Lecture 2 - 6/1/2022

Introduction

Nelson has posted videos from previous semesters in Panopto incase you want to watch them before class.

Ekesh Kumar's notes is available on the class website

Review

When we log into Grace, everyone's environment looks different because everyone has different directories and things in their environment.

At this point, you should have run the setup command.

- This will give you the `216` and `215public` directories.
  - the `216public` directory is read-only, so students cannot change it.

We will work on projects and things in the `216` directory. The `216public` directory is read-only and it is where class materials are posted.

The colors in Nelon's screen are due to the usage of MobaXTerm which has text highlighting.

This class has very basic unix commands

- cd
- ls
  - ls -F shows slash next to all the folders
- mkdir

When you connect to Grace, you will see a number next to `Grace` such as `grace3:`. This number represents the specific host within the cluster to which you have connected.

Copying Things

This command copies the `Week01` directory from the `216public` directory to our own `216` directory in our home directory.

```
cp -r ~/216public/lecture_examples/Week01/ ~/216
```

- `cp` is the copy command.
- `~` (tilde) represents the home directory.
- `-r` means recursive
- `~/216public/lecture_examples/Week01/` is what we want to grab.
- `~/216` is where we want to put it.

Note: pressing tab after typing the beginning of a word or command will autocomplete it. It will even autocomplete using the names of the contents which you can access.

Note: Since `~` represents the home directory, typing `cd ~` is the same as typing `cd`.

Removing things

This command removes the `Week01` and everything in it.

- Unix does not give any second chances. You cannot easily recover a deleted file.

```
rm -r Week01
```

To bypass all the confirmations, use `-f`, which means force.

Moving things

Suppose we want to move things from a directory `old` to `old2`.

The command `mv` moves things.

```
mv old old2
```

Typing in files

To type in a file, we can simply use an editor such as vi, vim, or nano. For example, we can type in a file called `p1.c` by using

```
vi p1.c
```

Then we can simply type code such as this:

```
#include <stdio.h>

int main() {
    int x = 100;

    printf("%d\", x + 2);

    return 0;
}
```

To save your work, hit escape, then hit `shift` + `:`, then type `w` and then `enter`.

- `shift` + `:` escapes the typing section of `vi` and lets you enter commands.
- the command `w` denotes write changes to file
- the command `q` denotes quit `vi`.

To summarize:

- Escape using `shift`+`:`.
- Write the file using `w`.
- Quit the editor using `q`.
- Commands can be chained together such that `wq` will write your changes to the file and subsequently quit `vi`. Often, we will write and quit so we just do `wq` in vim.

These commands are the same in `vim`.

More commands:

- `o` adds a line

Class Website

On the class website, you can see resources by going to the dropdown menu. Here, you will find all the Linux commands you will need for this class.

Aliases

When one wants to compile code in this class, we must use `gcc` with many flags like this:

```
gcc -ansi -Wall -g -O0 -Wwrite-strings -Wshadow -pedantic-errors -fstack-
protector-all -Wextra p1.c
```

This is a rediculous amount of flags to write, so we can create an alias for `gcc` such that whenever we use `gcc`, all those flags are used. To do this, follow the instructions here.

To know if you did the alias correctly, you will know if you pass the class.

- just kidding. You can check by typing `alias gcc`

C

Example: `print_example.c`

Let's take a look at `print_example.c` in `C-Language-II-Code` in `216public`. Nelson opened this code by using the command

```
more print_example.c
```

`more`, like `cat`, simply displays the text content of the file: in this case, `print_example`. Here is the output of `more print_example.c`:

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

```
int main(void) {   /* Notice the void to indicate no parameters */
   int age = 18;
   float salary = 100.50;
   char gender = 'F';
   const char *address = "AV Williams Bld";

   printf("Age: %d, Salary: %f, Gender: %c\n", age, salary, gender);
   printf("Address: %s\n", address);

   return 0;
}
```

Let's analyze this code.

- The first line is just the normal include statement for being able to use standard input and output.
- In the `main` parameters, we see `void`. This signifies that the program doesn't take any arguments.
- There are then four variables declared
  - `const char` is how a string is declared in C.
- The line

  ```
  printf("Age: %d, Salary: %f, Gender: %c\n", age, salary, gender);
  ```

  is a formatted string.
  - as you can see, the use of `%s` allows us to display a string.
  - the use of `%d` allows us to display a `double`
  - the use of `%f` allows us to display a `float`
  - the use of `%c` allows us to display a `char`
  - In a formatted string, make sure that all parameters are in the same order as their placeholders.

