L11
COMMAND LINE AND MAKING PROGRAMS GENERAL

Using Command Prompt to run a C/C++ program

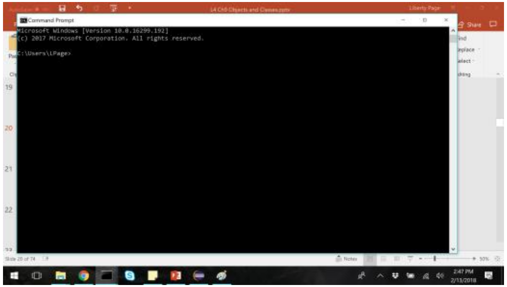
We'll discuss Command prompt in PC, Terminal in Mac or Bash in Linux/Unix

How to get to the Command Prompt/Terminal

- PC users:
  - **In start, type comm** and command prompt should be at the top of your search results. **Click on the icon** to start the command prompt window.

Mac Users:

Search for **Terminal** and run it.



```
Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\LPage>
```

Command Prompt & Terminal Commands

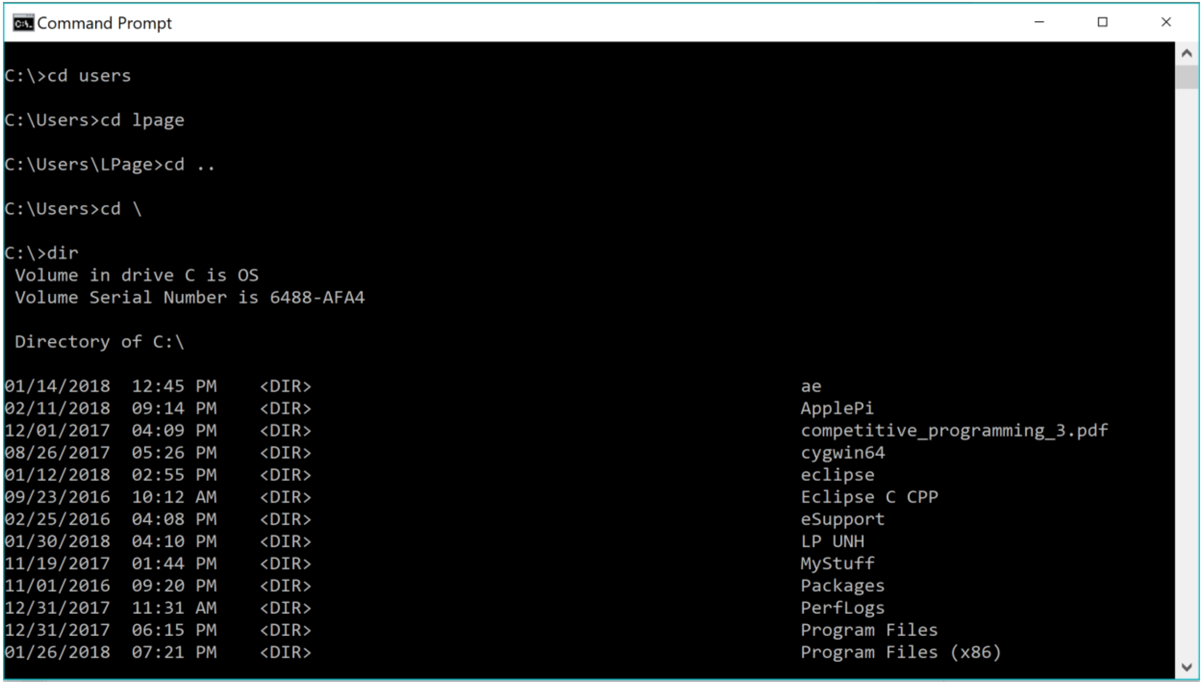
PC: Command Prompt	Mac/Linux/PC: Bash or Terminal
dir – list files in the directory	ls – list files in current folder (directory)
cd \ - change directory to root	cd – change directory to home folder cd / - change directory to root folder
cd .. go back by one directory	cd .. – go back one directory
cd <folder name> - go into this subdirectory	cd <folder name> - go into specified subdirectory
cls – clears the text from the screen	clear – clears text from the screen
chdir – like cd but more characters	cd c: - change directory to top of c drive

