringBuffer reverse() : " << dd << endl;

dd.setLength(1);
dd.append(" some value for dd");
dd.deleteStr(4, 9);  // Java's delete()
cout << "\n StringBuffer deleteStr(4,9) : " << dd << endl;

dd.setLength(1);
dd.append(" some value for dd");
dd.deleteCharAt(6);
cout << "\n StringBuffer deleteCharAt() : " << dd << endl;

dd.setLength(1);
dd.append(" some value for dd");
dd.replace(3, 8, str1);
cout << "\n StringBuffer replace() : " << dd << endl;

dd.setLength(1);
dd.append(" some value for dd. A quick brown fox.");
dd.substring(8);
cout << "\n StringBuffer substring(8) : " << dd << endl;

dd.setLength(1);
```

```
        dd.append(" some value for dd akjla akja kjk");
        dd.substring(8, 14);
        cout << "\n StringBuffer substring(8) : " << dd << endl;

        exit(0);
}
```

---

14. Appendix B String.h

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

```
//*****************************************************************
// Copyright policy is GNU/GPL and it is requested that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
//*****************************************************************

// To prevent memory leaks − a char class to manage character variables
// Always prefer to use String or string class
// instead of char[] or char *
//

#ifndef __STRING_H_
#define __STRING_H_

//#include <iostream> // do not use iostream as program becomes bulky..
//#include <stdlib.h> // for free() amd malloc()
#include <string.h> // for strcpy()
#include <ctype.h> // for isspace()
#include <stdio.h> // for sprintf()
#include <list.h> // for sprintf()
#include <math.h> // for modf(), rint()

#include "my_malloc.h"
#include "debug.h" // debug_(name, value)  debug2_(name, value, LOG_YES)

const short INITIAL_SIZE =      50;
const short NUMBER_LENGTH = 70;
const int MAX_ISTREAM_SIZE = 2048;

// a small class with a VERY MINIMUM of functions and variables...
// This class to be kept small...
class String
{
        public:
                String();
                String(char bb[]);  // needed by operator+
                String(int bb);  // needed by operator+
                String(unsigned long bb);  // needed by operator+
                String(long bb);  // needed by operator+
                String(float bb);  // needed by operator+
                String(double bb);  // needed by operator+
                String(const String & rhs);  // Copy Constructor needed by operator+
                String(int bb, bool dummy);  // for StringBuffer class
                ~String();
```

```
char *val() {return sval;} // Not safe to make sval public

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

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

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

String toUpperCase();
String toLowerCase();

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

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

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

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

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

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

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

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

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

String valueOf(double num) {return String(num);}
String valueOf(long num){ return String(num);}
String valueOf(char chars[]){ return String(chars);}
String valueOf(char chars[], int startIndex, int numChars);

void ensureCapacity(int capacity);
void setLength(int len);
```

```
void setCharAt(int where, char ch);
// See also StringBuffer class in this file given below

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

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

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

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

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

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

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

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

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

void clear();
int int_value();
double double_value();
String token(char seperator = ' ');  // see also explode()
String crypt(char *original, char *salt);
String getline(FILE *infp = stdin); // see also putline()
//String getline(fstream *infp = stdin); // see also putline()

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

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

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

////////////////////////////////////////////
//              List of duplicate function names
////////////////////////////////////////////
// char * c_str() // use val()
// bool find();  // Use regionMatches()
// bool search();  // Use regionMatches()
// bool matches(); // Use regionMatches()
// int rindex(String str2, int startIndex = 0); Use lastIndexOf()
// String blanks(int slength);  // Use repeat()
// String append(String str2); // Use concat() or + operator
// String prepend(String str2);  // Use + operator. See also append()
// String split(char seperator = ' ');  // Use token()
bool contains(char *str2, int startIndex = 0); // use indexOf()
// void empty(); Use is_empty()
// void vacuum(); Use clear()
// void erase(); Use clear()
// void zero(); Use clear()
// bool is_float(); Use is_numeric();
// bool is_decimal(); Use is_numeric();
// bool is_Digit(); Use is_numeric();
// double float_value(); Use double_value();
// double tofloat(); Use double_value();
// double numeric_value(); Use double_value();
// int tointeger(); Use int_value()
// int tonumber(); Use int_value()
```

```
        // String get(); Use substring()
        // String getFrom(); Use substring()
        // String head(int len); Use substring(0, len)
        // String tail(int len); Use substring(length()−len, length())
        // String cut(); Use deleteCharAt() or deleteStr()
        // String cutFrom(); Use deleteCharAt() or deleteStr()
        // String paste(); Use insert()
        // String fill(); Use replace()
        // char firstChar(); // Use substring(0, 1);
        // char lastChar(); // Use substring(length()−1, length());
        // String findNext(); Use token()

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

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

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

        static  list<String>                explodeH;  // list head

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

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

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

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

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

```
                inline void _ltrim();
                inline void _rtrim();
                inline void _substring(int startIndex, int endIndex);
};

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

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

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

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

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

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

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

// Global variables are defined in String.cpp

#endif // __STRING_H_
```

