C++ Programming HOW−TO

Table of Contents

Table of Contents

C++ Programming HOW−TO

Al Dev (Alavoor Vasudevan) alavoor@yahoo.com

v8.0, 26 April 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 sytems that is − Linux, MS DOS, Apple Macintosh OS, Windows 95/NT, OS/2, IBM OSes, 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!).

16.Appendix D my_malloc.cpp

17.Appendix E my_malloc.h

18.Appendix F debug.h

19.Appendix G debug.cpp

20.Appendix H Makefile

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 **20 to 30 times FASTER than** Java. Java runs very slow because it is an byte−code−interpreted language running on top of "virtual engine". 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"* 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 *** 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 a million 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 this world!!

Instead of using char * and char[] all the C++ programmers MUST use the **'mychar class'** which is given in this document and **'string class'** included in the STDLIB. The **'mychar 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 **'mychar 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 **'mychar 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 **'mychar 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 compiler optimizer options like −O or −O3 can speed up C++.

Nowadays, "C" language is primarily used for "systems programming" to develop operating sytems, 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 is under your control and never changes and you will never know what the client side web−browser OS is. It can be Windows 95/98/NT/2000 or Linux, Apple Mac, OS/2, Solaris etc..

2.Download mychar

All the programs, examples are given in Appendix of this document. You can download as a single tar zip, the mychar 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 mychar class

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

```
        g++ example.cpp −lmychar
```

See illustration sample code as given below −

```
        mychar 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]);
        }

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

A complete example program "example_mychar.cpp" implementing the mychar class is given in Appendix A and mychar class is given in Appendix B.

3.1 Operators

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

- Equal to **==**
- Not equal to **!=**
- Assignment **=**
- Add to itself and Assignment **+=**
- String concatenation or addition **+**

For example to use operators −

```
        mychar aa;
        mychar 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 **'mychar class'** provides these functions :−

- Current string length **length()**
- 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 mychar variable **implode(char *glue)**
- Joins the strings in the list−head pointer explodeH and returns the mychar variable **join(char *glue)**
- Repeat the input string n times **repeat(char *input, unsigned int multiplier)**
- Reverse the string characters **reverse()**
- 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 mychar functions are given here, but **DO NOT USE** these, and instead use operators like '+', '+=', '==' etc.. These are 'private' members of the 'mychar' 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 mychar ? **bool equalto(const mychar & rhs, bool type = false)**
- Is equal to char* ? **bool equalto(const char *rhs, bool type = false)**

For example to convert integer to string do −

```
        mychar  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 mychar to convert
        aa = (mychar) 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 "mychar.h" which is using the my_malloc() and my_free() functions.

5.Pointers are problems

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.

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. In languages like Java, pointers are not supported at all !!

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

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://metalab.unc.edu/pub/Linux/docs/HOWTO/other−formats/ or ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/other−formats/
- Plain text format is in: ftp://metalab.unc.edu/pub/Linux/docs/HOWTO or ftp://metalab.unc.edu/pub/Linux/docs/HOWTO

- Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://metalab.unc.edu/pub/Linux/docs/HOWTO or ftp://metalab.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 tool" which can be got from − http://www.xs4all.nl/~cg/sgmltools/ 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)

This document is located at −

- http://metalab.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://metalab.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
And resize the window with mouse. See man page on xdvi.
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. Ghostscript and gv are also available on other platforms like OS/2, Windows 95 and NT.

- 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
```

```
To use ghostscript give −
        ghostscript howto.ps
```

You can read HTML format document using Netscape Navigator, Microsoft Internet explorer, Redhat Baron Web browser or any 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 − you must 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.