How to navigate the file system using the PC Command Prompt

- PC users:
  - Using the Command Prompt command `cd`, **navigate to the folder where you have your binary files for a program (.exe), usually your debug folder.** This requires you to know where your work or workspace for Eclipse/Xcodes is located.
  - By default Eclipse puts them in

```
C:>cd c:\users\your username\work
```

- You can change folders (directory) in one cd command or change one subdirectory at a time.
- The `cd` command isn’t case sensitive.



Command Prompt

```
C:\>cd users

C:\Users>cd lpage

C:\Users\LPage>cd ..

C:\Users>cd \

C:\>dir
 Volume in drive C is OS
 Volume Serial Number is 6488-AFA4

 Directory of C:\

01/14/2018  12:45 PM    <DIR>                                          ae
02/11/2018  09:14 PM    <DIR>                                          ApplePi
12/01/2017  04:09 PM    <DIR>                                          competitive_programming_3.pdf
08/26/2017  05:26 PM    <DIR>                                          cygwin64
01/12/2018  02:55 PM    <DIR>                                          eclipse
09/23/2016  10:12 AM    <DIR>                                          Eclipse C CPP
02/25/2016  04:08 PM    <DIR>                                          eSupport
01/30/2018  04:10 PM    <DIR>                                          LP UNH
11/19/2017  01:44 PM    <DIR>                                          MyStuff
11/01/2016  09:20 PM    <DIR>                                          Packages
12/31/2017  11:31 AM    <DIR>                                          PerfLogs
12/31/2017  06:15 PM    <DIR>                                          Program Files
01/26/2018  07:21 PM    <DIR>                                          Program Files (x86)
```

How to run a C++ program from the prompt

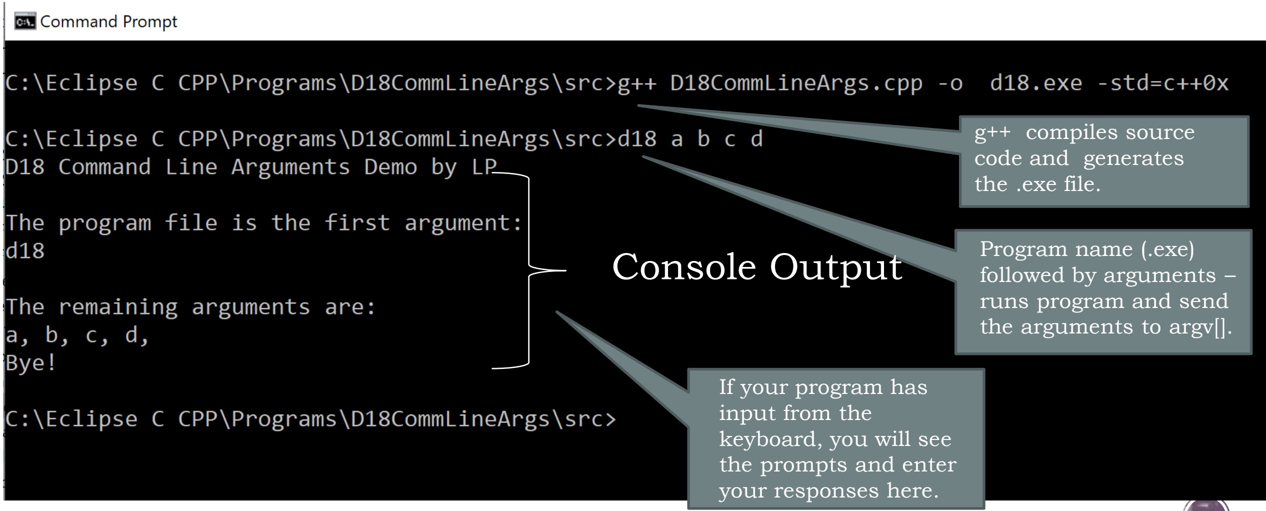
- Now check to make sure the .exe file is present using the `dir` PC command or `ls` Unix command for Mac.
- If so, you can run your program (program name is case sensitive)…

```
c:\eclipse c cpp\programs\d18\debug>progname
```