15. Appendix C String.cpp

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

```
//*****************************************************************
// Copyright policy is GNU/GPL and it is requested that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
//*****************************************************************

// To prevent memory leaks − a char class to manage character variables
// Always prefer to use  string class
// instead of char[] or char *
//

// To compile and test this program do −
//              g++ String.cpp

#include "String.h"

//#include <sys/va_list.h> for Format()
//#include <sys/varargs.h> for Format()

// Global variables ....
//String *String::_global_String = NULL; // global var
list<String>                 String::explodeH;

String::String()
{
        debug_("In cstr()", "ok");
        sval = (char *) my_malloc(sizeof(char)* INITIAL_SIZE);

        _pString = NULL;
}

String::String(char *bb)
{
        unsigned long tmpii = strlen(bb);
        sval = (char *) my_malloc(sizeof(char)* tmpii);
        strncpy(sval, bb, tmpii);
        sval[tmpii] = '\0';

        //debug_("In cstr(char *bb) bb", bb);
        debug_("In cstr(char *bb) sval", sval);
        #ifdef DEBUG
                //fprintf(stderr, "\nAddress of sval=%x\n", & sval);
                //fprintf(stderr, "\nAddress of this−pointer=%x\n", this);
        #endif // DEBUG

        _pString = NULL;
}

String::String(int bb)
{
        sval = (char *) my_malloc(NUMBER_LENGTH); // integers 70 digits max
        sprintf(sval, "%d", bb);
        debug_("In cstr(int bb) sval", sval);
```

```
        _pString = NULL;
}

String::String(unsigned long bb)
{
        sval = (char *) my_malloc(NUMBER_LENGTH); // long 70 digits max
        sprintf(sval, "%lu", bb);
        debug_("In cstr(unsigned long bb) sval", sval);

        _pString = NULL;
}

String::String(long bb)
{
        sval = (char *) my_malloc(NUMBER_LENGTH); // long 70 digits max
        sprintf(sval, "%ld", bb);
        debug_("In cstr(long bb) sval", sval);

        _pString = NULL;
}

String::String(float bb)
{
        sval = (char *) my_malloc(NUMBER_LENGTH); // float 70 digits max
        sprintf(sval, "%f", bb);
        debug_("In cstr(float bb) sval", sval);

        _pString = NULL;
}

String::String(double bb)
{
        sval = (char *) my_malloc(NUMBER_LENGTH); // double 70 digits max
        sprintf(sval, "%f", bb);
        debug_("In cstr(double bb) sval", sval);

        _pString = NULL;
}

// Copy Constructor needed by operator +
String::String(const String & rhs)
{
        // Do a deep−copy instead of compiler's default shallow copy copy−cstr
        debug_("In copy−cstr()", "ok");
        unsigned long tmpii = strlen(rhs.sval);
        sval = (char *) my_malloc(sizeof(char)* tmpii);
        strncpy(sval, rhs.sval, tmpii);
        sval[tmpii] = '\0';

        _pString = NULL;
}

// For use by StringBuffer class. Put a dummy
// variable for different signature.
// StringBuffer class imitates Java's StringBuffer object
String::String(int size, bool dummy)
{
        sval = (char *) my_malloc(sizeof(char)* size);
        debug_("In cstr(int size, bool dummy) sval", sval);
        #ifdef DEBUG
                //fprintf(stderr, "\nAddress of sval=%x\n", & sval);
                //fprintf(stderr, "\nAddress of this−pointer=%x\n", this);
```

```
        #endif // DEBUG

        _pString = NULL;
}

String::~String()
{
        debug_("In dstr sval", sval);
        #ifdef DEBUG
                //fprintf(stderr, "\nAddress of sval=%x\n", & sval);
                //fprintf(stderr, "\nAddress of this−pointer=%x\n", this);
        #endif // DEBUG
        my_free(sval);
        //delete [] sval;
        sval = NULL;

        delete _pString; _pString = NULL;
}

inline void String::_allocpString()
{
        // _pString will be deleted in destructor
        if (!_pString)  // if (_pString == NULL)
                _pString = new String(this−>sval);
        else
                *_pString = this−>sval;
}

// MUST use pointer−to−pointer **aa, otherwise the argument
// is NOT freed !!
/*
inline void String::_free_glob(String **aa)
{
        debug_("called _free_glob()", "ok" );
        if (*aa != NULL)  // (*aa != NULL)
        {
                debug_("*aa is not null", "ok");
                delete *aa;
                *aa = NULL;
        }
        //else
                debug_("*aa is null", "ok");

        //if (*aa == NULL)
        debug_("*aa set to null", "ok");
}
*/

// Imitating Java's charAt string function...
char String::charAt(int where)
{
        verifyIndex(where);
        return (sval[where]);
}

// Imitate Java's getChars function...
// The sourceStart specifies the index of the beginning of the substring
// and sourceEnd specifies an index that is one past the end of desired
// substring. Thus the substring contains characters from sourceStart
// through (sourceEnd − 1). The array that will receive the characters
// is specified by target. The index within target at which the substring
// will be copied is passed in targetStart. Care must be taken to assure
```

```
// that the target array is large enough to hold the number of characters
// in the specified substring.
// For e.g.  getChars(3, 6, aa, 0) on "ABCDEFGHIJK" gives aa ="DEF"
void String::getChars(int sourceStart, int sourceEnd, char target[], int targetStart)
{
        verifyIndex(sourceStart);
        verifyIndex(sourceEnd);

        if (sourceEnd >= sourceStart)
        {
                strncpy(& target[targetStart], & sval[sourceStart], sourceEnd − sourceStart);
                target[targetStart + (sourceEnd − sourceStart)] = 0;
        }
        else
        {
                cerr << "\ngetChars() − SourceEnd is greater than SourceStart!!\n" << endl;
                exit(1);
        }
}

// Imitate Java's getChars string function...
// Returns array of characters for the entire string
char* String::toCharArray()
{
        return (sval);
}

// Imitate Java's getBytes string function...
// Returns array of characters for the entire string
char* String::getBytes()
{
        return (sval);
}

// Imitate Java's equals string function...
bool String::equals(String str2)  // See also == operator
{
        return ( _equalto(str2.sval));
}

// Imitate Java's equals string function...
bool String::equals(char *str2)  // See also == operator
{
        return ( _equalto(str2));
}

// Imitate Java's equalsIgnoreCase string function...
bool String::equalsIgnoreCase(String str2)
{
        String  aa, bb;
        aa = this−>toLowerCase();
        bb = str2.toLowerCase();
        return ( aa._equalto(bb.sval) );
}

// Imitate Java's regionMatches string function...
// The startIndex specifies the index at which the region begins within
// the invoking String object. The string being compared is str2. The
// index at which comparison will start within str2 is specified by
// str2Index. The length of the substring being compared is numChars.
bool String::regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)
{
```

```
        verifyIndex(startIndex);
        str2.verifyIndex(str2StartIndex);
        if (strncmp(& this->sval[startIndex], & str2.sval[str2StartIndex], numChars) == 0)
                return true;
        else
                return false;
}

// Imitate Java's regionMatches string function...
// This is overloaded function of regionMatches
// If ignoreCase is true, the case of the characters is ignored, otherwise
// case is significant (i.e. if ignoreCase is true then ignore the
// case and compare)
// The startIndex specifies the index at which the region begins within
// the invoking String object. The string being compared is str2. The
// index at which comparison will start within str2 is specified by
// str2Index. The length of the substring being compared is numChars.
bool String::regionMatches(bool ignoreCase, int startIndex, String str2, int str2StartIndex, int
{
        if (ignoreCase)  // if (ignoreCase == true)
        {
                verifyIndex(startIndex);
                str2.verifyIndex(str2StartIndex);
                String string1, string2;
                string1 = this->toLowerCase();
                string2 = str2.toLowerCase();
                if (strncmp(& string1.sval[startIndex], & string2.sval[str2StartIndex], numChars)
                        return true;
                else
                        return false;
        }
        else
        {
                return regionMatches(startIndex, str2, str2StartIndex, numChars);
        }
}

// Imitate Java's toLowerCase string function...
//       String  ss("sometest");
//       String  egg = ss.toLowerCase();
String String::toLowerCase()
{
        _allocpString();

        for (long tmpii = strlen(_pString->sval); tmpii >= 0; tmpii--)
        {
                _pString->sval[tmpii] = tolower(_pString->sval[tmpii]);
        }
        return *_pString;  // return the object now
}

// Imitate Java's toUpperCase string function...
//       String  ss("sometest");
//       String  egg = ss.toUpperCase();
String String::toUpperCase()
{
        _allocpString();

        for (long tmpii = strlen(_pString->sval); tmpii >= 0; tmpii--)
        {
                _pString->sval[tmpii] = toupper(_pString->sval[tmpii]);
        }
```

```
        return *_pString;  // return the object now
}

// Imitate Java's startsWith string function...
bool String::startsWith(String str2)
{
        if (!strncmp(this->sval, str2.sval, strlen(str2.sval) )) // if (strncmp() == 0)
                return true;
        else
                return false;
}

// Imitate Java's startsWith string function...
// overloaded function
bool String::startsWith(char *str2)
{
        int lenstr2 = strlen(str2);
        if (!strncmp(this->sval, str2, lenstr2)) // if (strncmp() == 0)
                return true;
        else
                return false;
}

// Imitate Java's endsWith string function...
bool String::endsWith(String str2)
{
        // string length of str2 should be less than current string
        if (strlen(str2.sval) > strlen(sval))
                return false;

        if (!strncmp(& this->sval[strlen(sval) - strlen(str2.sval)], str2.sval, strlen(str2.sval)
                return true;
        else
                return false;
}

// Imitate Java's endsWith string function...
bool String::endsWith(char *str2)
{
        // string length of str2 should be less than current string
        if (strlen(str2) > strlen(sval))
                return false;

        if (!strncmp(& this->sval[strlen(sval) - strlen(str2)], str2, strlen(str2) ) ) // if (str
                return true;
        else
                return false;
}

// Imitate Java's compareTo string function...
// For sorting applications, you need to know which is less than, equal to
// or greater than the next.
// A string is less than another if it comes before the other in dictionary
// order. A string is greater than another if it comes after the other in
// dictionary order.
//  Less than zero  --> The invoking string is less than str2
//  Greater than zero  --> The invoking string is greater than str2
//  Zero  --> The two strings are equal.
int String::compareTo(String str2)
{
        int  flag = 0;
        // Compare letters in string to each letter in str2
```

```
        for (int tmpii = 0, tmpjj = strlen(sval), tmpkk = strlen(str2.sval); tmpii < tmpjj; tmpii
        {
                if (tmpii > tmpkk)
                        break;
                if (sval[tmpii] == str2.sval[tmpii])
                        flag = 0;
                else
                if (sval[tmpii] > str2.sval[tmpii])
                {
                        flag = 1;
                        break;
                }
                else // if (sval[tmpii] < str2.sval[tmpii])
                {
                        flag = −1;
                        break;
                }
        }
        return flag;
}

// Imitate Java's compareTo string function...
// Overloaded function of compareTo
int String::compareTo(char *str2)
{
        int  flag = 0;
        // Compare letters in string to each letter in str2
        for (int tmpii = 0, tmpjj = strlen(sval), tmpkk = strlen(str2); tmpii < tmpjj; tmpii++)
        {
                if (tmpii > tmpkk)
                        break;
                if (sval[tmpii] == str2[tmpii])
                        flag = 0;
                else
                if (sval[tmpii] > str2[tmpii])
                {
                        flag = 1;
                        break;
                }
                else // if (sval[tmpii] < str2[tmpii])
                {
                        flag = −1;
                        break;
                }
        }
        return flag;
}

// Imitate Java's compareToIgnoreCase string function...
int String::compareToIgnoreCase(String str2)
{
        String tmpaa = this−>toLowerCase(),
        tmpbb = str2.toLowerCase();

        return tmpaa.compareTo(tmpbb);
}

// Imitate Java's compareToIgnoreCase string function...
// Overloaded function
int String::compareToIgnoreCase(char *str2)
{
        String tmpaa = this−>toLowerCase(),
```

```
        tmpcc(str2), tmpbb = tmpcc.toLowerCase();

        return tmpaa.compareTo(tmpbb);
}

// Imitate Java's indexOf string function...
// Searches for the first occurence of a character or string
// Return the index at which the character or substring was
// found, or −1 on failure.
int String::indexOf(char ch, int startIndex = 0)
{
        verifyIndex(startIndex);
        int ii = startIndex;
        for (; ii < (int) strlen(sval); ii++)
        {
                if (sval[ii] == ch)
                        break;
        }
        if (ii == (int) strlen(sval))
                return −1;
        return ii;
}

// Imitate Java's indexOf string function...
// Overloaded function
int String::indexOf(char *str2, int startIndex = 0)
{
        verifyIndex(startIndex);
        char * tok;
        long res = −1;

        if ( !isNull() )
        {
                tok = strstr(sval + startIndex, str2);
                if (tok == NULL)
                        res = −1;
                else
                        res = (int) (tok − sval);
        }
        return res;
}

// Imitate Java's indexOf string function...
// Overloaded function
int String::indexOf(String str2, int startIndex = 0)
{
        verifyIndex(startIndex);
        char * tok;
        long res = −1;

        if ( !isNull() )
        {
                tok = strstr(sval + startIndex, str2.sval);
                if (tok == NULL)
                        res = −1;
                else
                        res = (int) (tok − sval);
        }
        return res;
}

// Imitate Java's lastIndexOf string function...
```

```
// Searches for the last occurence of a character or string
// Return the index at which the character or substring was
// found, or −1 on failure.
int String::lastIndexOf(char ch, int startIndex = 0)
{
        verifyIndex(startIndex);
        int ii;

        // Begin search from the last character of string
        if (!startIndex) // if (startIndex == 0)
                ii = strlen(sval);
        else
                ii = startIndex;
        for (; ii > −1; ii−−)
        {
                if (sval[ii] == ch)
                        break;
        }
        if (!ii && sval[ii] != ch) // if (ii == 0)
                return −1;
        return ii;
}

// Imitate Java's lastIndexOf string function...
// Overloaded function
int String::lastIndexOf(char *str2, int startIndex = 0)
{
        verifyIndex(startIndex);
    char *tok = NULL;
    int res = −1;

    register char *tmpaa = strdup(sval);  // malloc here
    if (!tmpaa) // tmpaa == NULL
    {
        cerr << "\nMemory alloc failed in strdup in lastIndexOf()\n" << endl;
        exit(−1);
    }

    if (!startIndex) // if (startIndex == 0)
        startIndex = strlen(sval);
    else
        tmpaa[startIndex+1] = 0;

    for (int ii = 0; ii <= startIndex; ii++)
    {
        tok = strstr(& tmpaa[ii], str2);
        if (tok == NULL)
            break;
        else
        {
            res = (int) (tok − tmpaa);
                        debug_("res", res);
            ii = res; // jump to where it matched (+1 in for loop)
        }
    }
    free(tmpaa);
        debug_("res", res);
        debug_("indexOf", & sval[res]);

        return res;
}
```

```
// Imitate Java's lastIndexOf string function...
// Overloaded function
int String::lastIndexOf(String str2, int startIndex = 0)
{
        verifyIndex(startIndex);
    char *tok = NULL;
    int res = −1;

    register char *tmpaa = strdup(sval);  // malloc here
    if (!tmpaa) // tmpaa == NULL
    {
        cerr << "\nMemory alloc failed in strdup in lastIndexOf()\n" << endl;
        exit(−1);
    }

    if (!startIndex) // if (startIndex == 0)
        startIndex = strlen(sval);
    else
        tmpaa[startIndex+1] = 0;

    for (int ii = 0; ii <= startIndex; ii++)
    {
        tok = strstr(& tmpaa[ii], str2.sval);
        if (tok == NULL)
            break;
        else
        {
            res = (int) (tok − tmpaa);
                        debug_("res", res);
            ii = res; // jump to where it matched (+1 in for loop)
        }
    }
    free(tmpaa);
        debug_("res", res);
        debug_("indexOf", & sval[res]);

        return res;
}

// Imitate Java's substring string function...
// The startIndex specifies the beginning index, and endIndex specifies
// the stopping point. The string returned contains all the characters
// from the beginning index, up to, but not including, the ending index.
String String::substring(int startIndex, int endIndex = 0)
{
        String tmpstr = String(sval);
        tmpstr._substring(startIndex, endIndex);
        return tmpstr;
}

// Imitate Java's concat string function...
String String::concat(String str2)
{
        return (*this + str2);
}

// Imitate Java's concat string function...
// overloaded function
String String::concat(char *str2)
{
        return (*this + str2);
}
```

```
// Imitate Java's replace string function...
// Replace all occurences of string 'original' with
// 'replacement' in 'sval'
String String::replace(char original, char replacement)
{
        // For example −
        //              replace('A', 'B') on sval = "some AAA and AAACC"
        //              reurns sval = "some BBB and BBBCC"
        //String *tmpstr = new String(sval); Use default copy cstr
        String tmpstr(sval);
        for (int ii = 0, len = strlen(sval); ii < len; ii++)
        {
                if (tmpstr.sval[ii] == original)
                        tmpstr.sval[ii] = replacement;
        }
        return tmpstr; // this will use copy constructor to make a default copy
}

// Imitate Java's replace string function...
// overloaded function
// Replace all occurences of string 'original' with
// 'replacement' in 'sval'
String String::replace(char *original, char *replacement)
{
        char *tok = NULL, *bb;
        register char *aa = strdup(sval);
        int lenrepl = strlen(replacement);

        // Allocate space for bb
        { // local scope
                int tmpii = 0;
                for (int ii = 0; ;ii++)
                {
                        tok = strstr(& aa[ii], original);
                        if (tok == NULL)
                                break;
                        else
                        {
                                ii = ii + (int) (tok −aa);
                                tmpii++;
                        }
                }
                if (!tmpii) // tmpii == 0, no match of 'original' found
                        return (String(sval)); // return original string
                tmpii = strlen(sval) + (tmpii * lenrepl) + 20;
                debug_("strstr tmpii", tmpii );
                bb = (char *) malloc(tmpii);
                memset(bb, 0, tmpii);
        }

        for (int res = −1; ;)
        {
                debug_("aa", aa);
                tok = strstr(aa, original);
                if (tok == NULL)
                {
                        strcat(bb, aa);
                        break;
                }
                else
                {
```

```
                        res = (int) (tok − aa);
                        strncat(bb, aa, res);
                        strcat(bb, replacement);
                        //bb[strlen(bb)] = 0;
                        debug_("res", res );
                        debug_("bb", bb );
                        strcpy(aa, & aa[res+lenrepl]);
                }
        }
        debug_("bb", bb );
        free(aa);
        String tmpstr(bb);
        free(bb);
        return tmpstr;
}
/*
another method of doing replace function but slow..
String String::replace(char *original, char *replacement)
{
        // For example −
        //              replace("AAA", "BB") on sval = "some AAA and AAACC"
        //              reurns sval = "some BB and BBCC"
        String bb(this−>before(original).sval);
        if (strlen(bb.sval) == 0)
                return String(sval); // return original string
        bb += replacement;

        String tmpaa(this−>sval), cc, dd;
        for (;;)
        {
                cc = tmpaa.after(original).sval;
                debug_("cc", cc.sval );
                if (!strlen(cc.sval)) // if (strlen(cc.sval) == 0)
                        break;

                dd = cc.before(original).sval;
                if (strlen(dd.sval) == 0)
                {
                        bb += cc;
                        break;
                }
                else
                {
                        bb += dd;
                        bb += replacement;
                }
                tmpaa = cc;
        }
        debug_("bb.sval", bb.sval );
        return bb;
}
*/

// Imitate Java's replace function −  StringBuffer
String String::replace (int startIndex, int endIndex, String str)
{
        verifyIndex(startIndex);
        verifyIndex(endIndex);
        int tmpjj = strlen(str.sval);
        if (tmpjj == 0)
                return *this;
        int tmpii = endIndex−startIndex−1;
```

```
        if (tmpjj < tmpii) // length of str is less than specified indexes.
                tmpii = tmpjj;
        debug_("sval", sval);
        debug_("str.sval", str.sval);
        strncpy(& sval[startIndex], str.sval, tmpii);
        sval[startIndex+tmpii] = 0;
        debug_("sval", sval);
        return *this;
}

// Imitate Java's trim string function...
String String::trim()
{
        //String *tmpstr = new String(sval);
        String tmpstr(sval);
        tmpstr._trim();
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr; // this will use copy constructor to make a default copy
}

// Imitate Java's insert string function...
String String::insert(int index, String str2)
{
        String tmpstr(this->insert(str2.sval, index).sval);
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr;
}

// Imitate Java's insert string function...
String String::insert(int index, char ch)
{
        char aa[2];
        aa[0] = ch;
        aa[1] = 0;
        String tmpstr(this->insert(aa, index).sval);
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr;
}

// Imitate Java's deleteCharAt string function...
String String::deleteCharAt(int loc)
{
        String tmpstr(sval);
        tmpstr._deleteCharAt(loc);
        return tmpstr;
}

// Imitate Java's delete string function...
// Note: -->Java name is "delete()", cannot use reserved name delete() in C++
// The startIndex specifies the index of the first character to remove,
// and endIndex specifies an index one past the last character to remove.
// Thus, the substring deleted runs from startIndex to (endIndex - 1)
String String::deleteStr(int startIndex, int endIndex)
{
        // For example -
        //      deleteStr(3,3) on val = 'pokemon' returns 'poon'
        String tmpstr(sval);
        tmpstr._deleteStr(startIndex, endIndex);
        return tmpstr;
}

// Imitate Java's reverse string function...
```

```
String String::reverse()
{
        // For example −
        //              reverse() on "12345" returns "54321"
        String tmpstr(sval);
        tmpstr._reverse();
        return tmpstr;
}

// Imitate Java's valueOf string function...
String String::valueOf(char chars[], int startIndex, int numChars)
{
        verifyIndex(startIndex);
        int ii = strlen(chars);
        if (startIndex > ii)
        {
                cerr << "\nvalueOf() − startIndex greater than string length of"
                        << "string passed" << endl;
                exit(0);
        }
        if ( (numChars+startIndex) > ii)
        {
                cerr << "\nvalueOf() − numChars exceeds the string length of"
                        << "string passed" << endl;
                exit(0);
        }

        char *aa = strdup(chars);
        aa[startIndex + numChars] = 0;
        String tmpstr(& aa[startIndex]);
        free(aa);
        return tmpstr;
}

// Imitate Java's ensureCapacity string function...
// For use by StringBuffer class.
// Pre−allocate room for certain number of chars, useful
// if you know in advance that you will be appending a large
// number of small strings to StringBuffer
void String::ensureCapacity(int capacity)
{
        sval = (char *) my_realloc(sval, capacity);
        sval[0] = '\0';
        debug_("In ensureCapacity(int capacity) sval", sval);
}

// Imitate Java's setLength string function...
// For use by StringBuffer class.
void String::setLength(int len)
{
        sval = (char *) my_realloc(sval, len);
        sval[0] = '\0';
        debug_("In ensureCapacity(int len) sval", sval);
}

// Imitate Java's setCharAt function −  StringBuffer
void String::setCharAt(int where, char ch)
{
        verifyIndex(where);
        sval[where] = ch;
        debug_("in StringBuffer dstr()", "ok");
}
```