13.Appendix A example_mychar.cpp

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

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

// And uncommnet the TEST_MYCHAR variable below −

#ifdef TEST_MYCHAR

#include <stdlib.h>  // for putenv
#include "mychar.h"

//////////////////////////////////////////////////
// A example program to demo usage of mychar
//////////////////////////////////////////////////

int main(int argc, char **argv)
{
        char p_name[1024];
        sprintf(p_name, "PROGRAM_NAME=%s", argv[0]);
        putenv(p_name);
        print_total_memsize(); // in the beginning
        mychar aa, bb;

        //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, "\n0) aa.val is :%sEOF\n", (aa + " my rhs " ).val);

// Testing the = operator
aa = " lhs " ;
fprintf(stdout, "0-1) 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, "\n0) With lsh operator+, aa.val is :%sEOF\n", (" my lhs " + aa ).val);

//aa.str_cpy(bb.val);
aa = bb;
aa.to_upper();
fprintf(stdout, "1) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa.to_lower();
fprintf(stdout, "2) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa.ltrim();
fprintf(stdout, "3) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa.rtrim();
fprintf(stdout, "4) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa.trim();
fprintf(stdout, "5) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa = aa + " testing newlines \n\n\n\n";
aa.chop();
fprintf(stdout, "5-1) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa = aa + " rhs ";
fprintf(stdout, "6) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa = " lhs " + aa;
fprintf(stdout, "7) aa.val is :%sEOF\n", aa.val);

// Sample addition of numbers
//aa = (mychar) 9989 + "kkk" + 33 ;
aa = 9999;
fprintf(stdout, "7-1) aa.val is :%sEOF\n", aa.val);

aa = bb;
aa = " lhs " + aa + " rhs " + " 9989 " + " 33 ";
fprintf(stdout, "8) aa.val is :%sEOF\n", aa.val);

aa = " AA value ";
aa = bb + "alkja " + " 99djd " ;
fprintf(stdout, "9) aa.val is :%sEOF\n", aa.val);

aa = " AA value ";
aa = (mychar) "alkja " + " 99djd " ;
fprintf(stdout, "10) aa.val is :%sEOF\n", aa.val);
```

```
        aa = " AA value ";
        aa += (mychar) " al dev test kkk... " + " al2 slkj" + " al3333 ";
        fprintf(stdout, "11) 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, "12) aa.val is :%sEOF\n", aa.val);

        aa = "12345678";
        aa.reverse();
        fprintf(stdout, "13) aa.val is :%sEOF\n", aa.val);

        aa = " AA value ";
        aa = aa + " add aa " + aa + 1111 +" "+ 2222 + " " + 3.344 + aa;
        fprintf(stdout, "14) aa.val is :%sEOF\n", aa.val);

        aa.roundd(123456.0123456789012345, 13);
        fprintf(stdout, "15) double aa.val is :%sEOF\n", aa.val);

        aa.roundf(123456.0123456789, 13);
        fprintf(stdout, "16) float aa.val is :%sEOF\n", aa.val);

        // Test equal to operators
        aa = " AA value ";
        mychar cc(" AA value ");
        if (aa == cc)
                fprintf(stdout, "\naa=%s and cc=%s are equal!!\n", aa.val, cc.val);
        else
                fprintf(stdout, "\naa=%s and cc=%s are NOT equal!!\n", aa.val, cc.val);
        cc = "CC";
        if (aa == cc)
                fprintf(stdout, "\naa=%s and cc=%s are equal!!\n", aa.val, cc.val);
        else
                fprintf(stdout, "\naa=%s and cc=%s are NOT equal!!\n", aa.val, cc.val);
        if (aa == " AA value ")
                fprintf(stdout, "\naa=%s and string are equal!!\n", aa.val);
        else
                fprintf(stdout, "\naa=%s and string are NOT equal!!\n", aa.val);
        if (aa == " AA valuexxx ")
                fprintf(stdout, "\naa=%s and string are equal!!\n", aa.val);
        else
                fprintf(stdout, "\naa=%s and string are NOT equal!!\n", aa.val);

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

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

        print_total_memsize(); // in the end
        exit(0);
}
```

```
#endif // TEST_MYCHAR
```

14.Appendix B mychar.h

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

```
//*****************************************************************
// Copyright policy is GNU/GPL but additional restriction is
// 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 mychar or string class
// instead of char[] or char *
//

#ifndef __MYCHAR_H_
#define __MYCHAR_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;

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

                char *val;

                unsigned long length() { return strlen(val); }

                void ltrim();
                void rtrim();
                void trim();
```

```
        void chop();

        void to_upper();
        void to_lower();

