Prep C
Datatypes and Directories

Dr. Madhav Rao

July 1, 2016

Datatypes

Create a directory named *dtypes* that hangs off of your *home* directory. Move into that directory. Create a C file named *minmax.c* and place into the following code.

```
#include <stdio.h>
#include <limits.h>

int main()
    {
    printf("Q1. Signed char minimum value: %d\n", SCHAR_MIN );
    printf("Q2. Signed char maximum value: %d\n", SCHAR_MAX );
    printf("Q3. Unsigned char minimum value: %d\n", 0 );
    printf("Q4. Unsigned char maximum value: %d\n", UCHAR_MAX );
    printf("Q5. Char minimum value: %d\n", CHAR_MIN );
    printf("Q6. Char maximum value: %d\n", CHAR_MAX );
    printf("Q7. Signed short minimum value: %d\n", SHRT_MIN );
    printf("Q8. Signed short maximum value: %d\n", SHRT_MAX );
    printf("Q9. Unsigned short minimum value: %d\n", 0 );
    printf("Q10. Unsigned short maximum value: %d\n", USHRT_MAX );
    printf("Q11. Signed int minimum value: %d\n", INT_MIN );
    printf("Q12. Signed int maximum value: %d\n", INT_MAX );
    printf("Q13. Unsigned int minimum value: %u\n", 0 );
    printf("Q14. Unsigned int maximum value: %u\n", UINT_MAX );
    printf("Q15. Signed long minimum value: %ld\n", LONG_MIN );
    printf("Q16. Signed long maximum value: %ld\n", LONG_MAX );
    printf("Q17. Unsigned long minimum value: %lu\n", 0 );
    printf("Q18. Unsigned long maximum value: %lu\n", ULONG_MAX );
    printf("Q19. Signed long long minimum value: %lld\n", LLONG_MIN );
    printf("Q20. Signed long long maximum value: %lld\n", LLONG_MAX );
    printf("Q21. Unsigned long long minimum value: %lu\n", 0 );
    printf("Q22. Unsigned long long maximum value: %llu\n", ULLONG_MAX );
    return 1;
    }
```

Compile this code and name the executable as *minmax* file. Run the executable. After running the executable, you will see 22 lines/statements coming from your minmax program. During the demonstration, you will be asked to explain the number of bits for each datatype.

Create another directory named *enum* within *dtypes* directory and form two new files named send.c and recv.c in *enum* directory. The two files are created to visualize how transfer of data takes place from sender-end to receiver-end. Part of the protocol in transmitting and receiving requires data and parity checking. We will not bother of parity today, but will learn later. We will assume that we are sending and receiving the data on the same terminal. The *ZLINE, MPD and QPD* are the names of the petrol pumps used by Larsen And Toubro company. We will not go into the details of the embedded controller operations of these petrol pumps. However think of this code written by developers for user operations. The user can select either of the three petrol pumps used, combined with the either of parity checking schemes. In the following code, *ZLINE* is used as petrol pumps and *no-parity* is used as parity-checking scheme.

Place the following code in send.c

```
#include<stdio.h>
int main()
    {
    enum petrolpumps {ZLINE=0, MPD, QPD};
    enum parityTypes {noparity = 0, odd, even};
    enum petrolpumps using= ZLINE;
    enum parityTypes parity = noparity;
    int packet = 0;

    /* Forming data packets */
    packet = using << 8;
    packet = packet | parity;

    /* Sending the data packet */
    fprintf(stderr,"In send program- Sending the packet\n");
    fprintf(stdout,"%d\n",packet);
    return 1;
    }
```

and the following code in recv.c file

```
#include<stdio.h>

void packetIdentify(int data)
    {
    enum petrolpumps {ZLINE=0, MPD, QPD};
    enum parityTypes {noparity = 0, odd, even};
    int pumps=0, par=0;
    par = data & 0xff;
    pumps = data >> 8;
    if(pumps == ZLINE)
        fprintf(stderr,"ZLINE pumps \n");
    else if(pumps == MPD)
        fprintf(stderr,"MPD pumps \n");
    else if(pumps == QPD)
        fprintf(stderr,"QPD pumps \n");
    else
        fprintf(stderr,"Not specified\n");

    if(par == noparity)
        fprintf(stderr,"No parity was used\n");
    else if(par == odd)
        fprintf(stderr,"Odd parity was used\n");
    else if(par == even)
        fprintf(stderr,"Even parity was used\n");
    else
        fprintf(stderr,"None parity specified\n");
    }

int main()
    {
    int packetrecv;
    fscanf(stdin,"%d",&packetrecv);
    fprintf(stderr,"In recv program, Receiving the packet\n");
    packetIdentify(packetrecv);
    return 1;
    }
```

