QUEEN'S UNIVERSITY FINAL EXAMINATION
FACULTY OF ARTS AND SCIENCE
SCHOOL OF COMPUTING

CISC220 Section 2: System-Level Programming - Shadi Khalifa
December 13th, 2016

INSTRUCTIONS TO STUDENTS:
This examination is 3 HOURS in length.
This examination worth 40 points.
There are two sections to this examination (Bash and C).

Answer **ANY** and **ONLY TEN** of the 12 questions [*Only the first 10 answers will be marked*]

Please answer questions in the **answer booklets**.

The following aids are allowed:
Casio FX-991 calculator

Exam paper is distributed with: a *Bash Commands Sheet* and a *C Code Sheet*.

Put your student number on all pages of all answer booklets, including the front.
GOOD LUCK!

PLEASE NOTE:
Proctors are unable to respond to queries about the interpretation of exam questions.
Do your best to answer exam questions as written.

SECTION 1: LINUX AND BASH

Question 1 [4 points]: [Give it 15 minutes]

A) **[1 point]** A Linux distribution is the combination of two components. Mention these **two components** and mention **two criteria for deciding which** Linux distribution to use.

B) **[3 points]** Write a bash script that takes a number of integer arguments and SUMs the EVEN numbers together and the ODD numbers together and writes out the results to the standard output. Example:

```
./sum 42 5 -6 2 9 -3

38 11
```

Question 2 [4 points]: [Give it 15 minutes]

A) **[1 point]** Answer the following [*one line per question*]:
   i. What does pipes do?
   ii. What does I/O redirection do?

B) **[3 points]** Write a **single line bash command using pipes |** that
- **reads** the file "software.txt" that has a list of software names, one name per line.
- **install** each of the software using apt-get. [*hint: no loops*]
- All **errors** should be **redirected** and **appended** to the "errors" file.

Question 3 [4 points]: [Give it 15 minutes]

A) **[2 points]** In the context of Version Control, what does these terms mean: Repository, Sandbox, Pull, Push, Trunk, Branch, Tag, and Merge? [*one line per term*]

B) **[2 points]** Write a **single line bash conditional command [using && and ||]** that
- **Add and commit** the file "myprogram.c" to the git **local repository**.
- If successful, **get** changes done by other team members from the git remote repository.
- If successful, **add** the file to the git **remote repository.**

Question 4 [4 points]: [Give it 15 minutes]

A) [1 point] In the context of Computer Networks, define the following: Switch, Router, Dynamic Host Configuration Protocol [DHCP] and Domain Name System [DNS]. [*one line per term*]

B) [3 points] Given the following IP table on your Linux machine:

```
Chain INPUT (policy ACCEPT)
target     prot opt   source          destination
ACCEPT     tcp  --    anywhere        anywhere          tcp dpt:ssh
DROP       tcp  --    10.0.0.55       anywhere          tcp dpt:http
DROP       all  --    anywhere        anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source                     destination
DROP       all  --  anywhere                   facebook.com
```

Choose either YES or NO for each of the following and write ONLY the corresponding iptable command. [DON'T write commands for both YES and NO]

i. Can your Linux machine access (send traffic to) facebook.com?
   - If yes, write the **iptables command** to either add or remove a rule to prevent access to facebook.com
   - If no, write the **iptables command** to either add or remove a rule to allow access to facebook.com

ii. Can your Linux machine get traffic from queensu.ca on port 21546?
   - If yes, write the **iptables command** to either add or remove a rule to prevent queensu.ca incoming traffic on port 21546
   - If no, write the **iptables command** to either add or remove a rule to allow queensu.ca incoming traffic on port 21546

iii. Can you SSH your Linux machine from the machine with IP address 10.0.0.55?
   - If yes, write the **iptables command** to either add or remove a rule to allow everyone BUT 10.0.0.55 SSH your Linux machine.
   - If no, write the **iptables command** to either add or remove a rule to allow ONLY 10.0.0.55 SSH your Linux machine.

Question 5 [4 points]: [Give it 15 minutes]

A) [1 point] What is the difference between a user process, a daemon process and a cron job? Mention who starts each and if they run in the foreground or background.[*one line per term*]

B) [3 points] What is the output of the following bash script when run using the following arguments?
(i) ./myscript.sh 10010101001
(ii) ./myscript.sh 10011110001
(iii) ./myscript.sh 1001011011

(Write your trace, values of $i, ${1:$i:1} and ${1: -$(($i+1)):1} to take partial marks if your final answer is not correct).

```
#!/bin/bash
i=0
l=${#1}
m=$(($l/2))

while [ $i -lt $m ]
  do
    if [ "${1:$i:1}" != "${1: -$(($i+1)):1}" ]
      then
      echo "0"
      exit
    fi
    i=$(($i+1))
done
echo "$1"
```

Question 6 [4 points]: [Give it 15 minutes]

A) [1 point] Mention **four** Version Control best practices.[*one line per term*]

B) [3 points] Write a script that reads a number of words from the user using the **read command**, stores them in an **Array** then checks each word if its **second half is the same as the first half** or not. Example:

```
./checker.sh

Insert a list of words: tomtom cisc220 moomoo 2020

Output: Yes No Yes Yes
```

SECTION 2: C PROGRAMMING LANGUAGE

Question 7 [4 points]: [Give it 15 minutes]

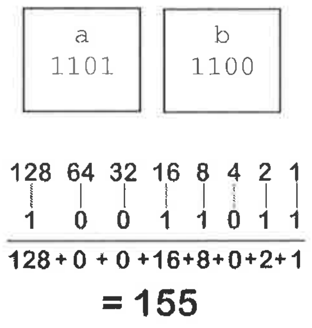
A) [1 point] What is the difference between compiling and interpreting code? Give example of a compiled language and an example for an interpreted language. [*one line per term*]

B) [3 points] What is the output of the following C program? What does this program do?

(Write your trace, values of a, b and c to take partial marks if your final answer is not correct).

```
#include<stdio.h>

int main(){
int a=13, b=12, c=0;
while ( a != 0) {
     c = b & a;
     b = b ^ a;
     c = c << 1;
     a = c;
}
c = b;
printf("a=%d - b=%d - c=%d \n", a, b, c);
return 0;
}
```



Question 8 [4 points]: [Give it 15 minutes]

A) [1 point] Given the following union

`union u {`
`   int x;`
`   double y;`
`   char z;`
`   int* p;`
`};`
	What is the output of the following, given that memory addresses are stored in 4 bytes?
	i. sizeof(union u);
	ii. sizeof(union u*);

B) [3 points] Write a C program that reads a number from the user and calculates its factorial.
Factorial n = n* n-1 * n-2 * …… * 1

Question 9 [4 points]: [Give it 15 minutes]

A) [1 point] What is the difference between a memory-leak and a dangling pointer? [*one line per term*]

B) [3 points] This code is supposed to create an array of 3 random numbers, sums them up and append the summation at the end of a file. The code has multiple errors. Errors are in bold and underlined and have a FIXME comment next to them. Rewrite the program code after fixing the errors.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
int* myfun(){
    int n[MAX]; //FIXME 1
    for(int i=0; i<MAX ; i++) {
      n[i] = rand();
   }
   return n;
}

void myfun2(int* a, int s /*FIXME 2*/){
   s=0; //FIXME 3
   for(int i=0; i<MAX ; i++){
     s+=a[i]; //FIXME 4
   }
}

void main(){
   FILE *f = fopen("data", "r"); //FIXME 5
   int *a = myfun();
   int s = 0;
   myfun2(a,s); //FIXME 6
   fprintf(f, "%d\n",s);
   //FIXME 7: Don't cause memory leak or a dangling pointer
   //FIXME 8: Make sure data is flushed and appended to the file
}
```

Question 10 [4 points]: [Give it 15 minutes]

A) [1 point] What is the difference between call-by-value and call-by-reference? [*one line per term*]

B) [3 points] Draw the **Symbol Table** [variable name and its memory address] and the **RAM Memory locations** [values stored in each variable] for the following program and **write down the program output**. Memory addresses **start from address: 100.** Use decimal values for the addresses, not hexadecimal. [int is 4 byte, pointers are 4 bytes, double is 8 bytes]

```
#include<stdio.h>
void myfun(int* a, double* b) {
   printf("Before %x %x %d %f\n", a, b, *a,*b);
   (*a)++;
   (*b)--;
   printf("After %x %x %d %f\n", a, b, *a,*b);
}
int main () {
   int x = 16;
   double y = 32.5;
   int z[] = {10,20,30};
   int *p = z;
   printf("%d %f %d %d %d %x %d\n", x, y, z[0],z[1],z[2],p,*p);
   //Write step 1 values in RAM

   myfun(p, &y);
   printf("%d %f %d %d %d %x %d\n", x, y, z[0],z[1],z[2],p,*p);
   //Write step 2 values in RAM