        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);

        long pos(char substr[], unsigned long start);

        void explode(char *seperator);
        void implode(char *glue);
        void join(char *glue);
        void repeat(char *input, unsigned int multiplier);
        void reverse();
        void replace(char *needle, char *str);
        void str_tr(char *from, char *to);
        void center(int length, char padchar = ' ');
        void space(int number = 0, char padchar = ' ');
        void xrange(char start, char end);
        void compress(char *list);
        void delstr(int start, int length);
        void insert(char *newstr, int start = 0, int length = 0, char padchar = ' ');
        void left(int length = 0, char padchar = ' ');
        void right(int length = 0, char padchar = ' ');
        void overlay(char *newstr, int start = 0, int length = 0, char padchar = ' ');
        mychar substr(int start, int length = 0);

        mychar at(char *regx); // matches first match of regx
        mychar before(char *regx); // returns string before regx
        mychar after(char *regx); // returns string after regx

        bool isnull();
        void clear();

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

        mychar& operator+= (const mychar & rhs); // using reference will be faster
        mychar& operator= (const mychar & rhs); // using reference will be faster
        bool operator== (const mychar & rhs); // using reference will be faster
        bool operator== (const char *rhs);
        bool operator!= (const mychar & rhs);
        bool operator!= (const char *rhs);

        static  list<mychar>                 explodeH;  // list head

private:
        //static mychar *global_mychar; // for use in add operator
        //inline void free_glob(mychar **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 mychar & rhs, bool type = false);
                bool equalto(const char *rhs, bool type = false);
};
// Global variables are defined in mychar.cpp

#endif // __MYCHAR_H_
```

15.Appendix C mychar.cpp

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

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

// Use string class or this class
//
// 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++ mychar.cpp

#include "mychar.h"

// Global variables ....
//mychar *mychar::global_mychar = NULL; // global var
list<mychar>                 mychar::explodeH;

mychar::mychar()
{
        debug_("In cstr()", "ok");
        val = (char *) my_malloc(sizeof(char)* INITIAL_SIZE);
}

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

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

```
mychar::mychar(int bb)
{
        val = (char *) my_malloc(NUMBER_LENGTH); // integers 70 digits max
        sprintf(val, "%d", bb);
}

mychar::mychar(unsigned long bb)
{
        val = (char *) my_malloc(NUMBER_LENGTH); // long 70 digits max
        sprintf(val, "%lu", bb);
}

mychar::mychar(float bb)
{
        val = (char *) my_malloc(NUMBER_LENGTH); // float 70 digits max
        sprintf(val, "%f", bb);
}

mychar::mychar(double bb)
{
        val = (char *) my_malloc(NUMBER_LENGTH); // double 70 digits max
        sprintf(val, "%f", bb);
}

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

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

// MUST use pointer-to-pointer **aa, otherwise the argument
// is NOT freed !!
/*
inline void mychar::free_glob(mychar **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");
}
*/

// Explodes the string and returns the list in
// the list-head pointer explodeH
void mychar::explode(char *seperator)
{
        char *aa = NULL, *bb = NULL;
        aa = (char *) my_malloc(length());
        for (bb = strtok(aa, seperator); bb != NULL; bb = strtok(NULL, seperator) )
        {
                mychar *tmp = new mychar(bb);
                mychar::explodeH.insert(mychar::explodeH.end(), *tmp);
        }
        my_free(aa);

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

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

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

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

// Repeat the input string n times
void mychar::repeat(char *input, unsigned int multiplier)
{
        // For example -
        // repeat("1", 4) returns "1111"
        if (!input) // input == NULL
        {
                val[0] = 0;
                return;
        }

        val = (char *) my_malloc(strlen(input) * multiplier);
```

```
        for (unsigned int tmpii = 0; tmpii < multiplier; tmpii++)
        {
                strcat(val, input);
        }
}

// Reverse the string
void mychar::reverse()
{
        // For example −
        //              reverse() on "12345" returns "54321"
        char aa;
        unsigned long tot_len = length();
        unsigned long midpoint = tot_len / 2;
        for (unsigned long tmpjj = 0; tmpjj < midpoint; tmpjj++)
        {
                aa = val[tmpjj];  // temporary storage var
                val[tmpjj] = val[tot_len − tmpjj − 1];  // swap the values
                val[tot_len − tmpjj − 1] = aa; // swap the values
        }
}

