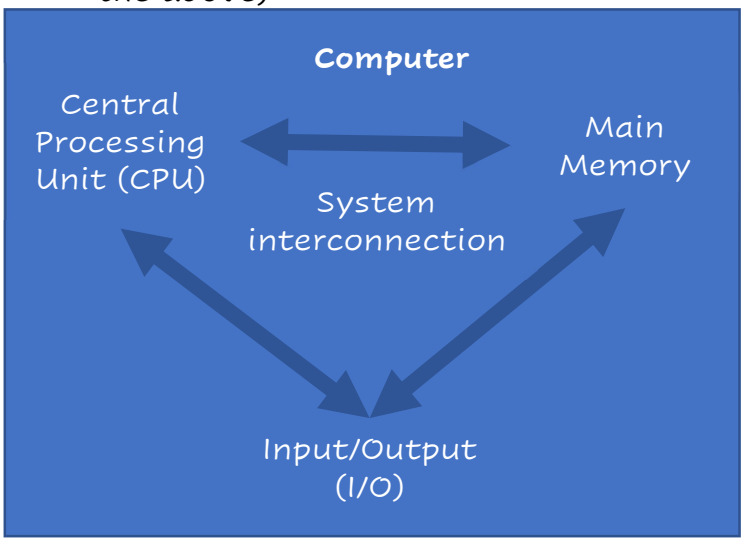
CS 159 Notes

Chapter 1- Computers

4 parts of computer:

1. A central processing unit
2. Main memory
3. Input/output hardware
4. Systems interconnection (connects all the above)

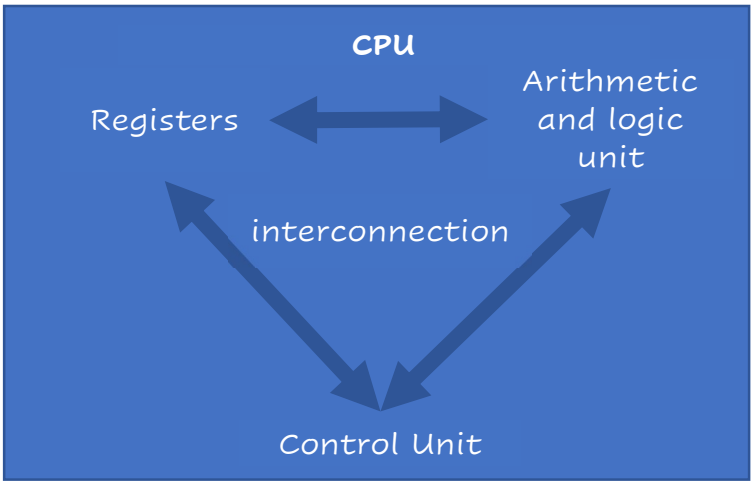


as programmers we only care about the central processing unit and the main memory storage

Processor

where all the processing of instruction and associated data will take place

1. Registers- small amount of memory local to the processor used for temporary storage related to the current instruction being executed
2. Arithmetic and Logic Unit- Hardware responsible for the calculations required of an instruction
3. Control Unit- coordinates the operation of the processor
4. Interconnection- connecting all the above