   (*p)=(*p) | z[1];
   printf("%d %f %d %d %d %x %d\n", x, y, z[0],z[1],z[2],p,*p);
   //Write step 3 values in RAM

   p++;
   (*p)--;
    printf("%d %f %d %d %d %x %d\n", x, y, z[0],z[1],z[2],p,*p);
   //Write step 4 values in RAM

    return 0;
}
```

Question 11 [4 points]: [Give it 15 minutes]

A) [1 point] What is the difference between a Structure and a Union? [*one line per term*]

B) [3 points] <u>Write</u> the `makefile` for a project that has the following files: `code1.c. code1.h. code2.c, code2.h, code3.c, code3.h, mainprogram.c`

i. Which Target(s) will get executed if you run `make`?

ii. Which Target(s) will get executed if you run `make mainprogram` after you ran `make`?

Question 12 [4 points]: [Give it 15 minutes]

A) [1 point] What is the difference between using Fork and using Multi-Threading in terms of the process ID and memory shared? [*one line per term*]

B) [3 points] Given the following code, **<u>write</u>** the following functions:

```
struct Node{
        struct Node* previous;
        int data;
        struct Node* next;
} ;
typedef struct Node Node;
Node* head = NULL;
Node* tail = NULL;
```

i. `void push(int data)` that creates a new node and insert it at index 1 (i.e. becomes the first node). It should connect the head pointer to it and connect itself to the next node. Make sure you update the tail pointer if this is the only node in the list.

ii. `int pop()` that returns the data value of the last node in the list and then deletes it. Make sure you update the head pointer if you pop the first node (i.e. only node in the list). Make sure you update the tail pointer after you pop a node.

GOOD LUCK

IT WAS A PLEASURE BEING WITH YOU IN THIS COURSE!!

bash Commands Sheet

bash variables:

- **`$SHELL`** stores the path to the current shell interpreter `/bin/bash`
- **`$0`** stores the name of the current shell interpreter `-bash`
- **`$SHLVL:`** stores the current shell level
- **`$BASH_VERSION`** stores the bash version
- **`$HOME`**: stores the name of your home directory.
- **`$USER:`** stores the user name of the current user
- **`$PATH:`** stores the paths to user program folders separated by colons (:)
  - You can add to the **`PATH`** in *.bashrc* : **`export PATH=$PATH:/path/to/dir1:/path/to/dir2`**
  - Or add to the end of the **`PATH`** variable in */etc/environment*

Set your own variables:

- To set a variable use [`var=val`], no space allowed on either side of the = sign.
- To access the variable value use $var or ${var} if there is no space after the variable name.
- **`echo:`** prints text or variables passed to it as an argument. [`echo "Hello Bash"`] [`echo $var`]
  - `today=Monday`
  - `echo This day is ${today}.  # or echo This day is $today.`
  - *Output:* `This day is Monday.`
- The **`export`** command [`export today`] will make the local shell variable *today* global.
- You can create expressions using **var=$((expression))** or **((var=expression))**
  - `total=$((3*37+12/2))` or `((total=3*37+12/2))`
  - `echo $total`
  - *Outputs* `117`
  - You can use +,-,*,/,% for calculating remainder, and **for power
- You can substitute commands using variable commands **$(command)**
  - `cp $(find . -name 'file*') files/`

Array Variables:

- **Indexed Array:** Use Integer indices:
  - `var=(val1 val2 val3)` to initialize the array,
  - `var+=(val4 val5)` to add new elements
  - `${var[i]}` to access array element *i*. The *i* variable can be negative to access the array from the end.
  - `${var[*]}` to retrieve all array elements
  - `${#var[*]}` returns the number of elements in the array.
  - In bash, arrays are sparse; they needn't be assigned with consecutive indices.
  - An element can be inserted at any index using `var[105]=newElement`
- **Associative Array:** Use String indices:
  - . Must be declared before assignment using the declare command with option -A [`declare -A var`]
  - `var=([ind1]=val1 [ind2]=val2)` to initialize the array
  - `${var[ind]}` to access an array element of index *ind*
  - `${var[*]}` to retrieve all array elements
  - `${#var[*]}` returns the number of elements in the array
  - You can also use `var[ind]=val` to set value *val* to array element of index *ind*

General commands:

- **`alias:`** creates an alias for a command with a set of options [alias lsa="ls -lha"], You can **`unalias`** if needed [`unalias lsa`] and You can list all aliases by not having any arguments [`alias`]
- **`cal:`** prints the calendar of the current month and **`date:`** print the current date and time
- **`whereis`**: locate the binary, source, and manual page files for a command [`whereis echo`]
- **`history`**: prints all commands that have been executed so far. [`history 10`] shows last 10 commands.
- **`!n`**: Execute command *n* from history. [`!1`] executes 1st command, [!-1] and [`!!`] execute last command.

Filesystem commands:

- **`pwd`:** prints the working/current directory. If you run a command, this is where the system will execute it.
- **`cd`:** change directories.
  - cd without any arguments will take you to your home directory. Similar to `cd /home/<username>` and `cd ~`
  - cd followed by two dots will take you to the parent directory. You can also use `cd ../../..` to go up 3 levels
  - \ = escape the next character. Useful when there is a space in the file/folder name. `cd file\ with\ space`
- **`mkdir`**: Make directory [`mkdir dir_name`]. Case Sensitive! "-p" option to create all parent dirs.
- **`rmdir`**: Remove *empty* dirs [`rmdir directory_name`] "-p" option to remove all parents.
- **`cp`**: Copy file [`cp file_to_copy place_to_copy_to`]. "-r" option to copy a directory
- **`mv`**: Move or rename a file or a directory [`mv old_location new_location`]
- **`rm`**: Remove a file [`rm file_to_remove`]. "-r" option to remove a dir ***empty or not***
- **`more`**: Read a file but you can only scroll down using enter key [`more file_to_read`]
- **`less`**: Read a file and you can only scroll up and down using direction keys. [`less file_to_read`]
- **`head`**: Display only the top *n* lines of a file. [`head -n10 file_to_read`]
- **`tail`**: Display only the last *n* lines of a file. [`tail -n10 file_to_read`]
- **`split`**: Split a file into pieces. Option "-b" specifies bytes per split, option "-l" specifies lines per split and option "-n" specifies number of splits. [`split -n3 or -l5 or -b9 file_to_split`].
- **`cat`**: Concatenate files or print a file if only one file argument is used. [`cat file1 file2`]. "-n" displays the line#
- **`tac`**: Concatenate and print files in reverse. . [`tac file1 file2`].
- **`paste`**: Merge text files showing them next to each other separated by TABs. [`paste file1 file2 file3`]
- **`cmp`**: Compare two files byte by byte and returns 0 if the files are the same; if they differ, the byte and line number at which the first difference occurred is reported. [`cmp file1 file2`]
- **`diff`**: Compare two files line by line. [`diff file1 file2`]
- **`wc`**: Counts number of lines, words and bytes (has extra byte for newline \n). [`wc file1`]
- **`touch`**: Update the access times of files and creates the file if it does not exist. [`touch file1`].
- **`dirname`**: prints the dir part of the pathname [`direname /path/to/file`] output:`/path/to`
- **`basename`**: prints just the part of the pathname after the last slash. [`basename /path/to/file.extension .extension`] output: `file`
- **`ln`**: create a hard link to a file, i.e another pointer to the same inode . [`ln file_name link_name`]
- **`ln -s`**: create a soft link to a file, i.e shortcut . [`ln -s file_name link_name`]
- **`find`**: search for files in a directory hierarchy [`find search_dir -name "pattern"`]
  - Use "." for the search_dir to search the working directory and "/" for the root directory.
- **`locate`**: find files by name on the whole disk searching the filenames database [`locate pattern`]
  - **`updatedb`:** updates the filenames database for the `locate` search
- **`ls`**: list the current directory [`ls`]. "`-a`'' : include hidden files & dirs, "`-l`'' : long format, "`-R`'' : list sub-directories recursively, "`-h`'' : human readable file sizes.

Wildcards:

- * = any sequence of zero or more characters
  - **`ls file*.txt`** *Output*: file10.txt file2000.txt file4.txt file.txt
- ? = any single character
  - **`ls file?.txt`** *Output*: file1.txt filen.txt fileR.txt
- **[abc]** = single character is **a** or **b** or **c**
  - **`ls file[15].txt`** *Output*: file1.txt file5.txt
- **[a-f]** = single character in range **a** through **f**
  - **`ls file[1-5].txt`** *Output*: file1.txt file2.txt file3.txt file4.txt file5.txt
- **{a,b,c}** = displays all options. You can have more than one {} in the command
  - **`echo file{1,5}.txt{6,7}.org`** *Output*: file<u>1</u>.txt<u>6</u>.org file<u>1</u>.txt<u>7</u>.org file<u>5</u>.txt<u>6</u>.org file<u>5</u>.txt<u>7</u>.org

File Permissions:

t rwx rwx rwx

- The first character [t]: **d** for directory, **l** for link, and **-** for file
- Permissions are:
  - **Read [r]**: can list contents of directory or open files for read-only
  - **Write [w]**: can add & remove files from directory or edit file content
  - **Execute (if it's a program) [x]**: can access files in directory or execute files
  - Permissions are set for three categories of users: file's **owner**, users in the file's **group**, and **others**
- We can change the permissions of a file/dir using the **`chmod`** command [`chmod permissions file`]
- Binary permissions notation: [`chmod 653 filename`] results in *rw- r-x -wx*

Read	**Write**	**Execute**
4	2	1

- Symbolic permissions notation: [`chmod u=rw,g=rx,o=wx filename`] results in *rw- r-x -wx*

Read : r,	Write : w,	Execute : x	
User : u,	Group : g,	Others : o,	All : a
Add permission : "+",	Remove permission : "-",	Set permission : "="	

- **chown** command changes file/dir ownership [`chown new_user:group_of_new_user filename`]
- **chgrp** works the same way as `chown` but to only change the group. [`chgrp new_group filename`]
- Use the *-R* option to apply the permission/ownership change on all subdirectories and files of a directory.

File Manipulation

- **`grep:`** searches the named input FILEs for lines containing a match to the given PATTERN
  - Use the "-e" option to specify the PATTERN to search for inside the files
  - Use the "-n" option to print the line number
  - Use the "-i" option for case insensitive matching
  - Use the "-v" option to select non-matching lines
  - Use the "-l" option to only show the file names without the matching line.
  - Example: Search files with names that has *.txt* and *.conf* in it for lines containing the words *file* or *line.*
    - `grep -e file -e line  *.{txt,conf}`
- **`awk:`** a pattern scanning and processing language for shaping the data. The "-F" option specifies the delimiter.
  - Example: Show the second column followed by Tab "\t" then the first column
    - `awk '{print $2 "\t" $1}' simple_data.txt`
- **`sort`**: sorts lines of text files. Default is sorting alphabetically based on the first word of each line.
  - The *-k* option defines which column to use for sorting.
  - The *-n* option tells sort to handle strings as numbers
  - The *–u* option outputs only unique lines
  - Example sort based on the second column, handle values as numbers and show unique lines
    - `sort -k2nu simple_data.txt`
- **`sed:`** is a command for modifying data from a file or stream. To replace all occurrences of a substring in a file:
    - `sed s/substring/replacement/ filename`

Pipes and I/O Redirection:

- Pipes | takes the output of one command and send it to another as input. [`echo "hello" | wc`]
- **`xargs`:** Execute a command on multiple files
    - Example: Compress all files with *.txt* and *.conf* extensions and have the words "*file*" and "*line*" in them to a single archive named *matching.tgz*
        - `grep -l -e file -e line *.{txt,conf} | xargs -d'\n' tar -czvf matching.tgz`
- xargs with option "-n" and no commands to execute limits the output per line [`echo a b c d | xargs -n2`]
- Redirecting **standard output** from screen to a file using the **1>** or > sign to overwrite and 1>> or >> to append.
    - `echo "Hello standard output redirection" > outputFile`
- Redirecting **standard error** from screen to a file using the **2>** sign to overwrite and 2>> to append:
    - `ls unreal 2> errorFile`
- Redirecting both **standard output & standard error** from screen to the same file using the **&>** sign to overwrite and &>> to append
    - `ls unreal &> outputErrorFile`
- Redirecting **standard error** from screen to **standard output** and redirecting **standard output** from screen to a file using the **2>&1** sign at the end of a command to redirect the standard error to the same file the standard output was redirected to.
    - `ls unreal >> outputErrorFile 2>&1`
- Get rid of output by redirecting standard output or standard error or both to **/dev/null**
    - `ls *.c 2> /dev/null #Get rid of standard error`
- Redirecting **standard input** from keyboard to a command using the <(command) :
    - No space between the < sign and the opening parentheses "("
    - `diff <(ls myprog_ver1/) <(ls myprog_ver2/)`
- A specialized form of input redirection uses the << sign to enter the contents on a command line.
    - You follow the << sign with a word; when that word appears alone at the start of an input line, the shell recognizes it as the end of the input. [`sort <<:q`]
- combine I/O redirections and pipes:
    - `sort < file.txt > sorted_file.txt 2> error_file.txt | cat sorted_file.txt`
    - Take the input from file.txt, do the sort and write the output to sorted_file.txt and write any error to error_file.txt, the display the sorted_file.txt using the cat command.

String Manipulation

- **${#string}:** Print string length.
  - var=cisc220
  - echo ${#var}
  - *Output*: 7
- **expr index $string $substring:** Prints the index of the first occurance of the *$substring* in the *$string*
  - *Don't forget the $ sign before your variable*
  - var=cisc220
  - expr index **$var** '220'
  - *Output*: *5*
- **${string:position:length}** : Extracts substring from $*string* at $*position* and with optional length *$length*.
  - var=cisc220
  - echo ${var:4:2}
  - *Output*: *22*
  - echo ${var:(-4)}
  - *Output*: c220
- **${string#substring}**: Deletes shortest [and ## for longest] match of $*substring* from **front** of $*string*.
  - **${string%substring}** deletes the shortest [and %% for longest] match of *$substring* from the **back**.
  - var=cisc220
  - echo ${var#"cisc"}
  - *Output*: *220*
- **${string/substring/replacement}**: Replace **first** match of $*substring* with $*replacement*.
  - Use ${string//substring/replacement} to replace **all** matches.
  - **${string/#substring/replacement}**: If $*substring* matches **front** end of $string, substitute $*replacement* for $*substring*.
  - **${string/%substring/replacement}:** If $*substring* matches **back** end of $string, substitute $*replacement* for $*substring*.
  - var=cisc220
  - echo ${var/"220"/"124"}
  - *Output:* cisc124
  - echo ${var/#"220"/"124"}
  - *Output:* cisc220 # didn't work as the string does not start with 220
  - echo ${var/%"220"/"124"}
  - *Output:* cisc124

Scripts:

- For bash shell scripts, the *first line in the script file* must be: **`#!/bin/bash`**
- To make the script file executable, run the following command: `chmod u+x scriptFile.sh`
- To run/execute the script file, type dot slash "./" before the script file name [`./scriptFile.sh`]
- To keep any changes to shell state that happen within a script after the script exists, Use `source scriptFile.sh args` instead of `./scriptFile.sh args` to run the script.
- Scripts can take arguments, which become shell parameters
  - **$0**: name of command/script
  - **$#**: number of arguments
  - **$1, $2**,...: the individual arguments
  - **$***: all the arguments in one string, separated by spaces
- The **shift** *n* command shifts the variables left by n. For example, With 4 arguments, if we run shift 2, $1 takes on the old value of $3, $2 the old value of $4, then $3 and $4 are unset, and $# is now 2 instead of 4.
- Exist status **0** means success, else means failure. Use **`exit n`** command to specify the script exist status. Shell parameter **$?** stores the exit status of the last command executed.

Read User Input:

- **read** command reads a line from the standard input and places it in variables.
- With several variables, the 1st word is placed in the 1st variable, the 2nd in the 2nd variable, and so on to the second-last variable; the rest of the line is placed in the last variable.
- The **-p** option prints a message before asking for input.
- The **-s** option makes the user input secure/not visible, typically used with password.
  - `read -p "enter your password, hit return to end:" -s var`
- The **-n** option sets the number of character user can input. If not used, the input ends when ENTER is hit.
  - `read -p "enter your 10 char input:" -n10 var1 var2 var3`
- The **-a** reads the data word-wise into the specified array instead of normal variables.
  - `read -p "enter elements separated by space, return to end:" -a varArray`

if Conditionals

- **With test commands**