// Replace all occurences of string 'needle' with 'str' in the haystack 'val'
void mychar::replace(char *needle, char *str)
{
        // For example −
        //              replace("AAA", "BB") on val = "some AAA and AAACC"
        //              reurns val = "some BB and BBCC"
}

// Translate certain chars
void mychar::str_tr(char *from, char *to)
{
        // For e.g ("abcd", "ABC") translates all occurences of each
        // character in 'from' to corresponding character in 'to'
}

// Center the text
void center(int length, char padchar = ' ')
{
        // For example −
        //              center(10, '*') on val="aa" returns "****aa****"
        //              center(10) on val="aa" returns "    aa    "
        // The result is a string of 'length' characters with val centered in it.
}

// 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 ' '
void space(int number, char padchar = ' ')
{
        // For example −
        //              space(3) on val = "I do not know"
        //                              will return "I   do   not   know"
        //              space(1, '_') on val = "A deep black space"
        //                              will return "A_deep_black_space"
        //              space() on val = "I   know     this"
        //                              will return "Iknowthis"
}
```

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

// Removes any characters contained in <list>. The default character
// for <list> is a blank ' '
void compress(char *list)
{
        // For example -
        //      compress("$,%") on val = "$1,934" returns "1934"
        //      compress() on val = "call me alavoor vasudevan" returns "callmealavoorvasudevan"
}

// Deletes a portion of string of <length> characters from <start> position.
// If start is greater than the string length then string is unchanged.
void delstr(int start, int length)
{
        // For example -
        //      delstr(3,3) on val = 'pokemon' returns 'poon'
}

// 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
void insert(char *newstr, int start = 0, int length = 0, char padchar = ' ')
{
        // For example -
        //      insert("something new", 4, 20, '*') on val = "old thing"
        //              returns "old something new*******thing"
}

// The result is string of <length> chars madeup of leftmost chars in val.
// Quick way to left justify a string.
void left(int length = 0, char padchar = ' ')
{
        // For example -
        //      left(10) on val = "Wig" returns "Wig       "
        //      left(4) on val = "Wighat" returns "Wigh"
}

// The result is string of <length> chars madeup of rightmost chars in val.
// Quick way to right justify a string.
void right(int length = 0, char padchar = ' ')
{
        // For example -
        //      right(10) on val = "never stop to saying" returns " to saying"
        //      right(4) on val = "Wighat" returns "ghat"
        //      right(6) on val = "4.50" returns "  4.50"
}

// 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
void overlay(char *newstr, int start = 0, int length = 0, char padchar = ' ')
{
        // For example -
        //      overlay("12345678", 4, 10, '*') on val = "oldthing is very bad"
```

```
        //              returns "old12345678**ery bad"
}

// sub string
mychar mychar::substr(int start, int length = 0)
{
        if (!length) // length == 0
                return(mychar(& val[start-1]) );
        else
        {
                mychar tmp = mychar(& val[start-1]);
                tmp.val[length-1] = 0;
                return(tmp);
        }
}

// If string is literrally equal to .. or not equal to
// If type is false then it is ==
bool mychar::equalto(const mychar & rhs, bool type = false)
{
        if (type == false) // test for ==
        {
                if (strlen(rhs.val) == length())
                {
                        if (!strncmp(rhs.val, val, length())) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
        else // test for !=
        {
                if (strlen(rhs.val) != length())
                {
                        if (!strncmp(rhs.val, val, length())) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
}

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

```
        else // test for !=
        {
                if (strlen(rhs) != length())
                {
                        if (!strncmp(rhs, val, length())) //  == 0
                                return true;
                        else
                                return false;
                }
                else
                        return false;
        }
}

// find position, matching substr beginning from start..
long mychar::pos(char *substr, unsigned long start)
{
        char * tok;
        long res = -1;

        if ( !isnull() && (start < strlen(val) ) )
        {
                tok = strstr(val + start, substr);
                if (tok == NULL)
                        res = -1;
                else
                        res = (long) (tok - val);
        }
        return res;
}

bool mychar::isnull()
{
        if (val[0] == '\0')
                return true;
        else
        {
                if (val == NULL)
                        return true;
                else
                        return false;
        }
}

void mychar::clear()
{
        val = (char *) my_realloc(val, 10);
        val[0] = '\0';
}