By running `gcc print_example.c`, we can see that the output is as expected:

```
Age: 18, Salary: 100.500000, Gender: F
Address: AV Williams Bld
```

printf conversion specifiers

- `%c` prints the corresponding argument to printf as an unsigned character
- `%d` prints as a decimal integer
- `%f` prints in floating point format
- `%s` prints as a string (null-terminated character array)
- `%%` prints a % (so `%%` prints as %)
- `%u` prints as an unsigned integer
- `%x` prints in hexadecimal (use `%X` for capital A-F)
- `%e` prints in exponential form (e.g., 6.02300e3)

Clear Command

The command `clear` will clear the entire screen to make the screen cleaner to look at.

Inputs in C

In C, we use the `scanf()` function to read input

- Definition:
  - Its declaration is also included with `#include <stdio.h>` (its definition in the C standard I/O library)
- Syntax:
  - scanf("format string", &variable1, &variable2, &variable3, ...);
- Format strings can contain:
  - Whitespace, meaning any whitespace at that point will be skipped
  - **Conversion specifiers** (`%d`, `%f`, `%c`, etc) which cause something to be read
    - Many (**but not all**) skip leading whitespace anyway
  - Any other characters, which are then required in input
- Arguments (variable names) indicate where converted input is to be stored, and their types should match the format specifiers
- `scanf()` does not check for type agreement between the conversion specifier and the associated variable
- If a given conversion specifier doesn't match, `scanf()` stops processing at that point
- **Value returned**: `scanf()` returns the number of items assigned into variables or EOF (End of File Marker)
- **Notice the `&` associated with the variables**
  - Next to a variable means “get the address of this variable”

Example: `reading.c`

Next, let's look at the program `reading.c`.

```
#include <stdio.h>

int main() {
   int value;
   char the_character;

   printf("Enter an integer: ");
   scanf("%d", &value);
   printf("First integer value is %d\n", value);

   printf("\nEnter a character and an integer: ");

   /* We need a space before %c; this space means consume spaces. */
   /* After the first integer we provided we have a newline. This */
   /* newline is the result of pressing enter.  If you don't      */
   /* remove it, the_next_character variable will have a newline  */
   /* character.  Remember, when you press enter that is actually */
   /* the '\n' character.                                         */

   /* Notice space before %c. */
```

```
    scanf(" %c%d", &the_character, &value);
    printf("The character is %c\n", the_character);
    printf("The second integer value is %d\n", value);

    return 0;
}
```

After compiling the code, here is a sample run:

```
Enter an integer: 23
First integer value is 23

Enter a character and an integer: X 45
The character is X
The second integer value is 45
```

Similar to `printf()`, the function `scanf()` also requires a formatted string.

- Notice the `&` sign next to the variable `value` in the formatted string. Here, the `&value` represents the address of the `value` variable. This is how `scanf()` formatted strings work.

Here's another sample run of the code:

```
Enter an integer:







          7
First integer value is 7

Enter a character and an integer:
```

Notice that there is a lot of space where the user entered spaces (you'll see them if you highlight the sample code-block) and newlines, but the program only took in the input once 7 was entered. This is because in `scanf()`, `%d` ignores whitespace such as spaces and new-lines.

Now, suppose we modifed the code such that the line

```
/* Notice space before %c. */
    scanf(" %c%d", &the_character, &value);
```

becomes

```
/* Notice no space before %c. */
   scanf("%c%d", &the_character, &value);
```

Here, we removed the space before `%c`.

Now, here's a sample run of this new code:

```
Enter an integer: 7
First integer value is 7

Enter a character and an integer: X 45
The character is

The second integer value is 7
```

Where did the `X` and the `45` go? Let's take a look at the input. This is what we typed:

```
7\nX 45
```

What did the code do with this?

1. The code took `7` as the first integer.
2. The code took `\n` as the character
   - This was printed after `The character is`
3. The code tried to take `X` as an integer, but since `X` isn't one, the process stops.