```
if test-commands ; then
commands ;
[ elif test-commands ; then
commands ;]*
[ else
commands ; ]
fi
```

[] means optional
* means zero or more
; splits commands written on the same line
Code **indentation** does **NOT** affect the meaning

- **With Arithmetic expression inside (( ))**

```
if (( $var1+$var2 > $var3 )) ; then
commands ;
[ elif (( $var1+$var2 != $var3 )); then
commands ;]*
[ else
commands ; ]
fi
```

You can make numerical comparisons inside double-parentheses using <=, >= , <, >, and !=

- **With Boolean expression inside [[ ]]**

```
if [[ $var1==$var2 || $var3!=$var4 ]] ; then
commands ;
[ elif [[ $var5<$var6 && $var7 -gt $var8 ]] ; then
commands ;]*
[ else
commands ; ]
fi
```

- **MUST** have spaces before and after the double square brackets [[ ]].
- Expressions can be combined with parentheses, && (and), || (or), and ! (not).
- To compare strings use ==, <, and > (but not <= or >=).

- **File Boolean Expressions:** The following file expressions may be used inside the if [[..]]
  - ***-e file*** File exists
  - ***-f file*** File exists and is a regular file
  - ***-d file*** File exists and is a directory
  - ***-h file*** File exists and is a symbolic link
  - ***-r file*** File exists and is readable
  - ***-w file*** File exists and is writable
  - ***-x file*** File exists and is executable
  - ***file1 -nt file2*** File1 exists and is newer than file2, or file2 doesn't exist
  - Example: Make a directory if it doesn't exist:
    ```
    if ! [ -d $1 ] ; then
    mkdir $1
    fi
    ```

Conditional Commands
- You can have a command conditionally executed if another command succeeds or fails.
- The command after the AND operator (&&) is executed if the previous command is successful.
  - Example: check if a directory exists and cd into it if it exists: `[ -d $1 ] && cd $1`
- The command after the OR operator (||) is executed if the previous command fails.
  - Example: check if a directory exists and create it if it doesn't: `cd $1 || mkdir $1`
  - You can combine conditional commands. Example: Create a directory, cd to it and echo "success". If either mkdir or cd fails, echo "Error":
    - `mkdir $1 && cd $1 && echo "Success" || echo "Error"`

case
- A case statement compares a variable against one or more patterns and executes the commands associated with that pattern.