// Remove trailing new-lines
void mychar::chop()
{
        unsigned long tmpii = strlen(val) - 1 ;
        for (; tmpii >= 0; tmpii--)
        {
                if (val[tmpii] == '\n')
                        val[tmpii] = 0;
                else
                        break;
        }
}
```

```
void mychar::ltrim()
{
        // May cause problems in my_realloc since
        // location of bb will be destroyed !!
        char *bb = val;

        if (bb == NULL)
                return;

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

        if (bb != NULL && bb != val)
        {
                debug_("doing string copy", "done");
                //str_cpy(bb); // causes problems in my_realloc and bb is getting destroyed!!
                strcpy(val, bb); // strcpy is ok since val space is > bb space
        }
        else
                debug_("Not doing string copy", "done");
}

void mychar::rtrim()
{
        for (long tmpii = strlen(val) − 1 ; tmpii >= 0; tmpii−−)
        {
                if ( isspace(val[tmpii]) )
                        val[tmpii] = '\0';
                else
                        break;
        }
}

void mychar::trim()
{
        rtrim();
        ltrim();
}

void mychar::to_lower()
{
        for (long tmpii = strlen(val); tmpii >= 0; tmpii−−)
        {
                val[tmpii] = tolower(val[tmpii]);
        }
}

// Use for rounding off fractions digits of floats
// Rounds−off floats with given precision and then
// stores the result into mychar's val field
// Also returns the result as a char *
void mychar::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) );

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

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

void mychar::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);

        val = (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(val, "%lu.%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 9999) // (MAX_PREC - 1) digits
                sprintf(val, "%lu.0%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 999) // (MAX_PREC - 2) digits
                sprintf(val, "%lu.00%lu", (unsigned long) integ_flt, deci_int);
        else
        if (deci_int > 99) // (MAX_PREC - 3) digits
                sprintf(val, "%lu.000%lu", (unsigned long) integ_flt, deci_int);
        else
```

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

void mychar::to_upper()
{
        for (long tmpii = strlen(val); tmpii >= 0; tmpii--)
        {
                val[tmpii] = toupper(val[tmpii]);
        }
}

void mychar::str_cpy(char bb[])
{
        debug_("In str_cpy bb", bb);
        if (bb == NULL)
        {
                val[0] = '\0';
                return;
        }

        unsigned long tmpii = strlen(bb);

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

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

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

void mychar::str_cpy(int bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%d", bb);
        str_cpy(tmpaa);
}

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

void mychar::str_cpy(float bb)
{
        char tmpaa[100];
        sprintf(tmpaa, "%f", bb);
        str_cpy(tmpaa);
```

```
}

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

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

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

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

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

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

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

mychar operator+ (const mychar & lhs, const mychar & rhs)
{
        /******************************************************/
        // Note : For adding two char strings, first cast mychar
        // as in -
        //aa = (mychar) "alkja " + " 99djd " ;
        /******************************************************/

        mychar tmp(lhs);
        tmp.str_cat(rhs.val);
        return(tmp);

        /*
        if (mychar::global_mychar == NULL)
        {
                mychar::global_mychar = new mychar;
                mychar::global_mychar->str_cpy(lhs.val);
                mychar::global_mychar->str_cat(rhs.val);
                //return *mychar::global_mychar;
                return mychar(mychar::global_mychar->val);
        }
        */
```

```
        /*
        else
        if (mychar::global_mychar1 == NULL)
        {
                debug_("1)global", "ok" );
                mychar::global_mychar1 = new mychar;
                mychar::global_mychar1->str_cpy(lhs.val);
                mychar::global_mychar1->str_cat(rhs.val);
                return *mychar::global_mychar1;
        }
        */
        /*
        else
        {
                fprintf(stderr, "\nError: cannot alloc global_mychar\n");
                exit(-1);
        }
        */

        /*
        mychar *aa = new mychar;
        aa->str_cpy(lhs.val);
        aa->str_cat(rhs.val);
        return *aa;
        */
}

mychar mychar::operator+ (const mychar & rhs)
{
        mychar tmp(*this);
        tmp.str_cat(rhs.val);
        debug_("rhs.val in operator+", rhs.val );
        debug_("tmp.val in operator+", tmp.val );
        return (tmp);
}

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

        this->str_cpy(rhs.val);
        debug_("rhs value", rhs.val );

        // Free global vars memory
        //free_glob(& mychar::global_mychar);
        //if (mychar::global_mychar == NULL)
                //fprintf(stderr, "\nglobal_mychar is freed!\n");

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

// Using reference will be faster in = operator
mychar& mychar::operator+= (const mychar & rhs)
{
        /*******************************************************/
        // Note : For adding two char strings, first cast mychar
```

```
        // as in -
        //aa += (mychar) "cccc" + "dddd";
        /*******************************************************/

        if (& rhs == this)
        {
                debug_("Fatal error: In operator+= rhs is equals 'this' ptr", "ok");
                return *this;
        }
        this->str_cat(rhs.val);
        return *this;
        //return (mychar(*this));
}

bool mychar::operator== (const mychar & rhs)
{
        return(equalto(rhs.val));
}

bool mychar::operator== (const char *rhs)
{
        return(equalto(rhs));
}

bool mychar::operator!= (const mychar & rhs)
{
        return(equalto(rhs.val, true));
}

bool mychar::operator!= (const char *rhs)
{
        return(equalto(rhs, true));
}
```