  - Your console output will print here, if your program has input, you can enter it in the command prompt window.
- If not, use g++ program to compile your source code using the source code in that folder (src). In this case, the .exe file gets put in the src folder, unlike Eclipse, which saves it in Debug.
  - If you need to compile the program, use cpp with the entire file name

```
C:\ eclipse c cpp\programs\d18\src>g++ progname.c -o newprogname.exe
-std=c++0x
For C use gcc instead of g++ and -std=c11
```

Command line arguments – in Command Prompt from the folder with the executable



Command Prompt

```
C:\Eclipse C CPP\Programs\D18CommLineArgs\src>g++ D18CommLineArgs.cpp -o  d18.exe -std=c++0x

C:\Eclipse C CPP\Programs\D18CommLineArgs\src>d18 a b c d
D18 Command Line Arguments Demo by LP

The program file is the first argument:
d18

The remaining arguments are:
a, b, c, d,
Bye!

C:\Eclipse C CPP\Programs\D18CommLineArgs\src>
```

Console Output
g++ compiles source code and generates the .exe file.
Program name (.exe) followed by arguments – runs program and send the arguments to argv[].
If your program has input from the keyboard, you will see the prompts and enter your responses here.

Command Line Arguments

How to add them and use them

Command Lines

- ►A **command line** is a line of text that you type into a command shell to cause a program to beexecuted.
- ►This method of executing a program is important when you are working through an internet connection to a remote computer.
- ►Even on your local machine, sometimes it is just *easier* to use the command line than to create aproject.
- ►A command line starts with the name of the program to execute, followed by zero or more **arguments**.
- ►Arguments can include switches, numbers, and file names, in varying combinations.
- ►Some arguments are optional. These are normally given before the required arguments. File names are normally last.

Command lines in your program: argv and argc

► When you use the full standard prototype for main:

```
int main (int argc, char* argv[]);//typical
int main (int argc, char** argv );//equivalent
```

►The operating system will accept information on the command line and deliver it to you through `argc` and `argv`.

►`argc` tells you how many strings are in the array.

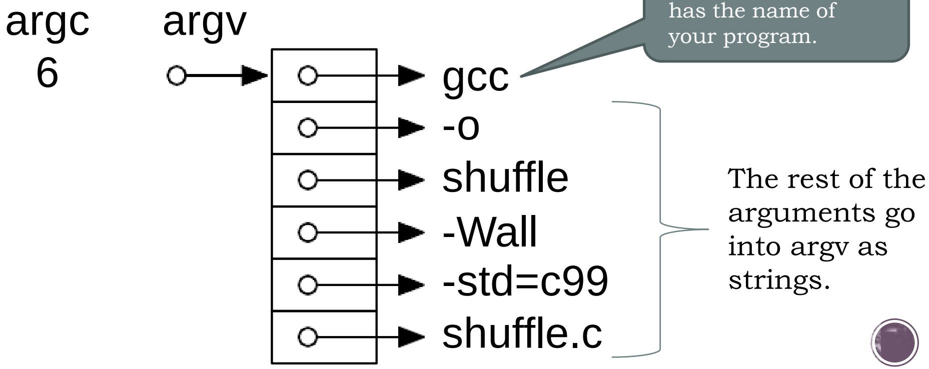
►`argv` is an array of whitespace-delimited fields that you typed on your command line, where each field has become a separate null-terminated string.

►Your job is to analyze these strings and set up your program's internal environment appropriately.

Example: argc and argv

- This is the Unix/command line command to **compile the shuffle program**.

```
gcc -o shuffle -Wall -std=c99 shuffle.c tools.c
```



argc
6
argv
gcc
-o
shuffle
-Wall
-std=c99
shuffle.c
The first slot of argv has the name of your program.
The rest of the arguments go into argv as strings.