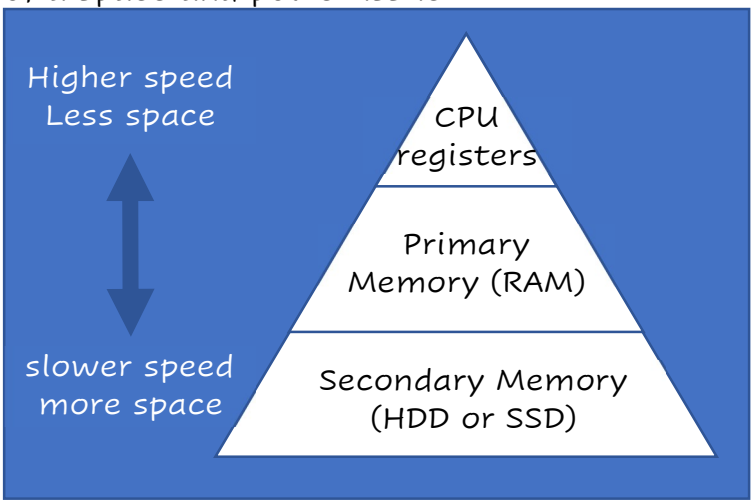


Main Memory

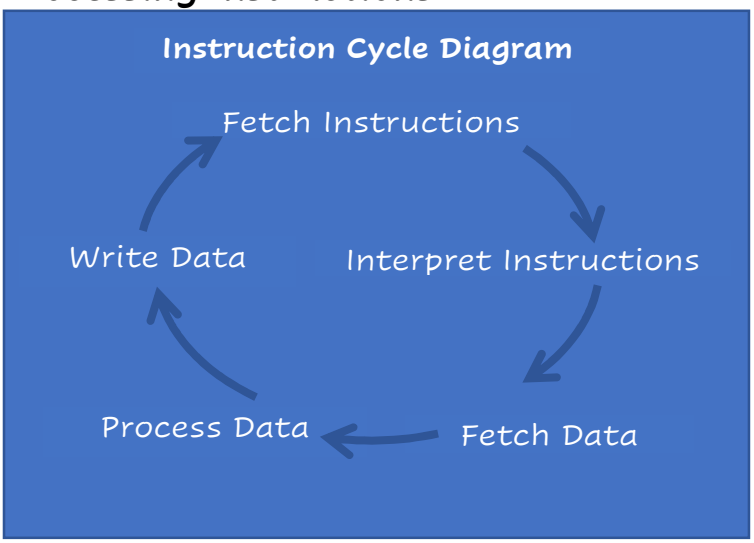
the memory of a computer will store the data and instruction of the programs that we will write
Primary memory- RAM (random access memory)
Secondary memory- SSD (solid state drive) or HDD (hard disk drive)
Second Memory is slower than primary Memory

- Further physical distance from CPU
- Increased time required to locate specific content

Can't grow size of primary memory because of a space and power issue



Processing Instructions



Languages

Natural languages- languages that are spoken and have evolved over centuries for of attributes and unfit for use by computer
Programming language- composing of keywords that come from natural language but are used without the ambiguity

High level Languages- uses natural language elements

Programming Languages- translate high level languages

Low Level Languages- instructions written specifically for the hardware

Machine language- language of the computer that is written in only 0's and 1's because the internal circuits of a computer are made of switches
compilation- translator. Turns programing language into low level language

Programming Process

1. Write/edit source code in programming language (lb00.c)
2. Save changes (:w)
3. Compile source code (gcc lb00.c)
4. Run executable (a.out)

Text editor- software used to write computer program (for this course will be vi)
Compiler- category of software that is used to convert the source programming language into the machine language
Syntax- set of rules of the programming language
Compiler attempts to validate your source code with the syntax of the C programming language
Error- violation in the rules of the syntax (no executable file will be created)
Compiling Process

- Preprocessor- red the source done and prepares it for translation.
- Preprocessor directives- special instructions that you give the preprocessor to prepare source done for translation

# include <stdio.h>	#define x 1

- Each time you modify the source file you must recompile the code and create a new a.out to see the changes as reflected in your source code

Structure Charts

Large programs are complex structures consisting of many interrelated parts and must be carefully designed
The structure chart shows how we break up our problem into sub problems and how data will flow between these modules
Cross hatch in lower right corner indicates that function is called fore than once

Flowchart

Visually describe visually the logic of an individual function (smaller tasks of larger problem)

Chapter 2- Introduction to C

Commenting

Internal documentation which are ignored by the compiler but used for reading and making sense of the code

- Single line comments //text
- Multiline comments /* text */

Comments can appear anywhere into the program
Don't comment every line
Must include an assignment header ever program
Must comment every variable

Indicators

Allow use to name data and other objects in program

variable in the program must be related to its purpose in the program
Rarely are single characters
Characters in all caps represent a symbolic/defined constant
First character must be alphabetic
Must consist of alphabetical characters numbers and underscores
can't duplicate a reserved word

Data Types

- Determines the memory necessary to store a value of a given type
- How value of a given type is stored in the memory of the computer
- Which operations can be performed on the value?