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

// overloaded function − directly changes object
// Variable dummy will give different signature to function
void String::substring(int startIndex, int endIndex, bool dummy)
{
        this−>_substring(startIndex, endIndex);
}

inline void String::_substring(int startIndex, int endIndex)
{
        verifyIndex(startIndex);
        verifyIndex(endIndex);
        if (!endIndex) // endIndex == 0
                strcpy(sval, & sval[startIndex] ) ;
        else
        {
                if (endIndex > startIndex)
                {
                        strcpy(sval, & sval[startIndex] ) ;
                        sval[endIndex −startIndex] = 0;
                }
                else
                {
                        cerr << "\n_substring() − startIndex is greater than endIndex!!\n"
                                << endl;
                        exit(−1);
                }
        }
}

// overloaded function − directly changes object
String String::deleteStr(int startIndex, int endIndex, bool dummy)
{
        this−>_deleteStr(startIndex, endIndex);
        return *this;
}

inline void String::_deleteStr(int startIndex, int endIndex)
{
        verifyIndex(startIndex);
        verifyIndex(endIndex);
        // For example −
        //      deleteStr(3,3) on val = 'pokemon' returns 'poon'
        char *tmpaa = strdup(sval); // malloc here
        strcpy(& tmpaa[startIndex], & tmpaa[endIndex]);
        *this = tmpaa;
        free(tmpaa);
}

// overloaded function − directly changes object
String String::deleteCharAt(int loc, bool dummy)
{
        this−>_deleteCharAt(loc);
        return *this;
}

inline void String::_deleteCharAt(int loc)
{
        char *tmpaa = strdup(sval); // malloc here
        strcpy(& tmpaa[loc], & tmpaa[loc+1]);
```

```
        *this = tmpaa;
        free(tmpaa);
}

// Returns string before regx. Matches first occurence of regx
String String::at(char *regx)
{
        char *tok = NULL;
        tok = strstr(sval, regx);
        if (tok == NULL)
                return(String(""));
        else
        {
                int res = (int) (tok − sval);
                char *lefttok = strdup(sval);
                memset(lefttok, 0, strlen(sval));
                strcpy(lefttok, & sval[res]);
                String tmpstr(lefttok);
                free(lefttok);
                return(tmpstr);
        }
}

// Returns string before regx. Matches first occurence of regx
String String::before(char *regx)
{
        char *tok = NULL;
        tok = strstr(sval, regx);
        if (tok == NULL)
                return(String(""));
        else
        {
                int res = (int) (tok − sval);
                char *lefttok = strdup(sval);
                lefttok[res] = 0;
                String tmpstr(lefttok);
                free(lefttok);
                return(tmpstr);
        }
}