This is why the space was put before the `%c` in that line. It signifies that the placeholder should skip all the spaces.

scanf() conversion specifiers

- `%d`, `%x`, `%o`
  - Reads a decimal, hex, or octal number into an `int`
- %f reads in a `float`
- %lf reads in a `double`
- %ld reads in a `long`
- %c reads in a `char`
- %s reads in a `string` (bounded by whitespace)

C control statements

- C has `if/else`, `while`, `for`, `do-while`, and `switch` statements, as in Java
  - Due to the compiler flags we use you can't declare variables in for loop
  - In C89/C90 you cannot declare variables in a for loop header;
    - Possible in C99/C11.

- `break`: immediately ends loop
- `continue`: skip remainder of loop body; returns to beginning of the loop
  - In case of `for` loop, performs third expression of the loop `(expr1;expr2;expr3)`
- Don't abuse `break/continue`, and rarely use `continue` if at all
- Compound statement
  - Can go anywhere a single statement goes
  - Surrounded by `{ }`
  - Can have block local variables declared at beginning of function

Example: `control_stmts.c`

Here is the example code:

```c
#include <stdio.h>

int main() {
   int limit, done, curr;

   /* Reading a positive value */
   do {
      printf("Enter limit to print even/odd values: ");
      scanf("%d", &limit);
      done = limit > 0;
      if (!done) {
         printf("Invalid value %d provided (must be > 0)\n", limit);
      }
   } while(!done);

   /* Printing even values */
   printf("Even values up to %d: ", limit);
   curr = 2;
   while(curr <= limit) {
      printf("%d ", curr);

      curr += 2;
   }
   printf("\n");

   /* Printing odd values */
   printf("Odd values up to %d: ", limit);
   for (curr = 1; curr <= limit; curr += 2) {
      printf("%d ", curr);
   }
   printf("\n");

   return 0;
}
```

As you can see, `while` loop and for loop are the `same`.

Also, notice that we **cannot** declare the iteration variable using `int curr` inside the `for` loop header.

Formatting

In `printf()`, if we wish to print a `float` to only 2 decimal places (such as for a currency), we can use the placeholder `.%2f` instead of `%f`.

Controlling Formatting

We can supply modifiers to conversion specifiers (e.g., a field width, precision, etc.) to format our output exactly as we want

- `%04x`: format as unsigned hex number, with 4 spaces and zero padding
- `%-10s`: format as string, allot 10 spaces, left justify (default is right justified)
- `%6.4f`: format as floating point, allot 6 spaces, 4 digits after the decimal point

Naming Convention in `C`

In C, we don't use camelCase by convention. Instead, we separate words with underscores. Instead of `letterGrade`, we would use `letter_grade`.

C Functions

- C is not object-oriented so we don’t use the term method (we use function)
- C Functions have the following format

  ```
  returnType functionName (parameter list) {
    /* statements */
  }
  ```

- To call a function - `functionName(argument list);`
- We have local variables and `return` statements as in Java
- Function prototype declarations provide information about a function's return type and parameters, but do not define the function
  - **Prototype is the first line of a function without the `{`**
- Using function declarations (prototypes) allows the compiler to check your function calls for correctness
- When a program (in a single file) has several functions (including `main`), in which order can the functions appear?
  - Without prototypes, functions must be defined before used (must appear before main if possible)
  - With prototypes, place prototypes at the top and functions can appear in any order
- Typical program organization
  - #includes
  - Prototypes
  - `main()` and functions (in any order although some people prefer `main()` first or last)

Example: `compute_grade.c`

Here is an example of the code that shows functional programming in C. It also that functions and vairable names are named using `underscores` **instead** of `camelCase`.

```c
#include <stdio.h>

/*
 * letter_grade prototype. Without it the function definition
 * must appear appear before main.
 */
char letter_grade(float score);  /* prototype */

int main() {
   /* Providing values for score1 and score2 to */
   /* see what happens when not enough values   */
   /* are provided by user.                     */
   float score1 = 77, score2 = 88, avg;
   int values_read;

   printf("Enter two scores using <score1> and <score2> format: ");
   values_read = scanf("%f and %f", &score1, &score2);
   printf("The number of values read is %d\n", values_read);

   avg = (score1 + score2) / 2;
   printf("Average for %f and %f is %.2f\n", score1, score2, avg);

   printf("Your letter grade is %c\n", letter_grade(avg));

   return 0;
}

/* Function definition */
char letter_grade(float score) {
   char grade;

   if (score >= 90) {
      grade = 'A';
   } else if (score >= 80) {
      grade = 'B';
   } else {
      grade = 'F';
   }

   return grade;
}
```

In C, everything is passed to functions by value (copy), not reference.