Integer- number without fractional part (short, int, long, long long)
Character-value that can be represented in the alphabet of the computer

- Each value has a corresponding small integer value
- Since it is stored as an integer you can perform arithmetic operations on character

Boolean- logical data (either true or false)

- Nonzero numbers are true and zero is false
- We will not use the bool data type this semester

Floating point- Number with fractional part (float, double, and long double)

Variables

Named memory location that has a data type
Must be declared and defined
Declaration- gives variable a name
Definition- reserves memory location
Variable initialization- setting variable equal to a value for the first time
Possible to initialize variable at the same time its created
Uninitialized variables will be assigned a location in memory that may have prior use

Constants

Literal constants- an unnamed value used to specify data (actual number)
Symbolic/defined constants- using the #define preprocessor all occurrences of symbol will be replaced by replacement value

```
#define symbol replacement
```

Use of constants help document the program by given meaning to operands in an expression
Defined constants

- No data type
- Literals they expand to do
- Don't use the assignment operator
- No extra memory

Difficult because it can be difficult to error check

Formatted IO (input and output)

Printf- used to displayed message and formatted data to the monitor

- printf(Format string, data list)

```
\n- new line
```

- Placeholder
  - Int %d
  - Long %ld
  - Long long %lld
  - Float %f
  - Double %lf
  - Long double %Lf
  - Char %c
- Placeholder- used to hold space for variable
  - Width modifier- used to reserve a given amount of space for a value to be displayed
  - Precision modifier- used with plot point data to determine the number of digits to display to the right of the decimal place indicates where the memory

```
%(flag)(width)(precision)(size)(code)
```

- Scanf- input function
  - Scanf(format string, address list)
  - Format string- list the types of data expected as input
  - Address list- indicates where in the memory of the computer the input should be stored

Errors

Syntax error- compile time error that arises due to a mistake you made that violates the syntax rules of the C programming language causing the compiler to stop working
Warning- feedback provided by the compiler that you might have a mistake
Run time error- error occurs while program is running

- Can cause crash (segmentation fault and floating exception)
- Output is not what is expected
- Logical error- run time error in which the programmer is responsible for implementing a flaw in the program

Chapter 3- Structure of C

Expressions

reduces to a single variable
Operator- language specific syntactical token that requires action to be taken +-*/^
Operand- receives operator's action
Precedence- determines which operator and operand belong and the order in which it will be evaluated
Assignment expressions- evaluates the operand on the right side of = and saves the value in the memory as the variable on the left
Compound assignment- apply a mathematical operation to a viable and save it back to that value (+=, -=, *=, /=, %=)

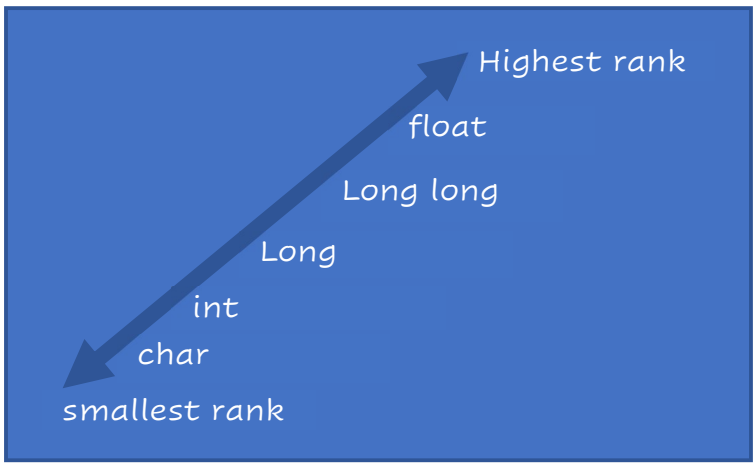
Prefix and postfix operators

Post fix expression- one operand followed by operator (gets data for use and then updates the value)
Prefix expression- one operator followed by operand (updates value then gets the data for use)
If a single variable is modified more than once in an expression the result is undefined

Mixed Expressions

Int/int gives the result of an int
In a mixed type expression, the lower ranked data type is implicitly converted to the datatype of the higher rank

3+2.1 converts 3 to 3.0



Implicit conversion- done automatically by computer without instructions from user
Explicit conversion- type casting (specifying what type of data the variable should be) when displayed on printf data is rounded when converted to in data is truncated assignment conversion- occurs when data type of the right side is different than the variable on the left side. The data on the right is converted to match the data type of the variable on he left
Selection via calculation

(n+1)%n=1 or 0 (if n=1)

Chapter 4- User Defined Functions

Functions are used:

1. To make program smaller to break problem down
2. Resus- in same program and other programs
3. Protect data

Functionally cohesive- Function could consist of a single task

- Eliminate redundant code
- Testable by itself separate from the rest of the program

Call function- the act of telling a function to run
Calling function- function that makes the call
Called function- function being told to run
Parameter passing- the act of sending data to a function
Returning- sending a value back to called function
Function declaration- in global declaration section and tells the compiler:

1. The type of value being returned from the function
2. The name of the function
3. The data being sent to the function, number of parameters, type of parameters, order of parameters

Function definition- contains the code needed to complete task
Void function returns no values

	Has parameters	No parameters
Value returned	Calculation	Input
No value returned	Outputs function	Welcome message

Values being passed to function are stored in new variables representing new unique memory locations to refer to same memory location we must pass my address
Pointer- variable that stores the memory address of another as its value we need such a variable to receive the address being sent from the calling function to the called function
Use pass by address when more than 1 value needs to be revised and is still functionally cohesive
You can call functions from functions but don't go to deep.
Structure chart should be wider than long
Downward communication- pass by value
Upward communication- passes my address

Chapter 5- Selection

Logical

Selecting code to execute or exclude based on a condition
Logical- piece of data that conveys the value of true or false
Logical data is garneted by a logical expression

- True- not zero
- False- zero

Logical operators

- !- not operator
  - Unary operator- 1 operand
  - Changes a true value to false and false value to true
- &&- and operator
  - Binary operator- 2 operands
  - If both operands are true evaluates to true otherwise evaluates too false
  - If the first operand is false it short circles and skips the second operand
  - Higher precedence than or operator
- ||- or operator
  - Binary operator- 2 operands
  - If both operands are false evacuates to false otherwise operates to true
  - If the first operand is true it short circles and skips the second operand

Relational Operators- binary operators where the result will be 1 or 0

- <- less than
- >- greater than
- <=- less than or equal to
- >=- greater than or equal too

Comparative operators

- == equal to
- !=not equal to

Compound statements- combining series to relational or logical operates resulting in a true or false logical value
The compiler uses a warning that you should use (and) when using && and|| in the same expression to make sure expression is evaluated the correct way

Complements

< → >=
> → <=
== → !=
&& → ||
! →!!
! → nothing
When complementing && operator with or operator () must be used round the operands to its precedence is higher

```
X && y || z → (!x || !y) && !z
```

Two-way selection

Logical expression evaluates to true or false

- If the expression is true goes down one path
- If the expression is false, it goes down the other path

```
if (logical expression)
{
  //true actions
}
else
{
  //false actions
}
```

- the else statement is optional
  - if body is blank delete it

nested if statement- you can put an if statement in the body of another if statement

{}- are not needed for 1 lined bodies but is a course programming standard

Conditional Expressions

ternary operands- to operators and three operands
first operator is a question mark that separates the first two expressions
the second operator is a colon that operates the last two expressions

```
condition ? true action : false action;
```

only used for very simple expressions

Multi-way Selection

used for more than two alternate sections of unique code

- if/else if/ else

```
if (logical expression)
{
  //true actions
}
else if (logical expression)
{
  //true actions
}
else
{
  //false actions
}
```

- evaluates if logical statement
  - if its false moves down and repeats until one is true
  - if it reaches the end then goes through the false actions
  - you can put as many else if statements as necessary

Switch

used to make selection among possible integer values

- favored when possible, integer values are well known and few
- the control expression that follows the keyword switch is evaluated and compared the series of cases
- each case represents an expected value of the control expression
  - associated with each case are 1 or more executable statements
  - the cases must be evaluated to an integer (so can be integer logical expression or a character)
- default is case option to cover all values not represented by the other individual cases
- without break statement as soon as one true case is found all of the following code is executed

Switch Construct Rules

1. control expression must be integral
2. each case label is a keyword case followed by constant expression
3. no to case labels can represent the same value
4. two case statements can be associated with the same executable statements
5. the default label is not required
6. there is a maximum of one default label

```
switch (condition)
{
  case #:
    //expressions
  case #:
    //expressions
default:
  //expressions
}
```