// Returns string after regx. Matches first occurence of regx
String String::after(char *regx)
{
        char *tok = NULL;
        tok = strstr(sval, regx);
        if (tok == NULL)
                return(String(""));
        else
        {
                int res = (int) (tok − sval);
                char *lefttok = strdup(sval);
                memset(lefttok, 0, strlen(sval));
                strcpy(lefttok, & sval[res + strlen(regx)]);
                String tmpstr(lefttok);
                free(lefttok);
                return(tmpstr);
        }
}

// Explodes the string and returns the list in
// the list−head pointer explodeH
```

```
// See also token()
void String::explode(char *seperator)
{
        char *aa = NULL, *bb = NULL;
        aa = (char *) my_malloc(strlen(sval));
        for (bb = strtok(aa, seperator); bb != NULL; bb = strtok(NULL, seperator) )
        {
                String *tmp = new String(bb);
                String::explodeH.insert(String::explodeH.end(), *tmp);
        }
        my_free(aa);

        list<String>::iterator iter1; // see file include/g++/stl_list.h
        debug_("Before checking explode..", "ok");
        if (String::explodeH.empty() == true )
        {
                debug_("List is empty!!", "ok");
        }

        for (iter1 = String::explodeH.begin(); iter1 != String::explodeH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        debug_("Iterator iter1 is NULL!!", "ok" );
                        break;
                }
                debug_("(*iter1).sval", (*iter1).sval);
        }
}

// Overloaded function of explode(). This will return an
// array of strings and total number in strcount reference
// variable.
// See also token()
String *String::explode(int & strcount, char seperator = ' ')
{
        String aa(sval);
        aa.trim(true);
        strcount = 0;
        for (int ii = 0, jj = strlen(aa.sval); ii < jj; ii++)
        {
                if (aa.sval[ii] == seperator)
                        strcount++;
        }

        String *tmpstr = new String[strcount+1];
        if (!strcount) // strcount == 0
                tmpstr[0] = aa.sval;
        else
        {
                for (int ii = 0; ii <= strcount; ii++)
                        tmpstr[ii] = aa.token();
        }
        return tmpstr;
}

// Implodes the strings in the list−head
// pointer explodeH and returns the String class
void String::implode(char *glue)
{
}
```

```
// Joins the strings in the list−head
// pointer explodeH and returns the String class
void String::join(char *glue)
{
        implode(glue);
}

// Repeat the input string n times
String String::repeat(char *input, unsigned int multiplier)
{
        // For example −
        // repeat("k", 4) returns "kkkk"
        if (!input) // input == NULL
        {
                return (String(""));
        }

        char *aa = (char *) my_malloc(strlen(input) * multiplier);
        for (unsigned int tmpii = 0; tmpii < multiplier; tmpii++)
        {
                strcat(aa, input);
        }
        String tmpstr(aa);
        my_free(aa);
        return tmpstr;
}

// Reverse the string
// Overloaded version of reverse(). This will directly
// change the object.
void String::reverse(bool dummy)
{
        this−>_reverse();
}
inline void String::_reverse()
{
        // For example −
        //              reverse() on "12345" returns "54321"
        char aa;
        unsigned long tot_len = strlen(sval);
        unsigned long midpoint = tot_len / 2;
        for (unsigned long tmpjj = 0; tmpjj < midpoint; tmpjj++)
        {
                aa = sval[tmpjj];  // temporary storage var
                sval[tmpjj] = sval[tot_len − tmpjj − 1];  // swap the values
                sval[tot_len − tmpjj − 1] = aa; // swap the values
        }
}

// Translate certain chars
// For e.g ("abcd", "ABC") translates all occurences of each
// character in 'from' to corresponding character in 'to'
String String::tr(char *from, char *to)
{
        int lenfrom = strlen(from), lento = strlen(to);
        if (lento > lenfrom)
                lento = lenfrom; // set it to least
        else
        if (lento < lenfrom)
                lenfrom = lento; // set it to least
        debug_("lento", lento);
```

```
        register char *aa = strdup(sval);
        for (int ii = 0, jj = strlen(sval); ii < jj; ii++) // for every char in val
        {
                for (int kk = 0; kk < lento; kk++) // for every char in "from" string
                {
                        if (aa[ii] == from[kk])
                                aa[ii] = to[kk];
                }
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// Center the text
String String::center(int padlength, char padchar = ' ')
{
        // For example −
        //              center(10, '*') on sval="aa" returns "****aa****"
        //              center(10) on sval="aa" returns "    aa    "
        // The result is a string of 'padlength' characters with sval centered in it.
        int tmpii = sizeof(char) * (padlength + strlen(sval) + 10);
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);

        for (int jj = 0, kk = (int) padlength/2; jj < kk; jj++)
        {
                aa[jj] = padchar;
        }
        strcat(aa, sval);
        for (int jj = strlen(aa), kk = jj + (int) padlength/2; jj < kk; jj++)
        {
                aa[jj] = padchar;
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// Formats the original string by placing <number> of <padchar> characters
// between each set of blank−delimited words. Leading and Trailing blanks
// are always removed. If <number> is omitted or is 0, then all spaces are
// in the string are removed. The default number is 0 and
// default padchar ' '
String String::space(int number, char padchar = ' ')
{
        // For example −
        //              space(3) on sval = "I do not know"
        //                              will return "I   do   not   know"
        //              space(1, '_') on sval = "A deep black space"
        //                              will return "A_deep_black_space"
        //              space() on sval = "I   know     this"
        //                              will return "Iknowthis"

        debug_("this−>sval", this−>sval );
        String tmpstr = this−>trim().sval;
        debug_("tmpstr.sval", tmpstr.sval );

        // count spaces
        int spacecount = 0;
        for (int ii = 0, jj = strlen(tmpstr.sval); ii < jj; ii++)
        {
```

```
                if (tmpstr.sval[ii] == ' ')
                        spacecount++;
        }
        debug_("spacecount", spacecount);

        char ee[2];
        ee[0] = padchar;
        ee[1] = 0;
        String bb = tmpstr.repeat(ee, spacecount);

        int tmpii = sizeof(char) * (strlen(tmpstr.sval) + (number * spacecount) + 20);
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
        for (int ii = 0, jj = strlen(tmpstr.sval); ii < jj; ii++)
        {
                if (tmpstr.sval[ii] == ' ')
                        strcat(aa, bb.sval);
                else
                {
                        ee[0] = sval[ii];
                        strcat(aa, ee);
                }
        }
        tmpstr = aa;
        free(aa);
        return tmpstr;
}

// The result is string comprised of all characters between
// and including <start> and <end>
String String::xrange(char start, char end)
{
        // For example −
        //      xrange('a', 'j') returns val = "abcdefghij"
        //      xrange(1, 8) returns val = "12345678"

        if (end < start)
        {
                cerr << "\nThe 'end' character is less than 'start' !!" << endl;
                return String("");
        }

        // Note: The 'end' is greater than 'start'!! And add +1
        int tmpii = sizeof(char) * (end − start + 11);
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
        debug_("xrange tmpii", tmpii);
        for (int ii = start, jj = 0; ii <= end; ii++, jj++)
        {
                aa[jj] = ii;
                debug_("xrange aa[jj]", aa[jj] );
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// Removes any characters contained in <list>. The default character
// for <list> is a blank ' '
String String::compress(char *list = " ")
{
        // For example −
```

```
        //      compress("$,%") on sval = "$1,934" returns "1934"
        //      compress() on sval = "call me alavoor vasudevan" returns "callmealavoorvasudevan"
        int lenlist = strlen(list);
        register char *aa = strdup(sval);
        for (int ii = 0, jj = strlen(sval); ii < jj; ii++) // for every char in sval
        {
                for (int kk = 0; kk < lenlist; kk++) // for every char in "from" string
                {
                        if (aa[ii] == list[kk])
                        {
                                strcpy(& aa[ii], & aa[ii+1]);
                        }
                }
        }
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// The <newstr> is inserted into sval beginning at <start>. The <newstr> will
// be padded or truncated to <length> characters. The default <length> is
// string length of newstr
String String::insert(char *newstr, int start = 0, int lengthstr = 0, char padchar = ' ')
{
        // For example −
        //      insert("something new", 4, 20, '*') on sval = "old thing"
        //              returns "old something new*******thing"
        int tmplen = sizeof(char) * strlen(sval) + strlen(newstr) + lengthstr + 10;
        char *tmpaa = (char *) malloc (tmplen);
        memset(tmpaa, 0, tmplen);
        if (!start) // start == 0
        {
                strcpy(tmpaa, newstr);
                strcat(tmpaa, this->sval);
        }
        else
        {
                strncpy(tmpaa, this->sval, start);
                strcat(tmpaa, newstr);
                strcat(tmpaa, & this->sval[start]);
        }

        String tmpstr(tmpaa);
        free(tmpaa);
        return tmpstr;
}

// overloaded insert function...
String String::insert(int index, String str2, bool dummy)
{
        *this =  this->insert(str2.sval, index).sval;
        //debug_("tmpstr.sval", tmpstr.sval);
        return *this;
}

// overloaded insert function...
String String::insert(int index, char ch, bool dummy)
{
        char aa[2];
        aa[0] = ch;
        aa[1] = 0;
        *this = this->insert(aa, index).sval;
```