16.Appendix D my_malloc.cpp

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

```
//*****************************************************************
// Copyright policy is GNU/GPL but additional restriction is
// 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_memsize);
                        //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).mem_size;
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size!!", func_name);
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*iter1).total_memsize);
                                        fprintf(ferr, "\n%s mem_size = %u", func_name, (*iter1).mem_size);
                                        fprintf(ferr, "\n%s amem_size = %u\n", func_name, amem_size);
                                }
                        }
                        (*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_memsize);
                        //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).mem_size;
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size!!", func_name);
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*iter1).total_memsize);
                                        fprintf(ferr, "\n%s mem_size = %u\n", func_name, (*iter1).mem_size);
                                }
                        }
                        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_size);
                        //fprintf(ferr, "\n%s Total memory allocated = %lu\n",  func_name, (*iter1).total_memsize);
                        if ((*iter1).total_memsize > 0 )
                        {
                                if ((*iter1).total_memsize >= (*iter1).mem_size )
                                        (*iter1).total_memsize = (*iter1).total_memsize - (*iter1).mem_size;
                                else
                                {
                                        fprintf(ferr, "\n\n%s total_memsize is less than mem_size!!", func_name);
                                        fprintf(ferr, "\n%s total_memsize = %lu", func_name, (*iter1).total_memsize);
                                        fprintf(ferr, "\n%s mem_size = %u", func_name, (*iter1).mem_size);
                                }
                        }
                        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::total_memsize);
        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 variable in main() function\n",
                                        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, program_name);
                else
                if (mtype == REALLOC)
                {
                        fprintf(ferr, "\n%s: %s - Fatal Error - my_realloc() failed.", date_str, program_name);
                        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, program_name, dtype);
                }

                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 mychar . To get this file, in the web−browser, save this file as 'Text' type.

```
//*****************************************************************
// Copyright policy is GNU/GPL but additional restriction is
// 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 mychar . To get this file, in the web−browser, save this file as 'Text' type.

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

#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 mychar . To get this file, in the web−browser, save this file as 'Text' type.

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

#ifdef DEBUG  // ONLY if DEBUG is defined then 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 = " << value << endl; }

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 = " << value.c_str() << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

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 = " << value << endl; }

// 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 mychar . To get this file, in the web−browser, save this file as 'Text' type.

```
#//*****************************************************************
#// Copyright policy is GNU/GPL but additional restriction is
#// 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=libmychar.a
DEST=/home/myname/lib

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

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

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

#PURIFY=purify -best-effort
```

```
SRCS=my_malloc.cpp mychar.cpp debug.cpp example_mychar.cpp
HDR=my_malloc.h  mychar.h debug.h
OBJS=my_malloc.o mychar.o debug.o example_mychar.o
EXE=mychar

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

CPPFLAGS=$(MYCFLAGS) $(OS_DEFINES)
CFLAGS=$(MYCFLAGS) $(OS_DEFINES)
#
#MYLIBDIR=-L$(MY_DIR)/libmy

ALLLDFLAGS= $(LDFLAGS)  $(MYLIBDIR)

COMMONLIBS=-lstdc++ -lm
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)
```