· -I = including directory

· -L = to include library

· -O = specifies object file

· -m = message

· Flags for shared memory : static and shared

· -shared : gives a shared library

· -Pic : static library

· Shared libraries are not embedded in executable but in loaded run times.

· Coverage tool : to check the code coverage percentage and help us to identify if we miss any unit test cases.

· Splint : It will help us to bug or debug at source level.

Generating the index data file

· $ ctags -R. ('.' is for index file)

· $ vi file1.c #relevant file name

· In vi run below command

· $ set tags=<path-of-tag-file>

· In above command path-of-tag-file can be absolute or relative path of tag files.

NAVIGATING

You can bring the cursor to the fun call of "mod1\_fun", and press ctrl-]. You will notice vi opens the files where mod fun is defined.

CODE

· Ctags?

Cscope

· It will generate cscope database. Which is called cscope.out file.

· It rebuilds the database.

· $ cscope 'find . -name'\*.[ch]'

· Or $ find . -name'\*[ch]'> cscope.files.

CODE :

· Find . -name '\*.[ch]' > cscope.files

· Ctrl d is to come out of the files.

· Ll

· Vi cscope.files

· Cscope -I cscope.files

· Ps

These files should not be pushed into the git so we should remove them before the pushing.

Performance profiling using gprof :

· Gprof allows us to measure how much percentage of time is spent in the diff funs. This way, we can focus on optimizing time.

· Gcc -o application -pg main.c

· Th option -pg adds instrumentation code

· $ ./application

· $ gprof ./application gmon.out

Valgrind Tools :

· >> valgrind -v --tool=memcheck --leak-check=full --show-reachable=yes --log-file=valclient2 ./a.out

o (a.out is the application name).

o This command will produce the valgrind report.

· >> vi valclient1

· >> vi main.c

· >> gcc -o app main.c

· >>valgrind -v --tool=memcheck --leak-check=full --show-reachable=yes --log-file=valclient2 ./app

· >>vi valclient2

· >>valgrind -v --tool=memcheck --leak-check=full --show-reachable=yes --log-file=valclient2 ./app

C Programming

· Basic Structure of C

o Document section: (Details like when and why it is created)

· /\*

Description:

[Example:

Modules add, sub

Int add(int val1,val2): add is doing addition of val1 with val2 and return the result to the called(main)]

Author: [initials of the author]

DOC/DOM: [if it was created by us then that date but if it is a modified then it will be authors date]

Version: [0.1v]

[1.0 stable version => basing of the project]

\*/

o Link section

o Definition section

o Global declaration section

· C Tokens

o The smallest individual units in a program are known as token.

They are :

§ Keywords

§ Identifiers

§ Constants

§ Numeric const

§ String const

§ Strings

§ Special symbols

§ Operators

§ Arithmetic

§ Bitwise

§ Logical

§ Relational

§ Assignment

· !high = low

· !low = high

· Modifiers:

o The basic datatypes may have several modifiers prece4ding them to serve the needs of various situations.

They are:

§ Signed

§ Unsigned

§ Long

§ Short

May be applied to character and integer datatypes. However, the modifier long may also be applied to double.

After applying the modifiers to basic datatypes, their size is as follows.

Name Size (in bytes) Range of values

Unsigned char 1 0 to 255

Signed Char 1 -128 to 127

Unsigned int 2 0 to 65535

Signed int 2 -3176 to 32767

Short int 2 -31768 to 32767

Long int 4 -

Long double 10 -

Code :

Ternary operator:

· Int a = 10;

· Int b = 20;

· Int c = 30;

· Res = (exp1) ? (exp2) : (exp3);

· Res = (a>b) ? ((a>c) ? a:c) : ((b>c) ? b:c);

· % operator is not to be used in float and string.

Managing Input Output Operations

· Reading a Character :

Reading a single character can be done by using the function getchar(). This can also be done with scanf().

· Syntax : ch = getchar();

· Input/Output statements:

To perform the basic i/o functions c provides the library of functions. This library is called stdio.h.

Ex :

Scanf(), printf(), getchar(), getch(), putchar(), gets(), puts(), etc.

· There are two types of i/o statements. They are:

o Formatted I/O statements

o Unformatted I/O statements

· Formatted I/O statements:

This enables the user to specify the type of the data and the way in which it should be read in or written out.

Ex : scanf(), printf().

· Unformatted I/O statements:

This do not specify the type of data and the way it should read in or written out.

Ex : getchar(), gets(), putchar(), puts(), etc.

· Scanf() :

Syntax :

scanf("Control String",address\_list);

Where

"control string" is a sequence of one or more character groups. Each character group is a combination of % symbol and one of the conversion characters. The control string specifies the type of the values which are to be supplied to the variables.

"address list"

Character Group

%c Read a single char

%d Read a decimal integer

%f Read a floating point value

%u Read a unsigned value

· "0d" implies that when we want the format filled with the 0 instead of the empty blocks then we can use "0" so that the empty will be filled with 0 and remaining will be filled with the given values.

· In printf you will be de referencing the value so that it can be stored.

· To remove buffer we use "fflush".

Error :

"Stack smashing detected" : terminated

>> This error occurs because c language does not have a garbage collector and all the memory stored up while executing the code is more.

Addresses of

i : 1440203472

f : 1440203476

ch : 1440203471

str1 : 1440203488

d : 1440203480

These are known as base addresses of any variables or where the addresses are stored at the starting.