Example: sorting.c
A Command-Line Program to Sort doubles

- Usage: `sorting [-r] maxItems inName outName`

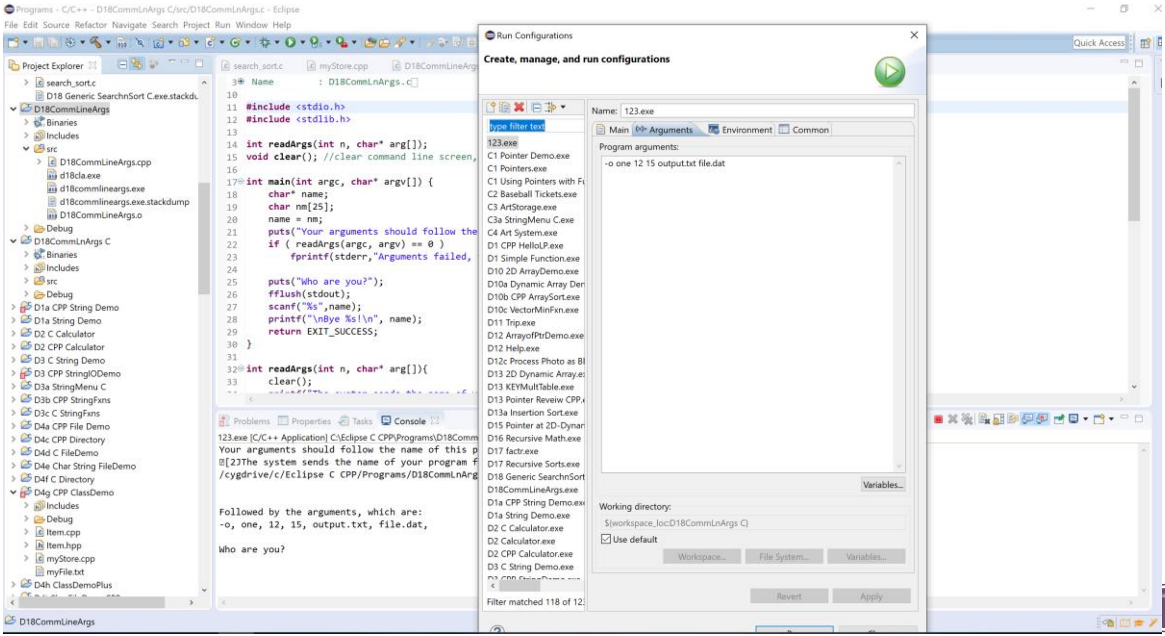
- The arguments to the sorting program are:
    - ► Optional: "-r" if you want the numbers in reverse (descending) order
  - ► `maxItems` is an integer - the # of doubles to read from the file and sorted.
  - ► `inName` is the name of the file that contains the unsorted data.
  - ► `outName` is the name of an output file for sorted data.

- When a command line is given that does not satisfy the requirements of a program, it is customary to display a usage comment and quit.

sorting.c: Interpreting the command line.

► If the number of arguments is wrong, display a usage comment.
► Set `argp` to point at the first argument.
► If `argc` is 5, the optional parameter is present. If so, check to be sure it is actually "-r". If so, set the boolean variable to indicate that you need to sort in reverse (descending) order.
Increment the argument pointer past the optional argument.
► Next is the number of doubles to read from the file and sort.
► Use `strtol()` to convert this string to a long integer.
► Open the input file (and test).
► Display an error comment and quit if the output file already exists.
Otherwise, openit.