```
        //debug_("tmpstr.sval", tmpstr.sval);
        return *this;
}

// The result is string of <length> chars madeup of leftmost chars in sval.
// Quick way to left justify a string.
String String::left(int slength = 0, char padchar = ' ')
{
        // For example −
        //      left(15) on sval = "Wig"       returns "Wig           "
        //      left(4) on  sval = "Wighat"    returns "Wigh"
        //      left() on   sval = "   Wighat" returns "Wighat   "
        if (!slength) // slength == 0
                slength = strlen(sval);
        debug_("left() slength", slength);

        int tmpii = slength + 20;
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);
        debug_("this->ltrim().sval ", this->ltrim().sval);
        strcpy(aa, this->ltrim().sval);
        debug_("left() aa", aa );

        int currlen = strlen(aa);
        if (currlen < slength)
        {
                // pad the string now
                char ee[2];
                ee[0] = padchar;
                ee[1] = 0;
                strcat(aa, this->repeat(ee, (unsigned int) (slength-currlen) ).sval);
        }
        else
        {
                aa[slength] = 0;
        }

        debug_("left() aa", aa );
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// The result is string of <length> chars madeup of rightmost chars in sval.
// Quick way to right justify a string.
String String::right(int slength = 0, char padchar = ' ')
{
        // For example −
        //      right(10) on sval = "never to saying   " returns " to saying"
        //      right(4) on  sval = "Wighat"             returns "ghat"
        //      right(8) on  sval = "4.50"               returns "    4.50"
        //      right() on   sval = "  4.50     "        returns "       4.50"

        if (!slength) // slength == 0
                slength = strlen(sval);
        debug_("right() slength", slength);

        int tmpii = slength + 20;
        char *aa = (char *) malloc(tmpii);
        memset(aa, 0, tmpii);

        int currlen = strlen(this->rtrim().sval);
```

```
        debug_("right() currlen", currlen );
        if (currlen < slength)
        {
                // pad the string now
                char ee[2];
                ee[0] = padchar;
                ee[1] = 0;
                strcpy(aa, this->repeat(ee, (unsigned int) (slength-currlen) ).sval);
                strcat(aa, this->rtrim().sval);
                debug_("right() aa", aa );
        }
        else
        {
                strcpy(aa, this->rtrim().sval);
                strcpy(aa, & aa[currlen-slength]);
                aa[slength] = 0;
        }

        debug_("right() aa", aa );
        String tmpstr(aa);
        free(aa);
        return tmpstr;
}

// The <newstr> is overlayed into sval beginning at <start>. The <newstr> will
// be padded or truncated to <length> characters. The default <length> is
// string length of newstr
String String::overlay(char *newstr, int start = 0, int slength = 0, char padchar = ' ')
{
        // For example -
        //      overlay("12345678", 4, 10, '*') on sval = "oldthing is very bad"
        //              returns "old12345678**ery bad"
        //      overlay("12345678", 4, 5, '*') on sval = "oldthing is very bad"
        //              returns "old12345ery bad"
        int len_newstr = strlen(newstr);
        if (!slength) // slength == 0
                slength = len_newstr;
        char *aa = (char *) malloc(slength + len_newstr + 10);
        aa[0] = 0;
        char ee[2];
        ee[0] = padchar;
        ee[1] = 0;
        if (len_newstr < slength)
        {
                // pad it now
                strcpy(aa, newstr);
                strcat(aa, this->repeat(ee, (slength-len_newstr)).sval );
        }
        else
        {
                strcpy(aa, newstr);
                aa[slength] = 0;
        }

        // Now overlay the string.
        String tmpstr(sval);

        debug_("tmpstr.sval", tmpstr.sval);
        for (int ii=start, jj=strlen(tmpstr.sval), kk=start+slength, mm=0;
                                ii < jj; ii++, mm++)
        {
                if (ii == kk)
```

```
                        break;
                if (mm == slength)
                        break;
                tmpstr.sval[ii] = aa[mm];
        }
        free(aa);
        debug_("tmpstr.sval", tmpstr.sval);
        return tmpstr;
}

// If string is literrally equal to .. or not equal to
// If type is false than it is ==
bool String::_equalto(const String & rhs, bool type = false)
{
        if (type == false) // test for ==
        {
                if (strlen(rhs.sval) == strlen(sval))
                {
                        if (!strncmp(rhs.sval, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
        else // test for !=
        {
                if (strlen(rhs.sval) != strlen(sval))
                {
                        if (!strncmp(rhs.sval, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
}

// If string is literrally equal to .. or not equal to
// If type is false than it is ==
bool String::_equalto(const char *rhs, bool type = false)
{
        if (type == false) // test for ==
        {
                if (strlen(rhs) == strlen(sval))
                {
                        if (!strncmp(rhs, sval, strlen(sval))) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
        else // test for !=
        {
                if (strlen(rhs) != strlen(sval))
                {
                        if (!strncmp(rhs, sval, strlen(sval))) //  == 0
                                return true;
```

```
                        else
                                return false;
                }
                else
                        return false;
        }
}

// Synonym function is vacuum()
void String::clear()
{
        sval = (char *) my_realloc(sval, 10);
        sval[0] = '\0';
}

// Remove trailing ALL given character 'ch' − see also chop()
// For example :
//      sval = "abcdef\n\n\n" then chopall() = "abcdef"
//      sval = "abcdeffffff" then chopall('f') = "abcde"
void String::chopall(char ch='\n')
{
        unsigned long tmpii = strlen(sval) − 1 ;
        for (; tmpii >= 0; tmpii−−)
        {
                if (sval[tmpii] == ch)
                        sval[tmpii] = 0;
                else
                        break;
        }
}

// Remove trailing character − see also chopall()
// chop() is often used to remove trailing newline character
void String::chop()
{
        sval[strlen(sval)−1] = 0;
}

// Overloaded version of trim(). This will directly
// change the object.
void String::trim(bool dummy)
{
        this−>_trim();
}

inline void String::_trim()
{
        this−>rtrim(true);
        this−>ltrim(true);
        debug_("this−>sval", this−>sval);
}

// Overloaded version of ltrim(). This will directly
// change the object.
void String::ltrim(bool dummy)
{
        this−>_ltrim();
}

inline void String::_ltrim()
{
        // May cause problems in my_realloc since
```

```
        // location of bb will be destroyed !!
        char *bb = sval;

        if (bb == NULL)
                return;

        while (isspace(*bb))
                bb++;
        debug_("bb", bb);

        if (bb != NULL && bb != sval)
        {
                debug_("doing string copy", "done");
                _str_cpy(bb); // causes problems in my_realloc and bb is getting destroyed!!
        }
        else
                debug_("Not doing string copy", "done");
}

String String::ltrim()
{
        String tmpstr(sval);
        tmpstr._ltrim();
        return tmpstr;
}

// Overloaded version of rtrim(). This will directly
// change the object.
void String::rtrim(bool dummy)
{
        this−>_rtrim();
}

inline void String::_rtrim()
{
        for (long tmpii = strlen(sval) − 1 ; tmpii >= 0; tmpii−−)
        {
                if ( isspace(sval[tmpii]) )
                        sval[tmpii] = '\0';
                else
                        break;
        }
}

String String::rtrim()
{
        String tmpstr(sval);
        tmpstr._rtrim();
        return tmpstr;
}

// Use for rounding off fractions digits of floats
// Rounds−off floats with given precision and then
// stores the result into String's sval field
// Also returns the result as a char *
void String::roundf(float input_val, short precision)
{
        float   integ_flt, deci_flt;
        const   short MAX_PREC = 4;

        debug_("In roundf", "ok");
```

```
        if (precision > MAX_PREC) // this is the max reliable precision
                precision = MAX_PREC;

        // get the integral and decimal parts of the float value..
        deci_flt = modff(input_val, & integ_flt);

        for (int tmpzz = 0; tmpzz < precision; tmpzz++)
        {
                debug_("deci_flt", deci_flt);
                deci_flt *= 10;
        }
        debug_("deci_flt", deci_flt);

        unsigned long deci_int = (unsigned long) ( rint(deci_flt) );

        sval = (char *) my_malloc(NUMBER_LENGTH); // float 70 digits max

        if (deci_int > 999) // (MAX_PREC) digits
                sprintf(sval, "%lu.%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 99) // (MAX_PREC − 1) digits
                sprintf(sval, "%lu.0%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9) // (MAX_PREC − 2) digits
                sprintf(sval, "%lu.00%lu", (unsigned long) integ_flt, deci_int);
        else
                sprintf(sval, "%lu.00000%lu", (unsigned long) integ_flt, deci_int);
}

void String::roundd(double input_val, short precision)
{
        double  integ_flt, deci_flt;
        const   short MAX_PREC = 6;

        if (precision > MAX_PREC) // this is the max reliable precision
                precision = MAX_PREC;

        debug_("In roundd", "ok");
        // get the integral and decimal parts of the double value..
        deci_flt = modf(input_val, & integ_flt);

        for (int tmpzz = 0; tmpzz < precision; tmpzz++)
        {
                debug_("deci_flt", deci_flt);
                deci_flt *= 10;
        }
        debug_("deci_flt", deci_flt);

        sval = (char *) my_malloc(NUMBER_LENGTH); // double 70 digits max

        unsigned long deci_int = (unsigned long) ( rint(deci_flt) );

        if (deci_int > 99999) // (MAX_PREC) digits
                sprintf(sval, "%lu.%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9999) // (MAX_PREC − 1) digits
                sprintf(sval, "%lu.0%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 999) // (MAX_PREC − 2) digits
                sprintf(sval, "%lu.00%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 99) // (MAX_PREC − 3) digits
```

```
                sprintf(sval, "%lu.000%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9) // (MAX_PREC − 4) digits
                sprintf(sval, "%lu.0000%lu", (unsigned long) integ_flt, deci_int);
        else // (MAX_PREC − 5) digits
                sprintf(sval, "%lu.00000%lu", (unsigned long) integ_flt, deci_int);
}

// Provided for documentation purpose only
// You must use the function indexOf()
bool String::contains(char *str2, int startIndex = 0)
{
        // For example −
        //              if (indexOf("ohboy") > −1 )
        //                      cout << "\nString contains 'ohboy'" << endl;
        //              if (indexOf("ohboy") < 0 )
        //                      cout << "\nString does NOT contain 'ohboy'" << endl;
        //              if (indexOf("ohboy", 4) > −1 )
        //                      cout << "\nString contains 'ohboy'" << endl;
        //              if (indexOf("ohboy", 4) < 0 )
        //                      cout << "\nString does NOT contain 'ohboy'" << endl;
        cerr << "\nYou must use indexOf() function instead of contains()\n" << endl;
        exit(−1);
}

// Synonym function is empty()
bool String::isNull()
{
        if (sval[0] == '\0')
                return true;
        else
        {
                if (sval == NULL)
                        return true;
                else
                        return false;
        }
}

// Leading, trailing white−spaces of string are ignored
bool String::isInteger()
{
        String tmpstr(sval);
        tmpstr.trim(true);
        debug_("tmpstr.sval", tmpstr.sval );
        if ( strspn ( tmpstr.sval, "0123456789" ) != strlen(tmpstr.sval) )
                return ( false ) ;
        else
                return ( true ) ;
}

// overloaded func
bool String::isInteger(int pos)
{
        verifyIndex(pos);
        return (isdigit(sval[pos]));
}

// Leading, trailing white−spaces of string are ignored
bool String::isNumeric()
{
        String tmpstr(sval);
```

```
        tmpstr.trim(true);
        debug_("tmpstr.sval", tmpstr.sval );
        if ( strspn ( tmpstr.sval, "0123456789.+−e" ) != strlen(tmpstr.sval) )
                return ( false ) ;
        else
                return ( true ) ;
}