Note that the function `letter_grade` appears as a prototype **before** `main`, but then is defined after it. The prototype appears here so because the compiler needs to know about `letter_grade` when it reads and compiles `main`, which calls `letter_grade`.

- Without this prototype, the program will not be able to compile.

Interesting Nelson Quote

"You see, we say that you can date Java, but you will marry C. `:D` Are you with me? This is the language you'll give a ring. YAAAY! Eh Jaja" - Nelson.

C Functions Cont.

- Arguments in C (**all types of arguments**) are passed by value
  - Pass by value - you initialize parameter with a copy of the argument
  - Let's quickly review passing by value
- Where should the function definition appear?
  - Before `main`? After `main`?
  - Can we have a function defined inside of another function?
    - The set of `gcc` flags that we are using does not allow it
- How about recursion? Yes, as we have a stack as in Java
- Can we have function overloading?
  - NO.
  - `printf` function looks like it is overloaded, but it is NOT
    - `printf` with three or twenty arguments refers to the same function
  - In C, functions can be designed to accept several paremeters. This is not overloading.

Manual Pages

To see documentation about functions in linux, you can use the command `man` followed by the funciton. For example, the command

```
man printf
```

will output these pages

```
PRINTF(1)                         User Commands                        PRINTF(1)



NAME
       printf - format and print data

SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION

DESCRIPTION
       Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

       --help display this help and exit
       --version
              output version information and exit

       FORMAT controls the output as in C printf.  Interpreted sequences are:

       \"     double quote

       \\     backslash

       \a     alert (BEL)

       \b     backspace

       \c     produce no further output

       \e     escape

       \f     form feed

       \n     new line

       \r     carriage return

       \t     horizontal tab

       \v     vertical tab

       \NNN   byte with octal value NNN (1 to 3 digits)

       \xHH   byte with hexadecimal value HH (1 to 2 digits)

       \uHHHH Unicode (ISO/IEC 10646) character with hex value HHHH (4 digits)

       \UHHHHHHHH
              Unicode character with hex value HHHHHHHH (8 digits)

       %%     a single %

       %b     ARGUMENT  as  a string with '\' escapes interpreted, except that
              octal escapes are of the form \0 or \0NNN

       and all C format specifications ending with one of diouxXfeEgGcs,  with
       ARGUMENTs converted to proper type first.  Variable widths are handled.

       NOTE:  your  shell  may  have  its own version of printf, which usually
       supersedes the version described here.  Please refer  to  your  shell's
       documentation for details about the options it supports.

       GNU  coreutils  online  help:  <http://www.gnu.org/software/coreutils/>
       Report printf translation bugs to <http://translationproject.org/team/>

AUTHOR
       Written by David MacKenzie.
```

```
COPYRIGHT
       Copyright © 2013 Free Software Foundation, Inc.   License  GPLv3+:  GNU
       GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
       This  is  free  software:  you  are free to change and redistribute it.
       There is NO WARRANTY, to the extent permitted by law.

SEE ALSO
       printf(3)

       The full documentation for printf is maintained as  a  Texinfo  manual.
       If  the  info  and printf programs are properly installed at your site,
       the command

              info coreutils 'printf invocation'

       should give you access to the complete manual.



GNU coreutils 8.22                July 2020                          PRINTF(1)
```

Furthermore, one can look at different pages of the manual by entering a page number as that parameter. For example

```
man 3 printf
```

will show the 3rd page of printf documentation. I'm not gonna put that here because it's even longer than this page.

Beware

- Local variables in a function must be declared at the top of the function otherwise you will get an error similar to the following:

  ```
  “declaration.c:6:4: error: ISO C90 forbids mixed declarations and code [-Wpedantic]”
  ```

- Do not add \n at the end of scanf

  ```
  scanf("%d\n", &age);
  ```

- Do not add \n at the end of scan
- If you want to compile and modify class examples you must copy them to your 216 directory

Example: `beware.c`

Take a look at this example code

```
int main(void) {
   int age;

   printf("Enter age: ");
   scanf("%d", &age);
   printf("Your age is: %d\n", age);

   return 0;
}
```

You'll notice that if you put `\n` at the end of `scanf()`, your code will not work as expected.