```
case $var in
*patt?rn[1-9]) commands ;;
*) commands ;;
esac
```

- The patterns use wildcards (* and ?) and character lists and ranges ([...]) extensively.
- Default pattern is the one gets executed if none of the other patterns match [optional]
- Default pattern is *)
- Cases end with ;; don't forget them!!!!

for Loop

```
for name in words ; do
[commands ;]*
done [redirections]
```

- The ***words*** can be any sequence of items separated by **spaces**
- The commands get executed once for each ***word*** in the sequence; where ***$name*** refers to the current ***word***.
- Command substitution **$(...)** can be used to generate ***words***.

- Echo the names of all the files in the current directory.
```
for X in *; do echo $X; done
```
- Echo the names of files with names starting with *s* **OR** ending in *.sh* in the current directory;
```
for X in s* *.sh; do echo $X; done
```
- Echo all the parameters of the current script.
```
for PARAM in $*; do echo $PARAM; done #Equivelant to: for PARAM; do echo $PARAM; done
```
- Copy a file to IPs 192.168.0.1, 192.168.0.5 and 192.168.0.200.
```
for var in 1 5 200; do scp file1.txt khalifa@192.168.0.$var:/home/khalifa; done
```
- Echo odd numbers from 1 to 10.
```
for X in $(seq 1 2 10); do echo $X; done
```
- Echo numbers from 1 to 9.
```
for n in {1..9}; do echo $n; done
```

Java-like for Loop

```
for (( n=1; n<=10; n++ )); do echo $n; done
```

- The first expression "n=1"is evaluated when first encountered.
- The second "n<=10" is a test.
- The third "n++" is evaluated after each iteration

while Loop

```
while test-commands ; do
[commands ;]*
done [redirections]
```

- Example: script to read line by line from a file **using input redirection to the while loop**:

```
while read -r line   # read user input. The -r option to skip parsing input
do
      echo $line
done <$1         # read from file [name in first argument] instead of keyboard
```

Loop Control

- **`break`** command forces a loop to exit. With a numeric argument, break can exit multiple nested loops.
  - Example: Read user input till entering :q

```
while true; do
    read x
    [ $x == ':q' ] && break
done
```

- **`continue`** command immediately starts a new iteration of the loop, bypassing any remaining commands.

Functions

- A shell *function* is a grouping of commands within a shell script.
- Functions are invoked in the same way as regular commands.
  - **functionName arg1 arg2**
- The command-line parameters $1, $2, and so on receive the function's arguments, temporarily hiding the global values of $1, $2, and so on. $0 remains the name of the full script.
- Use the **`echo`** command to return the function output and the **`return`** command to return the exit status.

```
functionName () {
  [commands ;]*
} [redirections]
```

```
function functionName {
    [commands ;]*
} [redirections]
```

Debugging

- **Full Debugging:**
  - Run your script using **`bash -x script.sh`** instead of *./script.sh*
  - OR use *./script.sh* but edit the first line in the script to be `#!/bin/bash -x`
  - Commands and their arguments are printed as they are executed. The + sign shows the command and others lines without the + sign present the command's output

```
+ echo "Hello bug"
Hello bug
```

- **Partial Debugging:**
  - Display debugging information only for troublesome zones
  - Use set -x command inside your script to mark the beginning of the debugging zone
  - Use set +x command inside your script to mark the end of the debugging zone
  - Run your script normally *./script.sh*
  - You can switch debugging mode on and off as many times as you want within the same script.

```
set -x                  # activate debugging from here
echo 'debugged'
set +x                  # stop debugging from here
```

Compression:

```
tar czvf archiveName.tgz file1 file2 dir1/ dir2/ file3
c: create , z: compress using gzip, v: verbose, f: into file
```

Decompression:

```
tar xzvf archiveName.tgz
x: decompress
```

Send backup to a backup server:

```
scp archiveName.tgz username@backupIP:/path/to/backup
```

Github:

- Clone a repository: `git clone <Your Remote Repository URL> <Name of your local folder>`
- Add file to local repository: `git add README.md`
  `git commit -m 'my first hello commit'`
- Add file to remote repository: `git push origin master`
- Get changes done by others from the remote repository: `git pull`
- see changes: `git log -p README.md`
- create tag v1.1: `git tag v1.1`
- list all tags: `git tag`
- create branch branch1: `git branch branch1`
- list of all branches: `git branch`
- Switch between branches: `git checkout branchname`
- Merge branches *feature1* and *feature2* into your current branch: `git merge feature1 feature2`
- view difference between two commits: `git difftool commit1ID commit2ID`

Package Management:

- List all installed software packages: `apt-cache pkgnames`
- Check package dependencies: `apt-cache showpkg <package>`
- Update the list of available software: `sudo apt-get update`
- Upgrade to the latest software version: `sudo apt-get upgrade`
- Upgrade and add or remove packages to fulfill the upgrade dependencies: `sudo apt-get dist-upgrade`
- Install a specific packages: `sudo apt-get install <package>`
- Install specific package version: `sudo apt-get install <package>=<version>`
- Remove packages, keeping configuration: `sudo apt-get remove <package>`
- Completely remove package with its configuration: `sudo apt-get purge <package>`
- Remove setup files: `sudo apt-get autoclean`
- Download Source Code of a package: `sudo apt-get source <package>`
- Download, Unpack and Compile a Package: `sudo apt-get --compile source <package>`
- Check broken dependencies: `sudo apt-get check`
- Install dependencies: `sudo apt-get build-dep <package>`
- Remove packages that were installed to satisfy dependencies for other packages and but they are now no longer required: `sudo apt-get autoremove`
- Install .deb setup file manually: `dpkg -i <.deb filename>`
- List all the installed packages: `dpkg -l`
- Remove a package: `dpkg -r <package name>`
- Purge a package with its configuration: `dpkg -p <package name>`
- Check a package is installed or not: `dpkg -s <package name>`
- Check where a packages was installed: `dpkg -L <package name>`
- Check dependencies: `ldd /bin/bash`

Linux devices:

- List devices: `ls -l /dev`
- Get information about the active, mounted, non-swap partitions, their size, their free space and their mount points: `df` [`df .` shows information about the partition the current directory belongs to].

```
Filesystem                       Size  Used Avail Use% Mounted on
udev                             982M     0  982M   0% /dev
tmpfs                            201M  3.3M  197M   2% /run
/dev/mapper/cisc220--vg-root      18G  2.6G   14G  16% /
/dev/sda1                        472M  194M  255M  44% /boot
```

- Print all partitions in a tree-like fashion: `lsblk`

```
NAME                       MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                          8:0    0 20.1G  0 disk
├─sda1                       8:1    0  487M  0 part /boot
├─sda2                       8:2    0    1K  0 part
└─sda5                       8:5    0 19.7G  0 part
  ├─cisc220--vg-root       252:0    0 17.7G  0 lvm  /
  └─cisc220--vg-swap_1     252:1    0    2G  0 lvm  [SWAP]
```

- Show directory sizes: `du` [use the `-a` option to show sizes of files also, not only directories]
- Copy binary blocks of data: `dd if=<source> of=<target> bs=<block size> count=<blocks count>`
- Manipulate the disk partition table: `sudo fdisk <device name>` [use `n` to create new partition and `d` to delete]
- List partitions [active data, swap and RAM devices]: `sudo fdisk -l`
- Format a partition: `sudo mkfs.ext3 <partition name>`
- Mount a partition: `mount -t type <partition name> <mounting point folder name>`
  [**vfat**: Windows FAT filesystem, **ntfs**: Windows ntfs filesystem, **iso9660**: CD-ROM filesystem, **ext3**: Linux filesystem]
- Unmount a device: `umount <partition name>` or `umount <mounting point folder name>`
- Snapshot of system memory usage [Both RAM and Swap space]: `free -h`

```
        total    used     free   shared  buff/cache   available
Mem:     2.0G     61M     1.1G     3.3M        766M        1.7G
Swap:    2.0G      0B     2.0G
```

- Live feed of the system memory usage [Both RAM and Swap space]: `top`

```
top - 14:10:32 up  3:34,  5 users,  load average: 0.00, 0.00, 0.00
Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2041448 total,  1200492 free,    62760 used,   785196 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1781332 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   37796   5932   4064 S  0.0  0.3   0:03.61 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
```

- Check if you already have swap space: `swapon -s`
- Create an empty 8GB swap file: `dd if=/dev/zero of=swapfile bs=8M count=1024`
- Set up the swap file: `mkswap ./swapfile`
- Enable swap file manually: `sudo swapon swapfile`

Linux Processes:

- Lists all processes started from the current shell: `jobs`
- Suspend a process: `control-z`
- Stop a foreground process: `control-c`
- Resuming a process in the foreground: Type the percentage `%` sign followed by the suspended process number.
- Resuming a process in the background: Use the `bg` command followed by the percentage `%` sign and the suspended process number.