Now compile the following code into respective executables. Name the executables as *send* and *recv*. For transmission and receiving purpose, we normally need a channel. In this activity we will make console-terminal as a channel. Our intention is to pipe the output of *send* program to *recv* program. Following linux command will do the needful.

```
./send | ./recv
```

Currently the send program sends *ZLINE* as the petrol-pump type with *no-parity.* You are supposed to modify the send.c file in such a way that you need to send all 9 combinations from *send* program. All nine combinations are stated in the below table.

No.	Petrol pump types	Parity type
1	ZLINE	no parity
2	MPD	no parity
3	QPD	no parity
4	ZLINE	odd
5	MPD	odd
6	QPD	odd
7	ZLINE	even
8	MPD	even
9	QPD	even

Also modify the recv program such that it receives 9 such combinations.

HINT: For nine combinations, use nine different variables. packet is one among nine such variables in send program or you can use a loop nine times. For recv program, use nine variables and call the packetIdentify nine times with each such variables or use a loop nine times to receive the data.

Directories to practice

But first, you will need to learn a little bit about Linux, the operating system you are using. Please do the following web tutorial:

http://www.ee.surrey.ac.uk/Teaching/Unix/

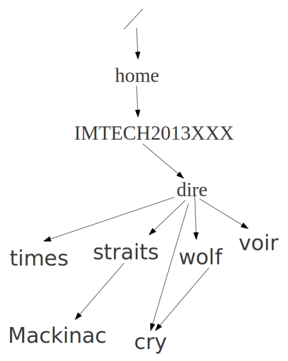
When you are done, continue to the next section.

Starting Up

To do this activity, you will use the following commands, among others:

ls	list the files in the current directory
ls *x*	list the file or directory *x*
cd *x*	change to directory *x*
cd ..	change to the parent directory *x*
cd	change to the home directory *x*
pwd	show the current directory *x*
mkdir *x*	create directory *x*

In your home directory, create a directory named *dire*. Move into the *dire* directory and create the following directory structure:



Your home directory is the one named after your name, not the directory named home. This is because Linux is designed as a multi-user system and there may be more than one home directory (one for each user). The *home* directory holds all the users' home directories.

Please read the rest of this actvity before starting. There's one tricky part that you need to keep in mind while creating directories.

Placing files in directories

In each directory, create a file whose name is the first letter of the directory. For example, in the *dire* directory, create a file named *d*. An easy way to create a file is to use the *touch* command. To create a file named *d*, issue the command:

```
touch d
```

A file created in this way exists, but has nothing in it. You could have also created an empty file with vim, as in:

```
vim d
```

and while in *vim*, immediately running the *vim* command `:wq`.

Sharing directories

To share a directory (such as *cry*), one parent directory needs to own it and the other parent needs to link to it. For this activity, have the *wolf* directory own the *cry* directory (that is to say, create the *cry* directory while in the *wolf* directory). Now move up to the *dire* direction and then share or link the *cry* directory with this command:

```
ln -s wolf/cry
```

Make sure you are in the *dire* directory! Now do a listing of the *dire* directory. You should see something like this:

```
cry   d   straits   times   voir   wolf
```

Listing the *cry* directory (`ls cry`), should yield:

```
c
```

Verification

To see if you did things correctly, run this command while in the *dire* directory:

```
find . -print
```

You should see the following output:

```
.
./d
./cry
./times
./times/t
./voir
./voir/v
./wolf
./wolf/w
./wolf/cry
./wolf/cry/c
./straits
./straits/Mackinac
./straits/Mackinac/M
./straits/s
```

Note that the file *c* in the directory *cry* is listed only once. The order of the listing will depend upon the order you create your directories.

Starting all over

If you've made a mistake and you wish to start all over, first move to the home directory and then run the command:

```
rm -r dire
```

The *rm* command with the *-r* option (the *-r* stands for *recursive*) is quite dangerous. It removes the entire directory hierarchy. Remember, any file or directory removed with the *rm* command is unrecoverable!

Demonstration

As usual, show the activities to your instructor. Before that, do a directory listing (`ls`); you should see something like

```
cry   d   straits   times   voir   wolf
```