Chapter 6- Repetition

rapidly repeating operations or series of operation
loop- using a construct to repeat designated statements under a predetermined set of conditions
loop iteration- one execution of the instructions inside the body of the loop construct
loop control expression (LCE)- the logical expression to determine whether the loop should initiate another iteration
loop initialization- preparation needed to be completed before the first evaluation of the loop control expression
loop update—an executable statement found inside the body of the loop that when executed enough times will lead to a change

in the evaluation of the loop control expression
loop control variable (LCV)- the variable that is initialized, part of the loop control expression and the recipient of the update action
pretest top- Prior to the start of each iteration the loop control expression is evaluated and while the expression remains true the repetition continues

- the minimum number of times the body of the pretest loop will iterate is zero
- the loop control expression will be. evaluated 1 more time that the body of the loop is ran

posttest loop- the loop control expression will be evaluated at the end. of each iteration to determine whether to continue with another iteration

- the minimum number of times the body of the pretest loop will iterate is one
- the loop control expression will be evaluated the same number of times as the body of the loop is running

counter controlled process- goes a specific number of times when can be determined. before the loop is ran
event controlled process- goes until a specific event occurs cannot be determined before the start of a loop

- input validation
- we do not have to validate. a particular type of data we only have to check that it is in acceptable range of values
- must put an appropriate message so that the user is aware of their mistake

while loop

- pretest loop
- minimum of 0 iterations

```
while (loop control expression)
{
  //expression
}
```

- there is no semicolon after the loop control expression- this will result in an infinite loop

do-while loop

- post test loop
- minimum of 1 iterations

```
do
{
  //expression
}while(loop control expression);
```

need {} for body of all loops for course standards

nested loop

loop inside of a loop

- it is a repetitive process inside another repetitive process
- inner loop must finish for next iteration of outer loop

infinite loop

logical error that corrects in condition of the loop control expression will never be false

- use control C to terminate program

for loop

- pretest loop
- control controlled process (cannot be used as an event-controlled process)
- brings together initialization loop control expression and update to a single line of code
- course standard. to use all three expressions in a for loop
- use. for loop if it it's a counter controlled process and all three expressions will be utilized

```
for (initialization; LCE; update)
{
  //expression
  //expression
}
```

```
int I;
for (i=0; i<5; i++)
{
  //expression
  //expression
}
```

recursion- function that calls itself

- recursive case- the condition when another call is made
- base case- when function stops calling itself

- recursion repeats until something stops it from calling itself and then returns up the chain
- each time a function is called the local variable occupy more memory
- when the number of times the recursive function is large or is unpredictable don't use recursion (stack overflow error)

Chapter 8- Arrays

- consecutive series of variables that all share one name and data type.

Each element (or variable) is associated with an index value that represents is position in the array
arrays are zero based- they start at the index zero
declaring and defining array

- informs the compiler of the name of the array the size of the array. and the data type of each element

```
//declaration
int function(int);
//call
var=function(x[i]);
//definition
int function(*var)
{
  return var+1;
}
```

- all arrays must be a fixed length (until introduced to dynamic memory allocation

```
type name[size];
```

```
int students[15];
```

array can be initialized in the declaration statement

```
type name[size]={value, value, value};
```

if the values provided is less than the defined size the unassigned elements are filled with zeros
the index value is used to access and assign individual elements in an array
you cannot print all elements in an array in one print statement you must print each element individually
if we try to access value outside the array size

- the program could work as expected- going beyond the limits of the array and into memory that is not reserved for the purposes of the array my not be reserved for any other purpose
- the program executes but results are unexpected- may be reserved for other purpose and multiple edits to the same memory may leave an unexpected value being stored
- program crashes- attempts to access memory that does not exist or the program does not have the ability to access

when passing an induvial element of an array toa function pass it the same way you pass a variable
an individual element is being passed by value
the element of an array can be passed by address the same way as a variable

```
//declaration
void function(int*);
//call
var=function(&x[i]);
//definition
void function(int *var)
{
  *var=*var+1
}
```

