UNIX C SOFTWARE DEVELOPMENT TOOLS

1) <u>MAKE</u>

The UNIX system programming environment provides a tool, make, that automaticallly keeps track of dependencies among files that make up a program, and makes it easy to create executable programs. In using make, you specify the way that parts of your program are dependent on other parts, or on other code. This specification of teh dependencies underlying a program is placed in a "Makefile". Here is an example of "Makefile".

```
# We put the comment here
# A basic Makefile to r_main and p_main


CC = cc
CFLAGS = -O


all : r_main p_main

r_main : reverse.o  r_main.o
	$(CC) -o r_main reverse.o r_main.o

p_main : p_main.o reverse.o palindrome.o
	$(CC) -o p_main  reverse.o palindrome.o p_main.o

reverse.o : reverse.h reverse.c
	$(CC) $(CFLAGS) -c reverse.c

palindrome.o : reverse.h palindrome.h palindrome.c
	$(CC) $(CFLAGS) -c palindrome.c

r_main.o : reverse.h r_main.c
	$(CC) $(CFLAGS) -c r_main.c

p_main.o : palindrome.h p_main.c
	$(CC) $(CFLAGS) -c p_main.c

clean :
	rm -f r_main p_main *.o core
```

A Makefile has serveral components to it, as follows.

- Comments

  Everything between "#" (pound sign) and the net RETURN is ignored by make.

- Variables

  "make" allows you to define named variables similar to those used in the shell. In the above example, we set CC to be cc, then the variable $(CC) contains the name of cc. We can set CC to be other compilers, such as gcc, g++, cxx or f77.

- Dependencies

  Dependencies are specified by naming the taget modeules on the left, followed by a colon, followed by the modules on which the target depends. For example, "all" depends on "r_main" and "p_main", and "r_main" depends on "reverse.o" and "r_main.o" while "reverse.o" depends on "reverse.h" and "reverse.c", ... so and so.

- Commands

  The dependency line is followed by the commands that must be executed if one or more of the dependent modules has been changed. Command lines must be indented at least one tab stop from the left margin. Tabs are required; the equivalent number of spaces don't work

When you run the "make" without any argument, the "make" tool looks for a file called "makefile" or "Makefile" in the current directory. The makefile is examined; source files that have been changed since they were last compiled are recompiled, and any file that depends on another that has been changed will be recomplied.

Here we "make" in the first time

```
csh> make
cc -O -c reverse.c
cc -O -c r_main.c
cc -o r_main reverse.o r_main.o
cc -O -c p_main.c
cc -O -c palindrome.c
cc -o p_main  reverse.o palindrome.o p_main.o
```

We excute the program

csh> r_main abc
The reverse string of abc is cba.

csh> p_main noon
noon is a palindrome.

When we "make" again, nothing will happen

csh> make

However, when we try modify a source file, says palindrome.h. We use the "touch" to update the time attribute of the "palindrome.h" file.

csh> touch palindrome.h

Now the "palindrome.h" should have the time attribute later than "palindrome.o". Now we "make" again, "p_main.c" is re-compiled according to the dependency specified in "Makefile".

csh> make
cc -O -c p_main.c
cc -O -c palindrome.c
cc -o p_main reverse.o palindrome.o p_main.o

If we update the "reverse.h", more files need to be re-compiled.

csh> touch reverse.h
csh> make
cc -O -c reverse.c
cc -O -c r_main.c
cc -o r_main reverse.o r_main.o
cc -O -c palindrome.c
cc -o p_main reverse.o palindrome.o p_main.o

We can also specify the "clean" dependency to clean up our previous programs and object files build.

csh> make clean
rm -f r_main p_main *.o core

Here we only want to build "r_main" program

csh> make r_main

```
cc -O -c reverse.c
cc -O -c r_main.c
cc -o r_main reverse.o r_main.o
```

"make" without argument is default to "make all"

```
csh>  make
cc -O -c p_main.c
cc -O -c palindrome.c
cc -o p_main  reverse.o palindrome.o p_main.o
```

2) **Build you own library**

You can use "ar" and "ranlib" to build your own library.