// overloaded func
bool String::isNumeric(int pos)
{
        verifyIndex(pos);
        return (isdigit(sval[pos]));
}

bool String::isEmpty()
{
        if (strlen(sval) == 0)
                return true;
        else
                return false;
}

// See also explode()
//      Warning : The String instance is modified by removing
//                      the returned token from the string. It is advised
//                          that you save the original string before calling
//                          this function like for example :
//              String savestr = origstr;
//              String aa, bb, cc;
//              aa = origstr.token();
//              bb = origstr.token();
//              cc = origstr.token();
//
//  This routine returns the first non−'seperator' (default
//  white−space) token string from the String instance
String String::token(char seperator = ' ')
{
        char ee[2];
        ee[0] = seperator;
        ee[1] = 0;
        char *res = strtok(sval, ee);
        if (!res) // if res == NULL
        {
                debug_("token", res);
                debug_("sval", sval);
                return(String(sval));
        }
        else
        {
                String tmpstr(res);

                // Should take string length of sval and not res
                // because strtok() had put a NULL ('\0') at the location
                // and also strtok() ignores the leading blanks of sval
                strcpy(sval, & sval[strlen(sval)+1]);
                debug_("token", res);
                debug_("sval", sval);
                return tmpstr;
        }
}
```

```
String String::crypt(char *original, char *salt)
{
        return String("");
}

int String::int_value()
{
        if ( strlen(sval) == 0 ) {
                cerr << "Cannot convert a zero length string "
                << " to a numeric" << endl ;
                abort() ;
        }

        if ( ! isInteger() ) {
                cerr << "Cannot convert string [" << sval
                << "] to an integer numeric string" << endl ;
                abort() ;
        }

        return ( atoi ( sval ) ) ;
}

double String::double_value()
{
        if ( strlen(sval) == 0 ) {
                cerr << "Cannot convert a zero length string "
                << " to a numeric" << endl ;
                abort() ;
        }

        if ( ! isNumeric() ) {
                cerr << "Cannot convert string [" << sval
                << "] to a double numeric string" << endl ;
                abort() ;
        }

        double d = atof ( sval ) ;

        return ( d ) ;
}

String String::getline(FILE *infp = stdin)
{
        register char ch, *aa = NULL;

        register const short SZ = 100;
        // Initial value of ii > SZ so that aa is alloc'ed memory
        register int jj = 0;
        for (int ii = SZ+1; (ch = getc(infp)) != EOF; ii++, jj++)
        {
                if (ii > SZ)  // allocate memory in steps of SZ for performance
                {
                        aa = (char *) realloc(aa, jj + ii + 15);  // +15 is safe mem
                        ii = 0;
                }
                if (ch == '\n')  // read untill newline is encountered
                        break;
                aa[jj] = ch;
        }
        aa[jj] = 0;
        _str_cpy(aa);  // puts the value in string
        free(aa);
```

```
        return *this;
}

/*
void String::Format(const char *fmt, ... )
{
        va_list iterator;
        va_start(iterator, fmt );
        va_end(iterator);
}
*/

inline void String::verifyIndex(unsigned long index) const
{
        if (index < 0 || index >= strlen(sval) )
        {
                // throw "Index Out Of Bounds Exception";
                cerr << "Index Out Of Bounds Exception at ["
                                << index << "] in:\n" << sval << endl;
                exit(1);
        }
}

////////////////////////////////////////////////////////
// Private functions start from here .........
////////////////////////////////////////////////////////
void String::_str_cpy(char bb[])
{
        debug_("In _str_cpy bb", bb);
        if (bb == NULL)
        {
                sval[0] = '\0';
                return;
        }

        unsigned long tmpii = strlen(bb);

        if (tmpii == 0)
        {
                sval[0] = '\0';
                return;
        }

        debug_("In _str_cpy tmpii", tmpii);
        debug_("In _str_cpy sval", sval);
        sval = (char *) my_realloc(sval, tmpii);
        //sval = new char [tmpii + SAFE_MEM_2];
        debug_("In _str_cpy bb", bb);

        strncpy(sval, bb, tmpii);
        debug_("In _str_cpy sval", sval);
        sval[tmpii] = '\0';
        debug_("In _str_cpy sval", sval);
}

void String::_str_cpy(int bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%d", bb);
        _str_cpy(tmpaa);
}
```

```
void String::_str_cpy(unsigned long bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%ld", bb);
        _str_cpy(tmpaa);
}

void String::_str_cpy(float bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%f", bb);
        _str_cpy(tmpaa);
}

void String::_str_cat(char bb[])
{
        unsigned long tmpjj = strlen(bb), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        debug_("sval in _str_cat() ", sval);
        strncat(sval, bb, tmpjj);
}

void String::_str_cat(int bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%d", bb);

        unsigned long tmpjj = strlen(tmpaa), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        strncat(sval, tmpaa, tmpjj);
}

void String::_str_cat(unsigned long bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%ld", bb);

        unsigned long tmpjj = strlen(tmpaa), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        strncat(sval, tmpaa, tmpjj);
}

void String::_str_cat(float bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%f", bb);

        unsigned long tmpjj = strlen(tmpaa), tmpii = strlen(sval);
        sval = (char *) my_realloc(sval, tmpii + tmpjj);
        strncat(sval, tmpaa, tmpjj);
}

//////////////////////////////////////////////////////
// All operator functions start from here .........
//////////////////////////////////////////////////////
String operator+ (const String & lhs, const String & rhs)
{
        /*****************************************************/
        // Note : For adding two char strings, first cast String
        // as in −
        //aa = (String) "alkja " + " 99djd " ;
        /*****************************************************/
```

```
        String tmp(lhs);
        tmp._str_cat(rhs.sval);
        return(tmp);

        /*
        if (String::_global_String == NULL)
        {
                String::_global_String = new String;
                String::_global_String->_str_cpy(lhs.sval);
                String::_global_String->_str_cat(rhs.sval);
                //return *String::_global_String;
                return String(String::_global_String->val);
        }
        */
        /*
        else
        if (String::_global_String1 == NULL)
        {
                debug_("1)global", "ok" );
                String::_global_String1 = new String;
                String::_global_String1->_str_cpy(lhs.sval);
                String::_global_String1->_str_cat(rhs.sval);
                return *String::_global_String1;
        }
        */
        /*
        else
        {
                fprintf(stderr, "\nError: cannot alloc _global_String\n");
                exit(-1);
        }
        */

        /*
        String *aa = new String;
        aa->_str_cpy(lhs.sval);
        aa->_str_cat(rhs.sval);
        return *aa;
        */
}

String String::operator+ (const String & rhs)
{
        String tmp(*this);
        tmp._str_cat(rhs.sval);
        debug_("rhs.sval in operator+", rhs.sval );
        debug_("tmp.sval in operator+", tmp.sval );
        return (tmp);
}

// Using reference will be faster in = operator
String& String:: operator= ( const String& rhs )
{
        if (& rhs == this)
        {
                debug_("Fatal Error: In operator(=). rhs is == to 'this pointer'!!", "ok" );
                return *this;
        }

        this->_str_cpy(rhs.sval);
        debug_("rhs value", rhs.sval );
```

```
        // Free global vars memory
        //_free_glob(& String::_global_String);
        //if (String::_global_String == NULL)
                //fprintf(stderr, "\n_global_String is freed!\n");

        //return (String(*this));
        return *this;
}

// Using reference will be faster in = operator
String& String::operator+= (const String & rhs)
{
        /*******************************************************/
        // Note : For adding two char strings, first cast String
        // as in −
        //aa += (String) "cccc" + "dddd";
        /*******************************************************/

        if (& rhs == this)
        {
                debug_("Fatal error: In operator+= rhs is equals 'this' ptr", "ok");
                return *this;
        }
        this−>_str_cat(rhs.sval);
        return *this;
        //return (String(*this));
}

bool String::operator== (const String & rhs)
{
        return(_equalto(rhs.sval));
}

bool String::operator== (const char *rhs)
{
        return(_equalto(rhs));
}

bool String::operator!= (const String & rhs)
{
        return(_equalto(rhs.sval, true));
}

bool String::operator!= (const char *rhs)
{
        return(_equalto(rhs, true));
}

char String::operator[] (unsigned long Index) const
{
        verifyIndex(Index);
        return sval[Index];
}

char & String::operator[] (unsigned long Index)
{
        verifyIndex(Index);
        return sval[Index];
}

istream & operator >> (istream & In, String & str2)
```

```
{
        // allocate max size of 2048 characters
        static char aa[MAX_ISTREAM_SIZE];

        In >> aa;
        str2 = aa; // assign to reference String
        return In; // return istream
}

ostream & operator << (ostream & Out, const String & str2)
{
        Out << str2.sval;
        return Out;
}

////////////////////////////////////////////////
//  Imitate Java's StringBuffer Object
//      StringBuffer class functions
////////////////////////////////////////////////

// Imitate Java's StringBuffer − the default constructor
// (the one with no parameters) reserves room for 16
// characters.
StringBuffer::StringBuffer()
        :String()   // calls base class cstr no params
{
        debug_("in StringBuffer cstr()", "ok");
}

// Imitate Java's StringBuffer
StringBuffer::StringBuffer(int size)
        :String(size, true)  // calls base class cstr
{
        // String(size, true) −− do not call here in body but call
        // in initialization stage which will avoid extra call on
        // default base−class constructor and will be faster
        // and very efficient
        debug_("in StringBuffer cstr(int size)", "ok");
}

// Imitate Java's StringBuffer
// calls base class cstr with string param
StringBuffer::StringBuffer(String str)
        :String(str.val())  // calls base class cstr
{
        // String(str.val()) −− do not call here in body but call
        // in initialization stage which will avoid extra call on
        // default base−class constructor and will be faster
        // and very efficient
        debug_("in StringBuffer cstr(String str)", "ok");
}

// Imitate Java's StringBuffer
StringBuffer::~StringBuffer()
{
        debug_("in StringBuffer dstr()", "ok");
}
```

16. Appendix D my_malloc.cpp

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

```
//*************************************************************
// Copyright policy is GNU/GPL and it is requested that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
//*************************************************************

/*
**      In your main() function put these lines −
                char p_name[1024];
                sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
                putenv(p_name);
                print_total_memsize(); // in the beginning
                ......
                ......
                print_total_memsize(); // in the end
*/