Command line arguments – in Eclipse



Programs - C/C++ - D18CommLnArgs C/src/D18CommLnArgs.c - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- search_sort.c
- D18 Generic SearchnSort C.exe.stackdu
- D18CommLineArgs
  - Binaries
  - Includes
  - src
    - D18CommLineArgs.cpp
    - d18cla.exe
    - d18commlineargs.exe
    - d18commlineargs.exe.stackdump
    - D18CommLineArgs.o
  - Debug
- D18CommLnArgs C
  - Binaries
  - Includes
  - src
  - Debug
- D1a CPP String Demo
- D1a String Demo
- D2 C Calculator
- D2 CPP Calculator
- D3 C String Demo
- D3 CPP StringIODemo
- D3a StringMenu C
- D3b CPP StringFxns
- D3c C StringFxns
- D4a CPP File Demo
- D4c CPP Directory
- D4d C FileDemo
- D4e Char String FileDemo
- D4f C Directory
- D4g CPP ClassDemo
  - Includes
  - Debug
  - Item.cpp
  - Item.hpp
  - myStore.cpp
  - myFile.txt
- D4h ClassDemoPlus

search_sort.c | myStore.cpp | D18CommLineArgs

```
 Name        : D18CommLnArgs.c

#include <stdio.h>
#include <stdlib.h>

int readArgs(int n, char* arg[]);
void clear(); //clear command line screen,

int main(int argc, char* argv[]) {
    char* name;
    char nm[25];
    name = nm;
    puts("Your arguments should follow the
    if ( readArgs(argc, argv) == 0 )
        fprintf(stderr,"Arguments failed,

    puts("Who are you?");
    fflush(stdout);
    scanf("%s",name);
    printf("\nBye %s!\n", name);
    return EXIT_SUCCESS;
}

int readArgs(int n, char* arg[]){
    clear();
```

Problems | Properties | Tasks | Console

```
123.exe [C/C++ Application] C:\Eclipse C CPP\Programs\D18Comm
Your arguments should follow the name of this p
[2JThe system sends the name of your program f
/cygdrive/c/Eclipse C CPP/Programs/D18CommLnArg


Followed by the arguments, which are:
-o, one, 12, 15, output.txt, file.dat,

Who are you?
```

Run Configurations

Create, manage, and run configurations

type filter text

123.exe
C1 Pointer Demo.exe
C1 Pointers.exe
C1 Using Pointers with Fu
C2 Baseball Tickets.exe
C3 ArtStorage.exe
C3a StringMenu C.exe
C4 Art System.exe
D1 CPP HelloLP.exe
D1 Simple Function.exe
D10 2D ArrayDemo.exe
D10a Dynamic Array Dem
D10b CPP ArraySort.exe
D10c VectorMinFxn.exe
D11 Trip.exe
D12 ArrayofPtrDemo.exe
D12 Help.exe
D12c Process Photo as Bl
D13 2D Dynamic Array.e
D13 KEYMultTable.exe
D13 Pointer Reveiw CPP.
D13a Insertion Sort.exe
D15 Pointer at 2D-Dynar
D16 Recursive Math.exe
D17 factr.exe
D17 Recursive Sorts.exe
D18 Generic SearchnSort
D18CommLineArgs.exe
D1a CPP String Demo.ex
D1a String Demo.exe
D2 C Calculator.exe
D2 Calculator.exe
D2 CPP Calculator.exe
D3 C String Demo.exe

Filter matched 118 of 12

Name: 123.exe

Main | Arguments | Environment | Common

Program arguments:

```
-o one 12 15 output.txt file.dat
```

Variables...

Working directory:

${workspace_loc:D18CommLnArgs C}

☑ Use default

Workspace... | File System... | Variables...

Revert | Apply

D18CommLineArgs

How agrv[ ] works

- The **OS passes an array of strings to `main()`** when the program is started up. Call this array `argv[]`.
- These strings can say anything and mean anything. That is up to the application designer.
- Each application must extract the information it expects from the `argv[]` array.
  - The **first argument (slot 0 of the array) is the name of the program**.
  - The rest of the slots contain the arguments, they are strings.
  - The number of items is in `argc`, an integer.