```
vim file4.txt
control-z                                   # The control-z to suspend the foreground vim process background
[1]+  Stopped    vim file4.txt
vim file7.txt &                             # The & sign to make the process run in the background
[2] 16173
jobs
[1]   Stopped    vim file4.txt
[2]+  Stopped    vim file7.txt
%1                                          # The %1 resumes the first suspended process [bring it to foreground]
vim file4.txt
control-z
bg %1                                       # Resume the first suspended process [bring it to background]
vim file4.txt &
```

- Print a snapshot of the processes currently running on the system: `ps`
  - `ps u` lists processes owned by the current user
  - `ps a` lists processes owned by any user
  - The `ps` command is usually used with options `aux` [`ps aux`] (no dash before the options)
  - The `ps axo` allows you to list the fields to display:
    `ps axo user,pid,ppid,pcpu,pmem,tty,start,time,comm`

```
ps u

USER      PID  %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
khalifa   1213  0.0  0.2  22976  5832 tty1     Ss   11:37   0:00 -bash
khalifa   1265  0.0  0.3  23680  6548 pts/0    Ss   11:38   0:00 -bash
```

- Find a process ID: `ps aux | grep <command or script name>`
- Kill a process: `kill -9 <process ID>`

Linux Daemons:

- List available daemons: `service --status-all`
- show the status of a single service: `service <daemon name> status`
- Starting a daemon: `service <daemon name> start`
- Stopping a daemon: `service <daemon name> stop`
- Restarting a daemon: `service <daemon name> restart`

Linux Runlevels:

- The standard Linux kernel supports **<u>Seven</u>** runlevels:
  - 0 - System halt; Shutdown.
  - 1 - Single user; usually used to do repairs.
  - 2 - Multiple users, no NFS (network filesystem); also used rarely.
  - 3 - Multiple users, command line (i.e., all-text mode) interface; the standard runlevel for most servers.
  - 4 - User-definable
  - 5 - Multiple users, GUI; the standard runlevel for most desktop systems.
  - 6 - Reboot; used when restarting the system.

- Each runlevel has a number of scripts located in the /etc/rc#.d/ directory (where # is the runlevel). For example, for runlevel 3, scripts are in the /etc/rc3.d/ directory.
  - Scripts with names starting with a "S" refers to a script that will start a process.
  - Scripts with names starting with a "K" refers to a script that will kill a process.
  - The number after the first letter in the script name specifies the order to run the script.
- Change the current runlevel: `init <new runlevel number>`
- Show current and previous runlevels: `runlevel`

Cron jobs:

- minuteOfHour hourOfDay dayOfMonth month dayOfWeek command
  - minuteOfHour : between '0' and '59'
  - hourOfDay: between 0 and 23 (0 is midnight)
  - dayOfMonth : between 1 and 31
  - month: numeric between 1 [Jan] and 12 [Dec], or as the name of the month (e.g. May)
  - dayOfWeek: numeric between 0 [Sunday] and 6 or as the name of the day (e.g. sun).
  - command: This is the command that to run. This field may contain multiple words.
- If you don't wish to specify a value for a field, just place a * in the field.
- cron supports 'step' values using */step
  - A value of */2 in the dayOfMonth field would mean the command runs every two days.
- cron accepts lists in the fields. Lists can be in the form:
  - 1,2,3 (meaning 1 and 2 and 3)
  - 1-3 (also meaning 1 and 2 and 3).

Downloading files:

- Using one thread: `wget <url>`
- Using multiple threads: `axel -n<number of threads> <url>`

Networking:

- See information about **all active** interfaces: `ifconfig`
- See information about **all active and inactive** interfaces: `ifconfig -a`
- See certain interface information: `ifconfig <interface>` `[ifconfig eth0]`
- Shutdown an interface: `ifconfig <interface> down [ifconfig eth0 down]`
- Start an interface: `ifconfig <interface> up[ifconfig eth0 up]`
- Assign **a static IP** to an interface:
```
ifconfig <interface> <ip-address> netmask <subnet-mask>
ifconfig eth0 10.0.0.102 netmask 255.255.255.0
```
- Restart the **software managing networking**: `sudo service networking restart`
- Find IP using domain name: `nslookup <domain-name>`
- Finding domain name using IP: `nslookup <ip-address>`
- View the current routes known to your computer: `route`
- Add a new route:
```
route add -net <destination-IP> netmask <destination-netmask> <interface>
route add -net 192.56.76.0 netmask 255.255.255.0 eth0
```
- Add a new route using the gateway IP:
```
route add -net <destination-IP> netmask <destination-netmask> gw <gateway-IP>
route add -net 192.56.76.0 netmask 255.255.255.0 gw 10.0.0.1
```
- Add a default route **[destination: 0.0.0.0]**, which will be used if no other route matches:
```
route add default gw <gateway-IP>
route add default gw 10.0.0.1
```
- Delete a route [Type route delete followed by the parameters you used to add the route]:
```
route delete -net <destination-IP> netmask <destination-netmask> <interface>
route delete -net 192.56.76.0 netmask 255.255.255.0 eth0
```

- Verify that a given address is actually reachable: `ping <domain name>`
- See all the gateways between your computer and another domain: `traceroute <domain name>`
- Translates between IP Address and local hardware addresses (MAC Address): `arp`

```
Address          HWtype   HWaddress           Flags Mask      Iface
192.168.0.101    ether    68:94:23:16:dd:2d     C             enp0s3
192.168.0.1      ether    c4:e9:84:87:b1:f8     C             enp0s3
```

- Add an entry in the ARP table: `arp -s <ip-address> <mac-address>`
  `arp -s 192.168.0.2 68:94:23:16:dd:22`
- To remove and entry from the ARP table: `arp -d <ip-address>`
  `arp -d 192.168.0.2`
- Display sockets: `sudo netstat -atnp`

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address      Stat   PID/Program name
tcp      0      0   127.0.0.1:3306     0.0.0.0:*            LISTEN      1065/mysqld
tcp      0      0   0.0.0.0:22         0.0.0.0:*            LISTEN       1048/sshd
tcp      0    464   192.168.0.103:22 192.168.0.101:25894 ESTABLISHED 2407/sshd
tcp6     0      0   :::80               :::*                LISTEN     1182/apache2
```

Firewall:

- The Linux ***iptables*** firewall splits rule into three chains:
  - INPUT: incoming traffic
  - FORWARD: traffic generated on this machine and going to this machine [localhost]
  - OUTPUT: outgoing traffic
- <u>**List**</u> all firewall rules in the three chains: `sudo iptables -L`
- <u>**Add**</u> a rule: `sudo iptables` followed by one or more of the following options:
  - The "-A *chain-name*" option appends a new rule to a rule chain. [-A INPUT]
  - The "-I *chain-name rule-index*" option lets you insert a rule in a specific place.
    [-I INPUT 5] insert rule into INPUT chain at 5th place.
  - The "-p *protocol*" option specifies the connection protocol used [-p tcp or -p udp].
  - The "-s *ip-address*" option specifies the source IP address[/mask]. [-s 10.0.0.100/32]
  - The "-d *ip-address*" option specifies the destination IP address[/mask] . [-d 10.0.0.100/32]
  - The "--sport *port-number*" specifies the source port(s) [A single port or a range given as start:end]. Must be proceeded with the "-p" option [-p TCP --sport 22]
  - The "--dport *port-number*" specifies the destination port(s) [A single port or a range given as start:end]. Must be proceeded with the "-p" option [-p TCP --dport 22]
  - The "-j *target*" option sets the rule target. [-j ACCEPT]
    - ACCEPT - Accept the packet
    - REJECT - Reject the packet and notify the sender
    - DROP - Silently ignore the packet and do not notify the sender
- <u>**Delete**</u> a rule: `sudo iptables -D <chain-name> <rule-index>`
  - Example: `sudo iptables -D INPUT 1`