First we only make the object files:

```
csh>  make clean
rm -f r_main p_main *.o core
csh>  make reverse.o
cc -O -c reverse.c
csh>  make palindrome.o
cc -O -c palindrome.c
```

Here we have the two object files

```
csh>  ls *.o
palindrome.o        reverse.o
```

Now we use "ar" to create a "string.a" library.

```
csh>  ar r string.a reverse.o palindrome.o
ar: Warning: creating string.a
```

We list the content of this library.

```
csh>  ar t string.a
reverse.o
palindrome.o
```

We make the "p_main.o" too.

```
csh>  make p_main.o
cc -O -c p_main.c
```

Now we try to use "string.a" library to compile our "p_main"

```
csh>  cc -o p_main  p_main.o string.a
ld:
Archive: string.a has no table of contents (not searched)
        add one with 'ar ts'
Undefined:
palindrome
```

However, the cc compiler cannot find the "palindrome" function in the "string.a" library. We need to use "ranlib" to adds a table of contents to the library.

```
csh>  ranlib string.a
csh>  ar t string.a
__________ELEL_
reverse.o
palindrome.o
```

Now, we can use "string.a" to compile our "p_main" program

```
csh>  cc -o p_main  p_main.o string.a
csh>  p_main nooooon
nooooon is a palindrome.
```

Here we specify another Makefile (Makefile2) to make use of the "string.a" library.

```
csh>  cat Makefile2
CC = cc
CFLAGS = -O



all : r_main p_main

r_main : string.a r_main.o
        $(CC) -o r_main r_main.o string.a

p_main : string.a p_main.o
```

```
        $(CC) -o p_main  p_main.o string.a

string.a : reverse.o palindrome.o
        ar r string.a reverse.o palindrome.o
        ranlib string.a

reverse.o : reverse.h reverse.c
        $(CC) $(CFLAGS) -c reverse.c

palindrome.o : reverse.h palindrome.h palindrome.c
        $(CC) $(CFLAGS) -c palindrome.c

r_main.o : reverse.h r_main.c
        $(CC) $(CFLAGS) -c r_main.c

p_main.o : palindrome.h p_main.c
        $(CC) $(CFLAGS) -c p_main.c


clean :
        rm -f r_main p_main *.o core *.a
```

Now we use the "Makefile2" instead of "Makefile"

```
csh>  make -f Makefile2 clean
rm -f r_main p_main *.o core *.a
```

Here, we test the dependency of "palindrome.h". The new "palindrome.o" in "string.a" will be replaced by the new one by the "ar r" command.

```
csh> make -f Makefile2 clean
rm -f r_main p_main *.o core *.a

csh>  touch palindrome.h
csh>  make -f Makefile2
cc -O -c reverse.c
cc -O -c palindrome.c
ar r string.a reverse.o palindrome.o
ar: Warning: creating string.a
ranlib string.a
cc -O -c r_main.c
```

```
cc -o r_main r_main.o string.a
cc -O -c p_main.c
cc -o p_main  p_main.o string.a
```

3) **<u>Use lint to check your potential bugs</u>**

Now we add a unreach statetment in "palindrome2.c"

```
csh>  diff palindrome2.c palindrome.c
21,22d20
< free (reversedStr);
<
```

We use "lint" to check the potential bugs in "palindrome.c" and "palindrome2.c"

```
csh>  lint palindrome.c


==============
name used but not defined
        reverse     palindrome.c(17)
name defined but never used
        palindrome          palindrome.c(11)
value type used inconsistently
        malloc      llib-lc(421) :: palindrome.c(15)
value type declared inconsistently
        malloc      llib-lc(421) :: palindrome.c(15)
```

```
csh>  lint palindrome2.c

palindrome2.c
==============
Warning: (23)  function palindrome has return(e); and return;
warning: statement not reached
        (21)
```

```
==============
name used but not defined
        reverse     palindrome2.c(17)
name defined but never used
        palindrome          palindrome2.c(11)
value type used inconsistently
        malloc      llib-lc(421) :: palindrome2.c(15)
value type declared inconsistently
        malloc      llib-lc(421) :: palindrome2.c(15)
free        llib-lc(424) :: palindrome2.c(21)
function argument ( number ) used inconsistently
        free( arg 1 )       llib-lc(424) :: palindrome2.c(21)
```

4) **Use profiler to find out the performance of your functions**

It's often handy to be able to see where a program is spending its time. For example, if a greater-than-expected amount of time is being spent in a particular fuction. it might be worth optimizing the function by hand for better perforamce. The "prof" utility allows you to obtain a program's profile.