Your Program Should Explain the Interface

- Every user interface must explain itself fully.
  - The user must know exactly how to communicate with the program, or it is useless.
  - The instructions must come to the console screen in a timely manner. Expecting the user to find the written documentation is unrealistic.
  - Command-line programs typically display usage comments.
  - Like any error comment, the usage comment should be as explicit as possible.
  - A typical usage comment for a program with three args:

```
"Usage: nData infile outfile"
```

Generic Functions

How Generic Functions work – the magic of void pointers

Void Pointers allow generic parameters

- **Void pointers** are pointers that point to a value that **has no type** .
- They have an **undetermined length** and **undetermined dereference properties**.
- This **allows** void pointers to **point to any data type**.
- **Limitation:** the **data pointed by them cannot be directly dereferenced** (since it has **no type)**.
- You will *always* have to **cast** the address in the void pointer **to some other pointer type** that points to a **concrete data type** before dereferencing it.
- **<u>Uses:</u>** to pass **generic parameters to a function**

```
int foo(void* myVoidVar, int typeSize);
```

C’s Generic Functions Work on any Type

- `qsort() and bsearch()`
- Sorting and searching operations are **independent of the type of data** being processed.
- You can sort an array of **any base type**.
- However, *an ordinary program must be edited and recompiled* to *change the base type* of the array.

qsort() and bsearch()

- OO languages overcome the requirement to edit and recompile by using classes.
- We can achieve part of this goal by using void pointers, which have the type: `void* myVoidPtr;`
- These generic functions depend on a specific comparison function to define the meaning of the idea “sorted”.
- We can sort in ascending or descending order, and we can sort on a whole data object or one field of a structure.

qsort() and bsearch()

- These functions are defined in `<stdlib.h>`.
- They let you sort or search an array of any base type.
- `qsort()` performs a quicksort.
- `bsearch()` performs a binary search.
- The arguments to `qsort()` are
  - A **pointer to the array to sort**. Cast the pointer to type `void*`.
  - The **number of items** to sort, an int.
  - The **size of the base type** of the array.
  - A **comparison function** that defines the sort order.
- `bsearch()` has another parameter at the beginning of the list, the **key value** to search for.

Comparison Functions

- A `qsort` comparison function has two parameters of type `const void*`.
- Each one is a void pointer to an array element.
- The two array elements will be compared.
- The result returned will be **0 if they are equal, negative if the first should sort before the second**, and **positive if the second should sort before the first**.

Example of Comparison Function

- This function **sorts** an array of doubles into ascending order:

```
int ascend ( const void* ap, const
void* bp )
{
//cast before dereferencing void
pointers
   double a = *(double*)ap;
   double b = *(double*)bp;
   return a-b;
}
```

Comparing Two Structures

- If you are sorting an array of structs, the comparison function must select one member of the struct to compare. Here is a function that will compare two CardT values based on the ranks of the two cards.

```
typedef
struct {SuitT suit; char rank; int pts;}CardT;

int ascend ( const void* ap, const void* bp )
{
  CardT a = *(CardT*)ap;
  CardT b = *(CardT*)bp;
   return a.rank - b.rank;
}
```

Example: Calling qsort() and bsearch()

```
// Sort the array in ascending order.
qsort( (void*)data, n, sizeof(double), ascend );

// Search for 12.4 in the sorted array.
bsearch( (void*)12.4, (void*)data, n, sizeof(double),
compare);

// For your reference:
int ascend ( const void* ap, const void* bp ) {
    double a = *(double*)ap;
    double b = *(double*)bp;
    return a-b;
}
```

Another Use of void*

- **Problem**: C++ helps you by dereferencing string and char[] pointers for you. That is great most of the time but how do you print the address of the pointers?
- **Solution: Cast the variable as a void pointer** first, so is has no type, thus you will get the **address in the pointer variable**.
- **Syntax:** Put (const void*) in front of the array name or string variable.

```
char letters[3][4];
cout << "Address: " << hex << (const void*)letters <<
dec << endl;
```

One Last Operator

We’ve discussed every operator except one

?: Conditional Ternary Operator

- Syntax:

  condition ? *value_if_true* : *value_if_false*

```
 return (a<b ? a : b);
```

- Can be rewritten as:

```
 if(a < b) return a;
 else return b;
```

?: Example

```
#include <iostream>
using namespace std;
int main ()
{ int a,b,c;
cout << "Enter two integers: "<< flush;
cin >> a >> b;
c = (a>b) ? a : b;
cout << c << endl;
}
```

Operation Precedence

- When an expression has two operators with the same precedence level, *grouping* determines which one is evaluated first: either left-to-right or right-to-left.
- Enclosing all sub-statements in parentheses (even those unnecessary because of their precedence) improves code readability.