Examples:

- Append a rule to allow SSH TCP port 22: `sudo iptables -A INPUT -p TCP --dport 22 -j ACCEPT`
- Insert a rule at index 1 to block access to your website from IP 10.0.0.55.
  `sudo iptables -I INPUT 1 -p TCP --dport 80 -s 10.0.0.55/32 -j DROP`
- Usually you have a DROP rule at the end of your firewall chains to force dropping any traffic that you did not explicitly allow. `sudo iptables -A INPUT -j DROP`
- Block facebook.com access from your computer: `sudo iptables -A OUTPUT -d facebook.com -j DROP`

User Accounts Management

- **List** all User Accounts: `cat /etc/passwd`
- **Add** a User Account: `adduser <username>` #Example: `adduser shadi`
- **Switch** between users: `su <username>` #Example: `su shadi`
- **Change password**: `passwd <username>` #Example: `passwd shadi`
- **Lock** user's password: `passwd -l <username>` #Example: `passwd -l shadi`
- **UnLock** user's password: `passwd -u <username>` #Example: `passwd -u shadi`
- **Delete** a User Account: `deluser <username>` #Example: `deluser shadi`
- **Give superuser permission**: `sudo vim /etc/sudoers` and add the following:

```
username Approved_Terminal=(ACT_AS_User:Group) Approved_Commands
                    shadi ALL=(ALL:ALL) ALL
```

Monitoring Users Activity

- Report the time since booted and the number of current users logged in: `uptime`
- reports the usernames of the logged in users: `users`
- reports the usernames of the logged in users, login time and connection: `who`
- A combination of **uptime** and **who** commands: `w`

C Code Sheet

- **Compile source file:** `gcc -Wall -c mycode.c` generates `mycode.o` object code file
- **Link object files:**
  `gcc -Wall -o executable-name objectfile1.o objectfile2.o` generates the executable program `executable-name` from object code files `objectfile1.o objectfile2.o`
- **Compile and Link:** `gcc -Wall -o executable-name mycode.c` generates `executable-name` executable file
- **Running C program:** `./executable-name`
- **Makefile:**
  - A makefile rule is usually split on two lines:
    ```
    Target : Dependencies
    <TAB> Command to build Target
    ```
  - But can be also written in one line
    ```
    Target : Dependencies; Command to build Target
    ```
    - **Example**:
      ```
      program.o: program.c; gcc -Wall -c program.c
      program: program.o calc.o
              gcc -Wall -o program program.o calc.o
      ```
  - Run **make** to execute only the **first Target** in the **makefile** in **current directory**.
  - **make -f makefile2** execute the **first Target** in the **makefile2** in **current directory**
  - **make calc.o** executes a specified Target [calc.o]
  - **make -n** doesn't run commands, just prints them
  - **make -B** assumes all files have changed (a "full Build")

- **Command-line Arguments:** `int main( int argc, char *argv[] )  {}`
  - **argc**: the number of command-line arguments
  - **argv**: String array of the command-line arguments
  - The first **argv** element (**argv[0]**) is the name of the program

Pre-Processor Directives

#include: Inserts a particular header "library" from another file. Example: `#include <stdio.h>`

`#include <filename>`: look in `/usr/include/` for a library provided by the C language.
`#include "filename"`: look in the current directory for a library normally created by you.

#define: Create a preprocessor macro [aka alias or constant]

- **Value macro [constant]:** `#define MAX_ARRAY_LENGTH 20`

```
printf("Max Array length is %d\n", MAX_ARRAY_LENGTH );
Replaced with printf("Max Array length is %d\n", 20);
```

- **Function macro:** `#define ADD(x,y) (x+y)`

```
printf("1+2 is %d\n", ADD(1, 2));
Replaced with printf("1+2 is %d\n", (1+2));
```

- **Predefined macros**

```
printf("Date :%s\n", __DATE__ ); // current date "MMM DD YYYY"
printf("Time :%s\n", __TIME__ ); // current time "HH:MM:SS"
printf("File :%s\n", __FILE__ ); // current filename i.e hello.c
printf("Line :%d\n", __LINE__ ); // current line number i.e 4
```

#ifndef: Check if macro is not defined

```
#ifndef MESSAGE
   #define MESSAGE "Message"
#endif
```

#ifdef: Check if macro is defined. Useful to implement different "modes", for example debugging and non-debugging mode, or linux and Windows mode.

```
#ifdef DEBUG
   /* Your debugging statements here */
#endif
```

#undef: Undefine a defined macro

```
#ifdef FILE_SIZE
   #undef FILE_SIZE
#endif
```

Variables

- **Definition:** `data-type variable1, variable2, …;`
  - Example: `int i, j, k;`
- **Initialization:** `variable1=value1;`
  - Example: `i = 5;`
- **Variables can be initialized in their definition:** `data-type variable1 = value1;`
  - Example: `int d = 3, f = 5;`
- **Data Types:**

```
– int or long           4 bytes [ -2,147,483,648 to 2,147,483,647]
– unsigned int or long 4 bytes [0 to 4,294,967,295]
– short                      2 bytes     [-32,768 to 32,767]
– unsigned short        2 bytes     [0 to 65,535]
– float                  4 byte [1.2E-38 to 3.4E+38]      6 decimal places
– double              8 byte [2.3E-308 to 1.7E+308]       15 decimal places
– long double          10 byte [3.4E-4932 to 1.1E+4932] 19 decimal places
– char                                1 byte      for a single character
– void              sometimes means nothing and sometimes means anything
```

  - No String and No Boolean!
  - Use the `sizeof(type)` Function to find the number of bytes used by a type.
    - Example: `sizeof(int); //returns 4`
  - Calculate a type's range: $2^{(\mathrm{sizeof(type)} * 8)} - 1$ (divide by 2 for signed types)
  - **Division**:
    - between **two ints**: result is an **int**
    - between **int & float** or **two floats**: result is a **float**
    - use "(float)" to cast an int into a float

```
int i =5;
float f = (float) i;
```

  - **Boolean:** Use Integers instead: `0 = false` and `non-zero = true`

- **Global variables:** Defined outside all functions. Use the keyword `extern` before the variable definition to access a global variable defined in another file. Example: **`extern`** `int errno ;`
- **typedef:** give an alias to a variable. Example: **`typedef`** `char* string;`

Arrays

- **Declaring Arrays:** *`data-type`* `arrayName [ arraySize ];`
  - Example: `double balance[10];`
- **Initializing Arrays**
  - *`data-type`* `arrayName [ arraySize ] = {val1, val2, val3, …};`
    - Example: `double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};`
  - *`data-type`* `arrayName [] = {val1, val2, val3, …};`
    - Example: `double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};`
- **Assign Values:** `arrayName [index] =value;`
  - Example: `balance[4] = 50.0;`
- **Accessing Array Elements:** `arrayName [index];`
  - Example: `double salary = balance[9];`
- **Multidimensional Arrays:** *`data-type`* `arraynName[size1][size2]...[sizeN];`

```
int a[2][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7}   /*  initializers for row indexed by 1 */
};
```

- OR `int a[2][4] = {0,1,2,3,4,5,6,7};`

Strings

- **Strings are one-dimensional array of characters terminated by a null character** `'\0'`.
  - `char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};`
  - `char greeting[] = "Hello";`
    - you do not place the *null* character at the end of a string constant. The C compiler automatically places the `'\0'` at the end of the string when it initializes the array.

#include<string.h>

- **String Functions:**
  - `strcpy(s1, s2);` Copies string `s2` into string `s1`.
  - `strcat(s1, s2);` Concatenates string `s2` onto the end of string `s1`.
  - `strlen(s1); Returns the length of string s1.`
  - `strcmp(s1, s2);`
    - Returns 0 if `s1` and `s2` are the same
    - less than 0 if `s1<s2`
    - greater than 0 if `s1>s2`

• #include<stdlib.h>

- `Use int atoi(char* str);` function to convert a string to integer
- `Use float atof(char* str);` function to convert a string to float

```
char str[10];
scanf("%s",str);
int n = atoi(str);
float f = atof(str);
```

Operators

- **Arithmetic Operators:** +, -, *, /, %, ++, --
- **Relational Operators:** ==, !=, >, <, >=, <=
- **Logical Operators:** &&, ||, !
  - If the operand is non-zero, then it is counted as True
  - If the operand is zero, then it is counted as False
- **Bitwise Operators:** View an integer as a string of Booleans
  - `A & B` bitwise AND [1 if both 1, otherwise 0]
  - `A | B` bitwise OR [0 if both 0, otherwise 1]
  - `~A` bitwise NOT [0 if 1 and 1 if 0] [when converted to decimal, negative of the number plus 1]
    - Example: ~4 = -5
  - `A ^ B` bitwise XOR [1 if both different, 0 if both same]
  - `A >>` n bitwise Shift Right by n bits
  - `A <<` n bitwise Shift Left by n bits

```
128  64  32  16  8  4  2  1
 |    |   |   |   |  |  |  |
 1    0   0   1   1  0  1  1
----------------------------
128+0 + 0 +16+8+0+2+1
        = 155
```