#include <stdio.h>
#include <alloc.h>  // for c++ −− malloc, alloc etc...
#include <stdlib.h>  // malloc, alloc..
#include <time.h>  // strftime, localtime, ...
#include <list.h>  // strftime, localtime, ...  see file include/g++/stl_list.h
//#include <debug.h> // debug_("a", a);  debug2_("a", a, true);

#include "my_malloc.h"

const short SAFE_MEM = 10;
const short DATE_MAX_SIZE = 200;

const short MALLOC = 1;
const short REALLOC     = 2;

const short VOID_TYPE =         1;
const short CHAR_TYPE =         2;
const short SHORT_TYPE =        3;
const short INT_TYPE =          4;
const short LONG_TYPE =         5;
const short FLOAT_TYPE =        6;
const short DOUBLE_TYPE =       7;

const char LOG_FILE[30] = "memory_error.log";

// Uncomment this line to debug total mem size allocated...
//#define DEBUG_MEM  "debug_memory_sizes_allocated"

static void raise_error_exit(short mtype, short datatype, char fname[], int lineno);

#ifdef DEBUG
class MemCheck
{
        public:
                MemCheck(void *aptr, size_t amem_size, char fname[], int lineno);
                void    *ptr;
                size_t  mem_size;
```

```
                static  list<MemCheck>              mcH;  // list head
                static  unsigned long           total_memsize;  // total memory allocated
};

// Global variables ....
list<MemCheck>              MemCheck::mcH;
unsigned long           MemCheck::total_memsize = 0;

MemCheck::MemCheck(void *aptr, size_t amem_size, char fname[], int lineno)
{
        char func_name[100];
        FILE    *ferr = NULL;
        sprintf(func_name, "MemCheck() − File: %s Line: %d", fname, lineno);

        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(−1);
                #else
                        return;
                #endif
        }

        // Search if the pointer already exists in the list...
        bool does_exist = false;
        list<MemCheck>::iterator iter1; // see file include/g++/stl_list.h
        //fprintf(ferr, "\n%s Before checking.. !!\n", func_name);
        if (MemCheck::mcH.empty() == true )
        {
                //fprintf(ferr, "\n%s List is empty!!\n", func_name);
        }
        for (iter1 = MemCheck::mcH.begin(); iter1 != MemCheck::mcH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        fprintf(ferr, "\n%s Iterator iter1 is NULL!!\n", func_name);
                        break;
                }
                if ( ((*iter1).ptr) == aptr)
                {
                        does_exist = true;
                        fprintf(ferr, "\n%s Already exists!!\n", func_name);
                        fprintf(ferr, "\n%s Fatal Error exiting now ....!!\n", func_name);
                        #ifdef DEBUG_MEM
                                exit(−1); //------------------------------------------------------
                        #else
                                return;
                        #endif
                        // Now change the mem size to new values...
                        // For total size − Remove old size and add new size
                        //fprintf(ferr, "\n%s total_memsize = %lu\n", func_name, (*iter1).total_m
                        //fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1).mem_size);
                        //fprintf(ferr, "\n%s amem_size = %u\n", func_name, amem_size);
                        (*iter1).total_memsize = (*iter1).total_memsize + amem_size;
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize − (*iter1
                                else
```

```
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*it
                                        fprintf(ferr, "\n%s mem_size = %u", func_name, (*iter1).m
                                        fprintf(ferr, "\n%s amem_size = %u\n", func_name, amem_si
                                }
                        }
                        (*iter1).mem_size = amem_size;
                }
        }

        // The pointer aptr does not exist in the list, so append it now...
        if (does_exist == false)
        {
                //fprintf(ferr, "\n%s aptr Not found\n", func_name);
                ptr = aptr;
                mem_size = amem_size;
                MemCheck::total_memsize += amem_size;
                MemCheck::mcH.insert(MemCheck::mcH.end(), *this);
        }
        fclose(ferr);
}

static inline void call_check(void *aa, size_t tmpii, char fname[], int lineno)
{
        MemCheck bb(aa, tmpii, fname, lineno);
        if (& bb);  // a dummy statement to avoid compiler warning msg.
}

static inline void remove_ptr(void *aa, char fname[], int lineno)
{
        char    func_name[100];
        if (aa == NULL)
                return;

        sprintf(func_name, "remove_ptr() − File: %s Line: %d", fname, lineno);
        FILE *ferr = NULL;
        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(−1);
                #else
                        return;
                #endif
        }

        bool does_exist = false;
        if (MemCheck::mcH.empty() == true)
        {
                //fprintf(ferr, "\n%s List is empty!!\n", func_name);
                //fclose(ferr);
                //return;
        }
        list<MemCheck>::iterator iter1; // see file include/g++/stl_list.h
        for (iter1 = MemCheck::mcH.begin(); iter1 != MemCheck::mcH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        fprintf(ferr, "\n%s Iterator iter1 is NULL!!\n", func_name);
```

```
                        break;
                }
                if ( ((*iter1).ptr) == aa)
                {
                        does_exist = true;
                        // Now change the mem size to new values...
                        // For total size − Remove old size
                        //fprintf(ferr, "\n%s total_memsize = %lu\n", func_name, (*iter1).total_m
                        //fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1).mem_size);
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize − (*iter1
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*it
                                        fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1)
                                }
                        }
                        MemCheck::mcH.erase(iter1);
                        break;  // must break to avoid infinite looping
                }
        }
        if (does_exist == false)
        {
                //fprintf(ferr, "\n%s Fatal Error: − You did not allocate memory!! \n", func_name
                //fprintf(ferr, "\n%s The value passed is %s\n", func_name, (char *) aa);
        }
        else
                //fprintf(ferr, "\n%s found\n", func_name);
        fclose(ferr);
}

static inline void call_free_check(void *aa, char *fname, int lineno)
{
        char func_name[100];
        sprintf(func_name, "call_free_check() − File: %s Line: %d", fname, lineno);

        FILE *ferr = NULL;
        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(−1);
                #else
                        return;
                #endif
        }

        bool does_exist = false;
        list<MemCheck>::iterator iter1; // see file include/g++/stl_list.h
        for (iter1 = MemCheck::mcH.begin(); iter1 != MemCheck::mcH.end(); iter1++)
        {
                if (iter1 == NULL)
                {
                        fprintf(ferr, "\n%s Iterator iter1 is NULL!!\n", func_name);
                        break;
                }
                if ( ((*iter1).ptr) == aa)
```

```
                {
                        does_exist = true;
                        //fprintf(ferr, "\n%s iter1.mem_size = %u\n", func_name, (*iter1).mem_siz
                        //fprintf(ferr, "\n%s Total memory allocated = %lu\n",  func_name, (*iter
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize − (*iter1
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*it
                                        fprintf(ferr, "\n%s mem_size = %u", func_name, (*iter1).m
                                }
                        }
                        MemCheck::mcH.erase(iter1);
                        break;  // must break to avoid infinite looping
                }
        }
        if (does_exist == false)
        {
                fprintf(ferr, "\n%s Fatal Error: free() − You did not allocate memory!!\n",
                                func_name);
                //fprintf(ferr, "\n%s The value passed is %s\n", func_name, (char *) aa);
                fclose(ferr);
                #ifdef DEBUG_MEM
                        exit(−1);
                #else
                        return;
                #endif
        }
        else
        {
                //fprintf(ferr, "\n%s found\n", func_name);
        }
        fclose(ferr);
}

void local_print_total_memsize(char *fname, int lineno)
{
        char func_name[100];
        sprintf(func_name, "local_print_total_memsize() − %s Line: %d", fname, lineno);

        FILE *ferr = NULL;
        ferr = fopen(LOG_FILE, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", LOG_FILE);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", LOG_FILE);
                #ifdef DEBUG_MEM
                        exit(−1);
                #else
                        return;
                #endif
        }

        fprintf(ferr, "\n%s Total memory MemCheck::total_memsize = %lu\n",  func_name, MemCheck::
        fclose(ferr);
}
#else  //------------> DEBUG

void local_print_total_memsize(char *fname, int lineno)
```

```
{
        // This function is available whether debug or no−debug...
}

#endif // DEBUG

void local_my_free(void *aa, char fname[], int lineno)
{
        if (aa == NULL)
                return;
        call_free_check(aa, fname, lineno);
        free(aa);
        aa = NULL;
}

// size_t is type−defed unsigned long
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;
}

// size_t is type−defed unsigned long
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;
}

// size_t is type−defed unsigned long
short *local_my_realloc(short *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (short) * (tmpqq);
        aa = (short *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // do not memset!! memset(aa, 0, tmpii);
```

```
        // Not for numbers!! aa[tmpqq−1] = 0;
        call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t is type−defed unsigned long
int *local_my_realloc(int *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (int) * (tmpqq);
        aa = (int *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // do not memset!! memset(aa, 0, tmpii);
        // Not for numbers!! aa[tmpqq−1] = 0;
        call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t is type−defed unsigned long
long *local_my_realloc(long *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (long) * (tmpqq);
        aa = (long *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // do not memset!! memset(aa, 0, tmpii);
        // Not for numbers!! aa[tmpqq−1] = 0;
        call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t is type−defed unsigned long
float *local_my_realloc(float *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (float) * (tmpqq);
        aa = (float *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // do not memset!! memset(aa, 0, tmpii);
        // Not for numbers!! aa[tmpqq−1] = 0;
        call_check(aa, tmpii, fname, lineno);
        return aa;
}

// size_t is type−defed unsigned long
double *local_my_realloc(double *aa, size_t size, char fname[], int lineno)
{
        remove_ptr(aa, fname, lineno);
        unsigned long tmpqq = size + SAFE_MEM;
        size_t  tmpii = sizeof (double) * (tmpqq);
        aa = (double *) realloc(aa, tmpii);
        if (aa == NULL)
                raise_error_exit(REALLOC, CHAR_TYPE, fname, lineno);
        // do not memset!! memset(aa, 0, tmpii);
        // Not for numbers!! aa[tmpqq−1] = 0;
        call_check(aa, tmpii, fname, lineno);
```

```
        return aa;
}

static void raise_error_exit(short mtype, short datatype, char fname[], int lineno)
{
        if (mtype == MALLOC)
        {
                fprintf(stdout, "\nFatal Error: malloc() failed!!");
                fprintf(stderr, "\nFatal Error: malloc() failed!!");
        }
        else
        if (mtype == REALLOC)
        {
                fprintf(stdout, "\nFatal Error: realloc() failed!!");
                fprintf(stderr, "\nFatal Error: realloc() failed!!");
        }
        else
        {
                fprintf(stdout, "\nFatal Error: mtype not supplied!!");
                fprintf(stderr, "\nFatal Error: mtype not supplied!!");
                exit(−1);
        }