From greatest to smallest priority, C++ operators are evaluated in the following order:

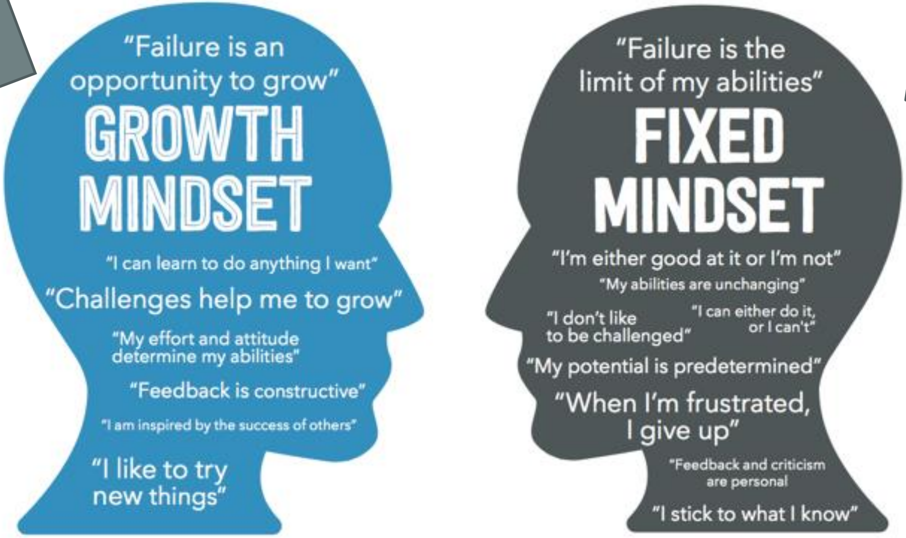
Level	Precedence group	Operator	Description	Grouping
1	Scope	::	scope qualifier	Left-to-right
2	Postfix (unary)	++ --	postfix increment / decrement	Left-to-right
		()	functional forms	
		[]	subscript	
		. ->	member access	
3	Prefix (unary)	++ --	prefix increment / decrement	Right-to-left
		~ !	bitwise NOT / logical NOT	
		+ -	unary prefix	
		& *	reference / dereference	
		new delete	allocation / deallocation	
		sizeof	parameter pack	
		(*type*)	C-style type-casting	
4	Pointer-to-member	.* ->*	access pointer	Left-to-right
5	Arithmetic: scaling	* / %	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	+ -	addition, subtraction	Left-to-right

Level	Precedence group	Operator	Description	Grouping
7	Bitwise shift	<< >>	shift left, shift right	Left-to-right
8	Relational	< > <= >=	comparison operators	Left-to-right
9	Equality	== !=	equality / inequality	Left-to-right
10	And	&	bitwise AND	Left-to-right
11	Exclusive or	^	bitwise XOR	Left-to-right
12	Inclusive or	|	bitwise OR	Left-to-right
13	Conjunction	&&	logical AND	Left-to-right
14	Disjunction	| |	logical OR	Left-to-right
15	Assignment-level expressions	= *= /= %= += -= >>= <<= &= ^= |=	assignment / compound assignment	Right-to-left
		?:	conditional operator	
16	Sequencing	,	comma separator	Left-to-right

Thank you for a great semester!

This is the right way to think!

No!!



"Failure is an opportunity to grow"

GROWTH MINDSET

"I can learn to do anything I want"

"Challenges help me to grow"

"My effort and attitude determine my abilities"

"Feedback is constructive"

"I am inspired by the success of others"

"I like to try new things"

"Failure is the limit of my abilities"

FIXED MINDSET

"I'm either good at it or I'm not"

"My abilities are unchanging"

"I don't like to be challenged"

"I can either do it, or I can't"

"My potential is predetermined"

"When I'm frustrated, I give up"

"Feedback and criticism are personal"

"I stick to what I know"