Decision Making

- **IF Condition:**

```
if(boolean_expression) {
   /* statement(s) if true */
}
else {
   /* statement(s) if false */
}
```

```
int x=5;
if(x>0) {
   printf("greater than 0");
}
else {
 printf("less than or equal 0");
}
```

- **Switch Case:**

```
switch(expression) {
   case constant-expression  :
      /*statement(s);*/
      break;
   default : /* Optional */
   /*statement(s);*/
}
```

```
char grade = 'B';
switch(grade) {
  case 'A' :
     printf("Excellent!\n" );
     break;
  case 'B' :
  case 'C' :
     printf("Well done\n" );
     break;
   default :
     printf("Invalid grade\n" );
   }
```

Loops

- **For Loop:**

```
for ( init; condition; increment ) {
   statement(s);
}
```

```
int a;
for( a=10; a<20; a=a+1 ){
 printf("value of a: %d\n", a);
}
```

- **While Loop:**

```
while(condition) {
   statement(s);
}
```

```
int a=10;
while( a<20 ){
 printf("value of a: %d\n", a);
 a=a+1;
}
```

- **Do...While Loop:**

```
do {
   statement(s);
} while( condition );
```

```
int a=10;
do {
 printf("value of a: %d\n", a);
 a=a+1;
} while( a<20 );
```

- **Break:** When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- **Continue:** When a continue statement is encountered inside a loop, the current iteration is immediately terminated and the next iteration starts.

Input Output
#include<stdio.h>

- **Output:**

```
printf("Message",parameter1, parameter2, …);
```

- In the Message, you can specify the types of parameters:
    - `%d`: integer (d for "decimal", not "double"!)
    - `%u`: unsigned int
    - `%ld`: long
    - `%lu`: unsigned long
    - `%f`: float or double
    - `%c`: single character
    - `%s`: string
    - %x: hexadecimal
- In the Message, you can specify the length of parameters:
    - `%nT` where `n` is the *number of characters* and `T` is the *parameter type.*
- In the Message, you can specify the precision for float and double parameters:
    - `%n.pT` where `n` is the *number of characters*, `p` is the *number of characters after the decimal point* and `T` is the *parameter type.*
- In the Message, use "\n" for new line.

```
printf("Hello %10s\n","Shadi");  // outputs: Hello      Shadi
printf("You have $%.2f\n",50.123456);  // outputs: You have $50.12
printf("You have $%10.2f\n",50.123456);  // outputs: You have $      50.12
printf("You have $%ld\n",50.023456);  // Wrong outputs: You have $1
```

- **Input:**

```
scanf("Input Types",parameter1, parameter2, …);
```

- In the Input Types, you can not have any text but specifying the types of parameters:
    - `%d`: integer (d for "decimal", not "double"!)
    - `%f`: float
    - `%lf`: double
    - `%c`: single character
    - `%s`: string
- In the Input Types, you can specify the length of parameters:
    - `%nT` where `n` is the *number of characters* and `T` is the *parameter type.*
- `scanf()` expects input in the same format as you provided.
- `scanf()` uses space as the delimiter, so "this is test" are three strings.
- `scanf()` returns the number of words read
- `scanf()` parameters [char, int, float, double] need to be POINTERS!!

```
char x[10]; // Arrays are fine since they are pointers to the 1st element.
scanf("%10s",x);      //input: 12345678901234567890
printf("Your input is %s\n",x); //output: 1234567890
int y;
scanf("%d", y); // Segmentation fault (core dumped)
//format '%d' expects argument of type 'int *', but argument 2 has type 'int'
scanf("%d", &y); // input: 5
printf("Your input is %d\n", y); // output: 5
```

Pointers-1

- **Variable Address for atomic variables** [int, float, char, ...]: `&variable-name`
- **Variable Address for arrays**: `array-name`
- **Declaration:** `data-type *pointer-name;`
  - *int* `*pointer2int;`
  - *float* `*pointer2float;`
- **Initialization:** `pointer-name = &variable-name; // or pointer-name = array-name;`

```
o  int k=5;
o  pointer2int= &k; // or int *pointer2int = &k;
o  int myarray[] ={1,2,3,4,5};
o  pointer2int = myarray; // or int *pointer2int = myarray;
```

- **Accessing the Variable Address:** `pointer-name`

```
printf("Address of variable k is %x\n", pointer2int ); //prints a0501c8
Equivalent to printf("Address of variable k is %x\n", &k); //prints a0501c8
```

- **Accessing the Variable Value [Dereferencing]:** `*pointer-name`

```
printf("Value stored in variable k is %d\n", *pointer2int );    //prints 5
Equivalent to printf("Value stored in variable k is %d\n ", k); //prints 5
```

- **Setting the Variable Value [Dereferencing]:** `*pointer-name = new-value;`

```
*pointer2int=2;
```

- **Pointer Arithmatic:**
  - **Incrementing** a Pointer using + or ++ [Move to the next address]

```
int  var[] = {10, 100, 200, 300, 400, 500};
int  *ptr = var; //point at first element
ptr++;  // point at second element
*(ptr + 2) = 1; // changes the fourth element's value var[3] to 1
ptr[4] = 0 ; // changes the fifth element's value var[4] to 0
```

  - **Decrementing** a Pointer using - or --

```
int  var[] = {10, 100, 200, 300, 400, 500};
int  *ptr = &var[5]; //point at the last element
ptr--;  // point at the fifth element var[4]
*(ptr - 2) = 1; // changes the third element's value var[2] to 1
```

  - Pointer **Comparison** using ==, < , >, <= , >=

```
int  var[] = {10, 100, 200, 300, 400, 500};
int  *ptr = var; //point at first element
i = 0;
while ( ptr <= &var[2] ) {
      printf("var[%d] %x - %d\n", i, ptr, *ptr );
      ptr++;
      i++;
}
```

Pointers-2

- **Generic Pointers** of data-type `void*`
    - void* pointers can be assigned and casted to any other pointer type.
    - void* pointers can NOT be dereferenced to get the variable value.

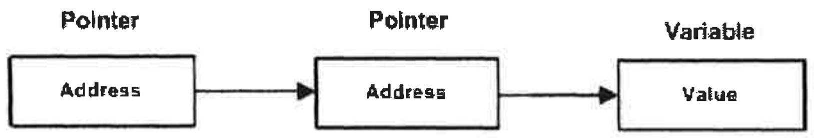
```
void *genericPtr;
int x = 7;
genericPtr = &x;
printf("%d\n", *genericPtr); // Illegal Dereferencing!
int *intptr = genericPtr; // Assigning the void* pointer to an int*
printf("%d\n", *intptr); // Legal Dereferencing
float f = 3.14159;
genericPtr = &f; // Assigning the void* pointer to an int*
// Legal casting the void* pointer to float*
printf("%f\n", *((float *)genericPtr));
```

- **Pointer Array:** `data-type *pointer-array-name[size];`

```
char *names[4] = {
      "Zara Ali",
      "Hina Ali",
      "Nuha Ali",
      "Sara Ali",
   };
   int i = 0;
   for ( i = 0; i < 4; i++) {
            printf("%c \t %s\n", *names[i], names[i] );
}
```

- **Pointer to Pointer:** *`data-type`* `**ptr2ptr-name = &ptr;`

```
int  var = 30;
int  *ptr = &var;
int  **pptr = &ptr;
printf("pptr address = %x\n", &pptr );
printf("ptr address = %x\n", pptr );
printf("var address = %x\n", *pptr );
printf("var Value = %d\n", **pptr);
```



- **Function Pointers** is good to know but NOT included in quizzes and exams

Functions

- **Function Declaration:** `return_type function_name( parameter list );`

```
int func(int x, int y);
```

- **Function Definition:**

```
return_type function_name( parameter list ) {
   //body of the function
}
```

```
int func(int x, int y) {
      return x+y;
    }
```

- **Call by Value:**

```
void myfun(int a) {
a=5;
}
void main () {
   int b = 200;
   myfun(b);
}
```

- **Call by Reference for variables:**

```
void myfun(int *a) {
*a=5;
}
void main () {
   int b = 200;
   myfun(&b);
}
```

- **Call by Reference for arrays:**

```
void myfun(int *a) {
*a=5;
}
void main () {
   int b[] = {1,2,3};
   myfun(b);
}
```

Dynamic Memory Allocation
#include<stdlib.h>

- Use function `malloc` (Memory Allocation) to allocate space in the Heap.
  `data-type *pointer = malloc(<number of bytes>);`
- You MUST use function `free` to release the allocated Heap space when you no longer need it.

```
int *b ;
b = malloc(sizeof(int)*1); // allocate 4 bytes for an int
*b = 5;
free(b);
```