        // Get current date−time and print time stamp in error file...
        char date_str[DATE_MAX_SIZE + SAFE_MEM];
        time_t tt;
        tt = time(NULL);
        struct tm *ct = NULL;
        ct = localtime(& tt); // time() in secs since Epoch 1 Jan 1970
        if (ct == NULL)
        {
                fprintf(stdout, "\nWarning: Could not find the local time, localtime() failed\n")
                fprintf(stderr, "\nWarning: Could not find the local time, localtime() failed\n")
        }
        else
                strftime(date_str, DATE_MAX_SIZE , "%C", ct);

        FILE *ferr = NULL;
        char    filename[100];
        strcpy(filename, LOG_FILE);
        ferr = fopen(filename, "a");
        if (ferr == NULL)
        {
                fprintf(stdout, "\nWarning: Cannot open file %s\n", filename);
                fprintf(stderr, "\nWarning: Cannot open file %s\n", filename);
        }
        else
        {
                // ***********************************************
                // ******* Do putenv in the main() function *********
                //              char p_name[1024];
                //              sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
                //              putenv(p_name);
                // ***********************************************
                char    program_name[200+SAFE_MEM];
                if (getenv("PROGRAM_NAME") == NULL)
                {
                        fprintf(ferr, "\n%sWarning: You did not putenv() PROGRAM_NAME env variabl
                                        date_str);
                        program_name[0] = 0;
                }
                else
```

```
                        strncpy(program_name, getenv("PROGRAM_NAME"), 200);

                if (mtype == MALLOC)
                        fprintf(ferr, "\n%s: %s − Fatal Error − my_malloc() failed.", date_str, p
                else
                if (mtype == REALLOC)
                {
                        fprintf(ferr, "\n%s: %s − Fatal Error − my_realloc() failed.", date_str,
                        char dtype[50];
                        switch(datatype)
                        {
                                case VOID_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                case CHAR_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                case SHORT_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                case INT_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                case LONG_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                case FLOAT_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                case DOUBLE_TYPE:
                                        strcpy(dtype, "char*");
                                        break;
                                default:
                                        strcpy(dtype, "none*");
                                        break;
                        }
                        fprintf(ferr, "\n%s %s − Fatal Error: %s realloc() failed!!", date_str, p
                }

                fprintf(ferr, "\n%s %s − Very severe error condition. Exiting application now....
                                date_str, program_name);
                fclose(ferr);
        }

        exit(−1);
}
```

---

17. Appendix E my_malloc.h

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

```
//*************************************************************
// Copyright policy is GNU/GPL and it is requested that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
```

```
//*************************************************************

/*
**      In your main() function put −
                char p_name[1024];
                sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
                putenv(p_name);
                print_total_memsize(); // in the beginning
                ......
                ......
                print_total_memsize(); // in the end
*/

/* Use zap instead of delete as this will be very clean!!
** Use do while to make it robust and bullet−proof macro
*/
#define zap(x) do { if (x) { delete(x); x = 0; } } while (0)

void *local_my_malloc(size_t size, char fname[], int lineno);

char *local_my_realloc(char *aa, size_t size, char fname[], int lineno);
short *local_my_realloc(short *aa, size_t size, char fname[], int lineno);
void local_my_free(void *aa, char fname[], int lineno);

void local_print_total_memsize(char fname[], int lineno);

#define my_free(NM) (void) (local_my_free(NM, __FILE__, __LINE__))
#define my_malloc(SZ) (local_my_malloc(SZ, __FILE__, __LINE__))
#define my_realloc(NM, SZ) (local_my_realloc(NM, SZ, __FILE__, __LINE__))
#define print_total_memsize() (void) (local_print_total_memsize(__FILE__, __LINE__))

#ifdef DEBUG  //------------> DEBUG
#else  //------------> DEBUG
#define call_check(AA, BB, CC, DD) ((void) 0)
#define call_free_check(AA, BB, CC) ((void) 0)
#define remove_ptr(AA, CC, DD) ((void) 0)
#endif //------------> DEBUG
```

---

18. Appendix F debug.h

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

```
//*************************************************************
// Copyright policy is GNU/GPL and it is requested that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
//*************************************************************

/*************************************************************
        Program for debugging C++/C programs
*************************************************************/

#define print_log(AA, BB, CC, DD, EE) ((void) 0)

#ifdef DEBUG
```

```
#include <iostream>
#include <string>
//#include <assert.h>  // assert() macro which is also used for debugging

const bool LOG_YES = true;  // print output to log file
const bool LOG_NO = false;  // Do not print output to log file

// Debugging code
// Use debug2_ to output result to a log file

#define debug_(NM, VL) (void) ( local_dbg(NM, VL, __FILE__, __LINE__) )
#define debug2_(NM, VL, LOG_FILE) (void) ( local_dbg(NM, VL, __FILE__, __LINE__, LOG_FILE) )

void local_dbg(char name[], char value[], char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], string value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], int value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], unsigned long value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], float value, char fname[], int lineno, bool logfile= false);
void local_dbg(char name[], double value, char fname[], int lineno, bool logfile= false);

#else  //--------> else

#define debug_(NM, VL) ((void) 0)
#define debug2_(NM, VL, LOG_FILE) ((void) 0)

#endif // DEBUG
```

---

19. Appendix G debug.cpp

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

```
//*****************************************************************
// Copyright policy is GNU/GPL and it is requested that
// you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
//*****************************************************************

/*****************************************************************
Program for debugging C++/C programs
*****************************************************************/


#ifdef DEBUG  // ONLY if DEBUG is defined than these functions below are needed

#include "debug.h"
//#include "log.h"

// Variable value[] can be char, string, int, unsigned long, float, etc...

void local_dbg(char name[], char value[], char fname[], int lineno, bool logfile) {
        if (value == NULL)
                return;
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %s\n", fname, lineno, name, value);
```

```
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], string value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %s\n", fname, lineno, name, value.c_str(
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], int value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %d\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], unsigned int value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %u\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], long value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %d\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], unsigned long value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %u\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], short value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %d\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], unsigned short value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %u\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], float value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %f\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

void local_dbg(char name[], double value, char fname[], int lineno, bool logfile) {
        if (logfile == true)
                print_log("\nDebug %s : Line: %d %s is = %f\n", fname, lineno, name, value);
        else
                cout << "\nDebug " << fname << ": Line: " << lineno << " " << name << " is = " <<

// You add many more here − value can be a class, ENUM, datetime, etc...


#endif // DEBUG
```

20. Appendix H Makefile

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

```
#//***************************************************************
#// Copyright policy is GNU/GPL and it is requested that
#// you include author's name and email on all copies
#// Author : Al Dev Email: alavoor@yahoo.com
#//***************************************************************

.SUFFIXES: .pc .cpp .c .o

CC=gcc
CXX=g++

MAKEMAKE=mm
LIBRARY=libString.a
DEST=/home/myname/lib

# To build the library, and main test program  uncomment line below :−
MYCFLAGS=−O −Wall

# To test without debug trace uncomment line below:−
#MYCFLAGS=−g3 −Wall

# To enable 'full debug ' tracing uncomment line below:−
#MYCFLAGS=−g3 −DDEBUG −Wall

#PURIFY=purify −best−effort

SRCS=my_malloc.cpp String.cpp debug.cpp example_String.cpp
HDR=my_malloc.h  String.h debug.h
OBJS=my_malloc.o String.o debug.o example_String.o
EXE=String

# For generating makefile dependencies..
SHELL=/bin/sh

CPPFLAGS=$(MYCFLAGS) $(OS_DEFINES)
CFLAGS=$(MYCFLAGS) $(OS_DEFINES)

#
# If the libString.a is in the current
# directory then use −L. (dash L dot)
MYLIBDIR=−L$(MY_DIR)/libmy −L.

ALLLDFLAGS= $(LDFLAGS)  $(MYLIBDIR)

COMMONLIBS=−lstdc++ −lm
MYLIBS=−lString
LIBS=$(COMMONLIBS)  $(MYLIBS)

all: $(LIBRARY) $(EXE)

$(MAKEMAKE):
```

```
        @rm −f $(MAKEMAKE)
        $(PURIFY) $(CXX) −M  $(INCLUDE) $(CPPFLAGS) *.cpp > $(MAKEMAKE)

$(EXE): $(OBJS)
        @echo "Creating a executable "
        $(PURIFY) $(CC) −o $(EXE) $(OBJS) $(ALLLDFLAGS) $(LIBS)

$(LIBRARY): $(OBJS)
        @echo "\n*********************************************"
        @echo "   Loading $(LIBRARY) ... to $(DEST)"
        @echo "*********************************************"
        @ar cru $(LIBRARY) $(OBJS)
        @echo "\n "

.cpp.o: $(SRCS) $(HDR)
#       @echo "Creating a object files from " $*.cpp " files "
        $(PURIFY) $(CXX) −c  $(INCLUDE) $(CPPFLAGS) $*.cpp

.c.o: $(SRCS) $(HDR)
#       @echo "Creating a object files from " $*.c " files "
        $(PURIFY) $(CC) −c $(INCLUDE) $(CFLAGS) $*.c

clean:
        rm −f *.o *.log *~ *.log.old *.pid core err a.out lib*.a afiedt.buf
        rm −f $(EXE)
        rm −f $(MAKEMAKE)

#%.d: %.c
#       @echo "Generating the dependency file *.d from *.c"
#       $(SHELL) −ec '$(CC) −M $(CPPFLAGS) $< | sed '\''s/$*.o/& $@/g'\'' > $@'
#%.d: %.cpp
#       @echo "Generating the dependency file *.d from *.cpp"
#       $(SHELL) −ec '$(CC) −M $(CPPFLAGS) $< | sed '\''s/$*.o/& $@/g'\'' > $@'

# Must include all the c flags for −M option
#$(MAKEMAKE):
#       @echo "Generating the dependency file *.d from *.cpp"
#       $(CXX) −M  $(INCLUDE) $(CPPFLAGS) *.cpp > $(MAKEMAKE)

include $(MAKEMAKE)
#include $(SRCS:.cpp=.d)
#include $(SRCS:.c=.d)
```