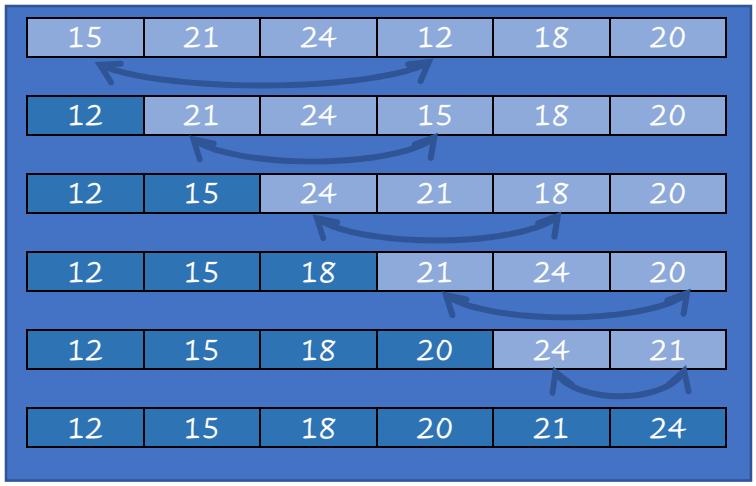
An entire array is passed by address where the value passed is the first item in the array (index 0)

- if x is an array (int x[3]={1,4,7}) then when x is passed to a function it looks like function(x)

the index of an array represents the offset from where the array begins in the memory of the computer
Index range violations- the address represented by the name of the array is added to the specified index to calculate a new memory address which may be beyond the memory specifically allocated for the array

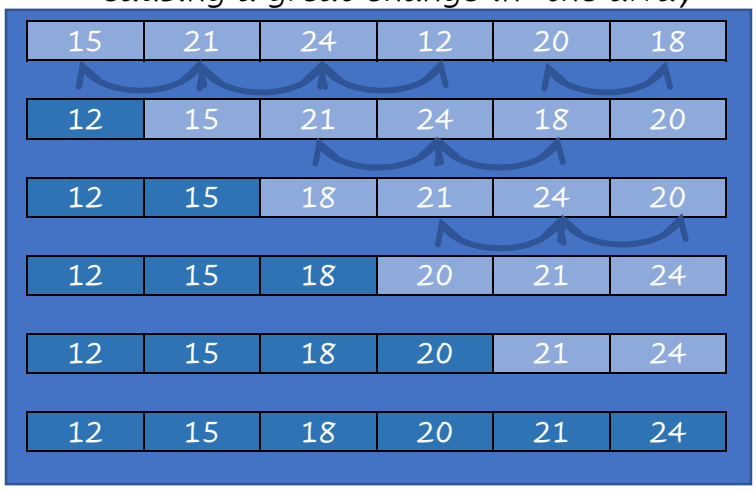
Sorting

Selection sort- traverse the unsorted list to identify the smallest (or largest value) exchange this value with the value in the sorted list that is adjacent to the sorted list. Repeat until the unsorted list is empty



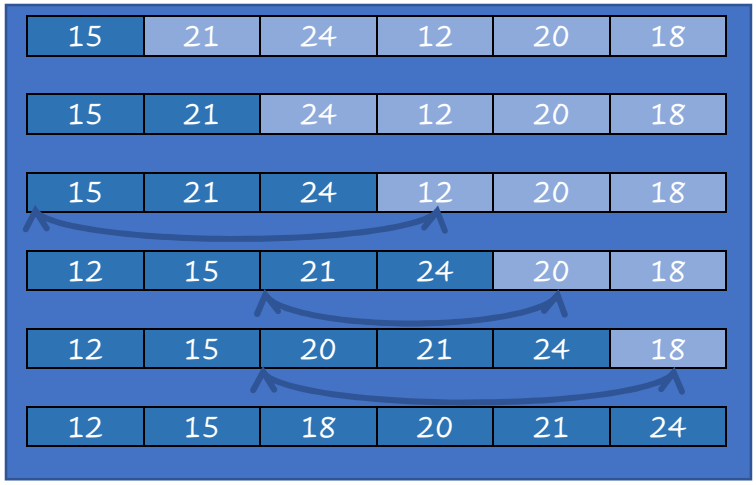
Bubble Sorting- traverse the unsorted list and compare the neighboring elements. Each time there is a pair of elements that are out of order swap them

- many swaps can occur in a single pass causing a great change in the array



Insertion Sort

- compare the value in the unsorted list that is adjacent to the sorted list and determine where among the values in the sorted list it is to be inserted



N-1 passes are required to sort a task of N elements

These algorithms are not optimized to end when the data reaches a sorted state we must be careful not to add logic that requires to much effort