Debugging

- **Step 1:** Collect debugging information when compiling your program using the –g option for gcc:
  - `gcc -g -Wall -o program-name filename.c`
- **Step 2:** After you compile your code, run gdb on your executable: `gdb program-name`
- **Step 3:** Add breakpoints to stop program at a certain line of function: `break <function-name>`
- **Step 4:** Run your program in debug mode by typing `run`.
- **Step 5:** Print variables as you please in the gdb debugger: `print <variable-name>`
- **Step 6:** Continue the execution after a breakpoint.
  - Type "c": Debugger will ***Continue*** executing until the next break point.
  - Type "n": Debugger will execute the ***Next*** line as single instruction.
  - Type "s": Debugger will ***Step*** inside the function and executes it line by line.
- **Step 7:** Type "quit" to exit the gdb debugger

Files-1 #include<stdio.h>

- **Create** a new file or **Open** an existing one using `fopen`:

```
FILE *fopen( const char * filename, const char * mode );
```

  - *Returns*: a pointer of type FILE to handle the file or NULL if error
  - *filename* is the name of the file to open/create
  - *mode* is a string of one or more option
    - "r " = read-only [no-create]
    - "w" = write and override [delete-create]
    - "a" = write and append [create]
    - "r+" = read-write override-on-write [no-create]
    - "w+" = read-write [delete-create]
    - "a+" = read-write append [create]

```
FILE *fp = fopen("test.txt", "w");
```

- **Close** a file using `fclose`:

```
int fclose( FILE *fp );
```

  - *Returns*: 0 on success and EOF [End-Of-File Macro] otherwise.
  - *fp* is the file handler pointer

```
fclose(fp);
```

- **Write** to a File using `fprintf`

```
int fprintf(FILE *fp, "Message", parameter 1, parameter 2, …)
```

  - *Returns* a non-negative value on success, otherwise **EOF** .
  - *fp* is the file handler pointer
  - *Second Argument* is a string specifying the format of the parameters
  - *Rest of Arguments* are parameters it write to the file

```
fprintf(fp,"%s %i\n",mystring, i);
```

- **Read** a file *word-by-word* [space is the delimiter] using `fscanf`

```
int fscanf(FILE *fp, "Message", parameter 1, parameter 2, …)
```

  - *Returns* a non-negative value on success, otherwise **EOF** .
  - *fp* is the file handler pointer
  - *Second Argument* is a string specifying the format of the parameters
  - *Rest of Arguments* are parameters it reads from the file
  - fscanf is very useful when you are reading tables of data from a file

```
char buff[255];
fscanf(fp, "%s", buff);
```

- **Read** a file *line-by-line* [\n is the delimiter] using `fgets`

```
char *fgets( char *buf, int n, FILE *fp );
```

  - fgets reads up to n-1 characters from the file referenced by fp.
  - It copies the read string into the buffer buf, appending a null character '\0' to the string.
  - If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character.

```
char buff[255];
fgets(buff, 255, fp);
```

- **Detecting End-Of-File [EOF]**

```
int feof(FILE *fp)
```

  - *Returns* a non-zero value when EOF, else zero is returned.

```
while(1)   {
    fgets(buff, 255, fp);
    if( feof(fp) ) {
       break ;}
    printf("%s", buff);
}
```

Files-2
#include<stdio.h>

- **Tell where the read-write indicator is**

  `int ftell(FILE *fp)`
  - *Returns* the current file read-write indicator position and -1 if error.
  - *fp* is the file handler pointer

```
fseek(fp, 0, SEEK_END);

int len = ftell(fp);
```

- **Move the read-write indicator:**

  `int fseek(FILE *fp, int length, int start)`
  - Moves the file read-write indicator to the specified position. *Returns* zero if successful
  - *fp* is the file handler pointer
  - *length* is the number of bytes [characters] to move from the start value.
  - *start* is the position from where length is added. Can have one of these values:

– `SEEK_SET`	Beginning of file
– `SEEK_CUR`	Current position of the file pointer
– `SEEK_END`	End of file

- **Predefined File Pointers:** You can use the predefined File Pointers `stdin, stdout, stderr` to read from the keyboard and write to the screen.
  - `stdin` takes input from keyboard unless program is being run with standard input redirected
  - `stdout & stderr`: writes to screen unless program is being run with standard output/error redirected

```
char buff[255];
fgets(buff, 255, stdin);
fprintf(stdout, "%s\n", buff);
fprintf(stderr, "Message to stderr");
```

Error Handling

- You can get a more meaningful run time error messages using: `errno, perror() and strerror()`

#include<errno.h>

  – Most C function set an error code in the C global variable `errno` when a run time error happens.

#include<string.h>

  – `void perror("Message")` prints to screen the string you pass to it, followed by a colon, a space, and then the error message associated with the current `errno` value.
  – `char* strerror(int errno)` uses the `errno` value to find and return a char pointer [string] of the error message associated with that `errno` value.

#include<stdlib.h>

- If there is a run-time error that you cannot handle, make sure your program exists with an error status using the exit(int code) function.
  – `exit(EXIT_SUCCESS); // or exit(0);`
  – `exit(EXIT_FAILURE); // or exit(-1);`

Structures

- **Definition:**

```
struct structure-name {
            data-type variable1;
            data-type variable2;
};
```

```
struct Point{
      int x;
      int y;
};
```

- **Initialization [Create an "Object"]:**

```
struct structure-name object-name = {init-value1, init-value2};
```

```
struct Point p = {1,2};
```

- **Assign/Change values:**

```
object-name.variable1 = value;
```

```
p.x = 10;
```

- **Alternative 1:**
- **Definition:**

```
typedef struct{
            int x;
            int y;
} Point;
```

- **Initialization [Create an Object]:**

```
Point p = {1, 2};
```

- **Alternative 2:** Create structure then give it an alias:
- **Definition:**

```
Struct Point{
       int x;
       int y;
};
typedef struct Point Point;
```

- **Initialization [Create an Object]:**

```
Point p = {1, 2};
```

- **Alternative 3:** Create structure and objects in a single step:

```
Struct Point{
       int x;
       int y;
} p1,p2, p3;
```

- **Alternative 4:** Create anonymous structure that only has 3 objects:

```
Struct {
       int x;
       int y;
} p1,p2, p3;
```

Structures are treated like any other data type: can have pointers pointing at them, can create arrays of them, can have a structure defined in a structure and can pass them to and from Functions either by-value or by-reference.

Unions

- **Definition:**

```
union union-name {
            data-type variable1;
            data-type variable2;
};
```

```
union Data{
        int x;
        float y;
        char[20] z;
};
```

- **Create an "Object":**

```
        union union-name object-name;
```

```
union Data d;
```

- **Assign/Change values:**

```
        object-name.variable1 = value;
```

```
d.x = 10;
Or d.y = 20.5;
Or d.z="abcd";
```

Processes

#include<stdlib.h>

- `system()` takes a single string parameter and executes it as a different process similar as if you had typed it on the command line. `system ("ls -lah"); // list all files in current directory`
- `sleep(int seconds)` causes a program to suspend for the number of specified seconds.
- `pid_t fork();` creates a copy of the current process
  - In the child: **pid_t** value is zero.
  - In the parent: **pid_t** value is the pid of the new child.
  - negative **pid_t** : fork failed – no new process
  - `pid_t getpid();` *returns process' own pid*
  - `pid_t getppid();` *returns pid of process' parent*

#include <sys/wait.h>

- The `wait ()` system call causes a parent process to pause until its child exits.
- `pid_t child_pid = wait( int* stat_val);`
  - The call returns the PID of the child process.
  - It sets the stat_val pointer stores the child exit status.
- You need to use the WEXITSTATUS() function to print out the child exit status.
- `int child_status = WEXITSTATUS( int stat_val);`

#include <unistd.h>

- The `execvp(char cmd[], char* args[])` functions replace the current process by running some other program.
  - Doesn't return to your program if success!!!!!!
  - Returns -1 if FAILED to execute the command.
- The first execvp parameter is the name of the command to execute.
- The second parameter is an array of command arguments given that
- The first element of the array MUST be the command itself AGAIN!!.
- The last element of the array MUST be NULL

Multi-Threading
#include <pthread.h>

- `#include <pthread.h>`
- Define the thread function
  - `void* foo(void *args) { }`
  - The function must return void * and take one void * argument.
- Create a thread
  - `pthread thread;`
  - `int pthread_create(pthread_t *thread, NULL,`
    `void *(*thread_function) (void *), void *arg);`
    - Returns zero on success
- Exit current thread
  - `void pthread_exit(void *value_ptr);`
    - value_ptr is a value to pass to the caller. Use NULL.
- In main, wait thread to complete [OPTIONAL]
  - `int pthread_join(pthread_t thread, void **value_ptr);`
  - value_ptr is a value passed from the thread. Use NULL.
- Compile using the "-pthread" option.

```
gcc -Wall -pthread -o program program.c
```