In order to use "prof", we need to compile the program with the "-p" flag. We define the "-p" flay in the "Makefile3"

```
csh> make -f Makefile3 clean
rm -f r_main p_main pr_main *.o core
csh>  make -f Makefile3 pr_main
cc -O  -c pr_main.c
cc -O  -c reverse2.c
cc -p -o pr_main reverse2.o pr_main.o
```

Now we run the "pr_main" program

```
csh>  pr_main
The reverse of 10th is ht01
The reverse of 1st is ts1
The reverse of 2nd is dn2
The reverse of 3rd is dr3
The reverse of 4th is ht4
The reverse of 5th is ht5
The reverse of 6th is ht6
The reverse of 7th is ht7
The reverse of 8th is ht8
```

```
The reverse of 9th is ht9
The reverse of a is a
The reverse of Aaron is noraA
The reverse of ABA is ABA
The reverse of Ababa is ababA
The reverse of aback is kcaba
The reverse of abalone is enolaba
....
```

After the the "pr_main" program execution, a "mon.out" is generated

```
csh>  ls *.out
mon.out
```

Now, we use "prof" to see the "pr_main" program profile.

```
csh>  prof pr_main
Profile listing generated Mon Aug 22 23:14:51 1994 with:
          prof pr_main

blkclr and bzero (../bzero.s) synonymous: using latter
-------------------------------------------------------------------------------
*  -p[rocedures] using pc-sampling;                                           *
*  sorted in descending order by total time spent in each procedure;          *
*  unexecuted procedures excluded                                             *
-------------------------------------------------------------------------------

Each sample covers 8.00 byte(s) for 5.9%  of 0.1700 seconds

%time     seconds  cum %     cum sec  procedure (file)

41.2      0.0700   41.2      0.07 write (../write.s)
29.4      0.0500   70.6      0.12 reverse2 (reverse2.c)
11.8      0.0200   82.4      0.14 string (../doscan.c)
 5.9      0.0100   88.2      0.15 fscanf (../scanf.c)
 5.9      0.0100  100.0      0.17 fstat (../fstat.s)
```

We can also use "pixie" to get the "calls/lines" information. The steps are as follows

```
csh>  pixie -o pr_main.pixie pr_main
pixie registers: r31, r30, and r23.
old code = 22192 bytes, new code = 65592 bytes (3.0x)
```

```
csh> pr_main.pixie
The reverse of 10th is ht01
The reverse of 1st is ts1
The reverse of 2nd is dn2
The reverse of 3rd is dr3
The reverse of 4th is ht4
The reverse of 5th is ht5
The reverse of 6th is ht6
The reverse of 7th is ht7
The reverse of 8th is ht8
The reverse of 9th is ht9
The reverse of a is a
....
```

The "pr_main.Addrs" and "pr_main.Counts" are generated after "pr_main.pixie" execution.

```
csh> prof -pixie pr_main pr_main.Addrs pr_main.Counts
Profile listing generated Mon Aug 22 23:16:21 1994 with:
prof -pixie pr_main pr_main.Addrs pr_main.Counts

blkclr and bzero (../bzero.s) synonymous: using latter
--------------------------------------------------------------------------------
*  -p[rocedures] using basic-block counts;                                    *
*  sorted in descending order by the number of cycles executed in each        *
*  procedure; unexecuted procedures are excluded                              *
--------------------------------------------------------------------------------

1102308 cycles

        cycles %cycles  cum %     cycles  bytes procedure (file)
                                   /call  /line

        756012   68.58  68.58       7561     18 reverse2 (reverse2.c)
        185043   16.79  85.37         55     14 _flsbuf (../flsbuf.c)
         81864    7.43  92.80        819     17 _doprnt (../doprnt.c)
         30620    2.78  95.58        307     34 string (../doscan.c)
         21196    1.92  97.50        212     27 _doscan (../doscan.c)
          9994    0.91  98.41         34      7 strlen (../strlen.c)
          8700    0.79  99.19         29     16 ungetc (../ungetc.c)
          1900    0.17  99.37         19     30 printf (../printf.c)
          1329    0.12  99.49       1329     21 main (pr_main.c)
          1000    0.09  99.58         10     20 fscanf (../scanf.c)
```

818	0.07	99.65	205	13 morecore (../malloc.c)
623	0.06	99.71	623	19 _fwalk (../data.c)
606	0.05	99.76	6	16 write (../write.s)
588	0.05	99.82	19	32 strncmp (../strncmp.c)
400	0.04	99.85	200	32 _findenv (../getenv.c)
304	0.03	99.88	76	10 malloc (../malloc.c)
178	0.02	99.90	45	12 fclose (../fclose.c)
132	0.01	99.91	66	19 monitor (./mon.c)
131	0.01	99.92	66	5 bzero (../bzero.s)
108	0.01	99.93	108	15 _findiop (../findiop.c)

csh> prof -lines pr_main pr_main.Addrs pr_main.Counts

Profile listing generated Mon Aug 22 23:17:14 1994 with:
   prof -lines pr_main pr_main.Addrs pr_main.Counts

blkclr and bzero (../bzero.s) synonymous: using latter

```
----------------------------------------------------------------------------------
*  -l[ines] using basic-block counts;                                              *
*  grouped by procedure, sorted by cycles executed per procedure;                  *
*  '?' means that line number information is not available.                       *
----------------------------------------------------------------------------------
```

procedure (file)	line	bytes	cycles	%cycles
reverse2 (reverse2.c)	10	16	400	0.04
	16	12	300	0.03
	18	24	600	0.05
	19	24	3912	0.35
	21	**20**	**750200**	**68.06**
	23	8	200	0.02
	25	16	400	0.04
_flsbuf (../flsbuf.c)	58	28	23828	2.16
	63	20	17020	1.54
	71	24	0	0.00
	72	12	0	0.00
	76	8	0	0.00
	77	16	0	0.00
	78	8	0	0.00
	82	16	10212	0.93

As the line of "reverse2.c" shows a lot of cycles, we should look at this line.

5) **<u>UNIX debugger: dbx</u>**

The UNIX debugger dbx allows you to symbolically debug a program. Although it is not as good as most professional debuggers on the market, it comes as a handly standard utility. dxb includes the following facilities:

- single stepping
- breakpoints
- accessing and modifying variables
- searching for functions
- tracing

In order to use the "dbx", we need to compile the program with the "-g" option

```
csh> cc -g -o ptest_bug ptest_bug.c
```

Now we go into the debugger to debug the "ptest_bug" program

```
csh> dbx ptest_bug
dbx version 2.10.1
Type 'help' for help.
reading symbolic information ...
main:  34  if (argc !=2) {printf ("Usage: %s [test string] \n",argv[0]);exit (1);}
```

We list the "main" function

```
(dbx) list main
          28            return (palinAux (str, start + 1, stop -1));
          29  }
          30
          31  main (int argc, char *argv[])
          32  {
          33
          34     if (argc !=2) {printf ("Usage: %s [test string] \n",argv[0]);exit (1);}
          35
```

```
 if (palindrome(argv[1]))
          printf ("%s is a palindrome.\n",argv[1]); else
      printf ("%s is not a palindrome.\n",argv[1]) ;
```

We list the "palindrome" function

(dbx) list palindrome

```
  int palindrome (str)
  char *str;
  {
>  return (palinAux (str, 1, strlen (str)));
  }
  int palinAux (str, start, stop)
  char *str;
```

We list the "palinAux" function

(dbx) list palinAux

```
  char *str;
  int start;
  int stop;
  {
>  if (start >= stop)
          return (TRUE);
  else if (str[start] != str[stop])
          return (FALSE);
  else
          return (palinAux (str, start + 1, stop -1));
  }
```

We want to trace the "palinAux" function

(dbx) trace palinAux

[2] trace palinAux

We set the breakpoint in "palinAux" function.

(dbx) stop in palinAux

[3] stop in palinAux

Run the program in dbx

(dbx) run noon

[2] calling palinAux(**str = 0x7fffbce6 = "noon", start = 1, stop = 4**) from function palindrome

[3] stopped at [palinAux:23 ,0x400214] if (start >= stop)

Then print the value of "str[start]" and "str[stop]"

(dbx) print str[start]

'o'

(dbx) print str[stop]

'\000'

List the "palindrome" function

(dbx) list palindrome

```
      5  int palindrome (str)
      6
      7  char *str;
      8
      9  {
     10
>*   11  return (palinAux (str, 1, strlen (str)));
     12
     13  }
     14
     15  int palinAux (str, start, stop)
     16
     17  char *str;
```

Advance the program step by step

(dbx) step

```
            [palinAux:25 ,0x400234]          else if (str[start] != str[stop])
(dbx) step
            [palinAux:26 ,0x40025c]          return (FALSE);
(dbx)
step
             [main:38 ,0x400310]   printf ("%s is not a palindrome.\n",argv[1]) ;
(dbx)
step
noon is not a palindrome.
            [main:41 ,0x400324]    }
(dbx) quit
```

After the debugging, we know that we pass the wrong arguments in function "palinAux". Then we correct the program and save it as ptest.c

```
csh> ls -l *.c
-rw-r--r--  1 shlam       585 Aug 22 18:45 ptest.c
-rw-r--r--  1 shlam       583 Aug 22 18:45 ptest_bug.c

csh> diff ptest_bug.c ptest.c
11c11
< return (palinAux (str, 1, strlen (str)));
---
> return (palinAux (str, 0, strlen (str)-1));
```

Recompile the "ptest" and run the dbx again

```
csh> cc -g -o ptest ptest.c
csh> dbx ptest
dbx version 2.10.1
Type 'help' for help.
reading symbolic information ...
main:  34  if (argc !=2) {printf ("Usage: %s [test string] \n",argv[0]);exit (1);}
(dbx) trace palinAux
[2] trace palinAux
(dbx) stop in palinAux
[3] stop in palinAux
```

Show the status of our setting.

```
(dbx) status
```

[2] trace palinAux
[3] stop in palinAux
(dbx) run noon
[2] calling palinAux(str = 0x7fffbce2 = "noon", start = 0, stop = 3) from function palindrome
[3] stopped at [palinAux:23 ,0x400214] if (start >= stop)
(dbx) step
[palinAux:25 ,0x400234] else if (str[start] != str[stop])
(dbx)
step
[palinAux:28 ,0x400264] return (palinAux (str, start + 1, stop -1));
(dbx)
step
[2] calling palinAux(str = 0x7fffbce2 = "noon", start = 1, stop = 2) from function palinAux
[2] [palinAux:23 ,0x400214] if (start >= stop)
(dbx)
step
[palinAux:25 ,0x400234] else if (str[start] != str[stop])
(dbx)
step
[palinAux:28 ,0x400264] return (palinAux (str, start + 1, stop -1));
(dbx)
step
[2] calling palinAux(str = 0x7fffbce2 = "noon", start = 2, stop = 1) from function palinAux
[2] [palinAux:23 ,0x400214] if (start >= stop)
(dbx)
step
[palinAux:24 ,0x40022c] return (TRUE);
(dbx)
step
[main:37 ,0x4002f4] printf ("%s is a palindrome.\n",argv[1]); else

Finally, we got the correct answer.

(dbx)
step
noon is a palindrome.
[main:41 ,0x400324] }

We can get the on-line help in dbx

(dbx) help

Command syntax: "help <keyword>", <keyword> is one of the following list:

```
most_used, quit, alias, record, playback, history, lineedit, run,
debug, rerun, stop, step, next, trace, delete, catch,
ignore, cont, return, when, goto, print, printf, printregs,
where, status, whatis, which, whereis, assign, tag, up,
down, func, dump, display, list, search, edit, file,
use, set, setenv, sh, stopi, conti, stepi, nexti,
tracei, variable, register, builtin, expression
```

```
(dbx) help trace
trace <exp>                  - print <exp> when it changes
trace <exp> at <line>        - print <exp> when <line> is reached
trace <exp> in <proc>        - print <exp> when call <proc>
trace <exp> at <line> if <exp> - print <exp> when <line> is reached and <exp>
trace <exp> in <proc> if <exp> - print <exp> when call <proc> and <exp>
(dbx) quit
```

After the debugging, we can use "strip" command to removes all of the symbol table, relocation, debugging, and profiling infomration from the debugged program

```
csh>  ls -l ptest
-rwxr-xr-x  1 shlam      40972 Aug 22 20:11 ptest

csh>  strip ptest
csh>  ls -l ptest
-rwxr-xr-x  1 shlam      24576 Aug 22 20:14 ptest
```

The size of "ptest" is reduced after the "strip" command.