Pass Number	Selection/Bubble	Insertion
1	1	2
2	2	3
3	3	4
4	4	5
5	6	6

Searching

- Process used to find the location of a target in a data set (location, number of occurrences, presence of the target in the data set

Sequential Search- traverse the array one index at a time until we find the target or search the entire array

- Every element is unique and data is not sorted

17	21	5	15	22	31	3	12	25	30
0	1	2	3	4	5	6	7	8	9

Search for 31

- 17 != 31
- 21 != 31
- 5 != 31
- 15 != 31
- 22 != 31
- 31 == 31 idx=5 STOP (if its not unique keeps going

- Performance varies based on values of the index in the array and the size of the array
  - If we have a very large set it takes a long time to check every element
- To verify element is not in array each element must be checked
- -1 is not a valid index in the array so we use it to determine if the target value was not found in the array
- If the target is found the loop ends and the index is returned
- If the end of the loop is reached, then the value is not found

Binary Search

- Used to reduce the time (average and worst case) to complete a binary search
- Data in the array must be sorted

- For each iteration we cut our data in half
    - If we sort the data frequently it is worth sorting
- Binary sort should only be used when the data set is large and searched frequently
- First- this integer will represent the lowest index value and which the target value may potentially be found in the array
    - When mid is greater than the target value last is set to mid-1
- Last- this integer will represent the highest index value at which the target value may potentially be found in the array
    - When mid is less than the target value first is set to mid+1
- Mid- the average of the first and last ((first + last)/2) will determine where in the array the next comparison will be made with the target value.
    - Updated at the start of every start of every iteration in the binary search process

1	3	4	6	8	10	13	15	16	20
0	1	2	3	4	5	6	7	8	9

For value 13

1	3	4	6	8	10	13	15	16	20
0	1	2	3	4	5	6	7	8	9
first				mid					last

1	3	4	6	8	10	13	15	16	20
0	1	2	3	4	5	6	7	8	9
					First Mid	last			

1	3	4	6	8	10	13	15	16	20
0	1	2	3	4	5	6	7	8	9
						First Last mid			

- Stops when first is greater than last (when value is not found or when mid=target value)

Multidimensional Array

Single dimension arrays- one row of indexed column values

Multidimensional arrays- array with more than one dimension (2 or 3 or 4 extra)

- The identifier will continue to represent the location where the memory allocation begins
- Declaration and definition tell the compiler the name, data type, size (extend) of each dimension

Each dimension is zero based
Intersection of every row and column represent the capacity to store a value of the defined type
Total capacity is the product of the extend of each dimension
When passing a multidimensional array to a function the first dimension extend does not need to be declared but the rest of them do (ie data[][COLS])

Chapter 9- Pointers and Pointer Applications

Pointer- variable that sores a memory address of another variable (though the memory address the data can be accessed and manipulated)
The address operation (&) returns the address of the memory allocated to a variable
"%p" placeholder is commonly used to print the address of a variable

Pointer Variables

As with all variables we must declare a pointer before we can assign a memory address to it
To initialize the pointer variable, we just put *identifier
If pointer is not initialized, then the pointer may be an invalid memory address and could result in a segmentation fault
Indirection operator (*) places before pointer variable to access the value stored at the location to which the pointer references

Array and Pointers

The name of an array much like a point represents a memory location with the index representing the offset from the address

```
a[i]=*(a+i)
```

Increase a pointer by one ill advance it to the next element in the array no matter what type of data is in the array (different data types require different amounts of memory)

Memory Allocation

Malloc- allocates a specific number of bytes of memory. A pointer variable must be used to reference the starting address of the newly allocated memory returned from the malloc function
Sizeof() operator will accept a data type and return the number of bytes need to store one value of this type
Data=(int*)malloc(sizeof(int)*20)

Chapter 10 & 11- Strings

String- series of characters with the capability to be treated as a single unit
This delimiter character "\0" represents the end of the data within a character array
Declaration of string char str[11] = "Good Day"

- The remaining characters are filled with zeros

String Input/Output Functions

%s placeholder- represents a string and is used in both input and output functions
Size of string should be one more than the length of (1 delimeter)
The scanf function only reads data up until a new line ('\n') or any white space
Gets()- alternative to scanf

- Gets function will accept the input terminated by a new array and make delimiter-terminated string out of it
- Error with it- Can go beyond the size of the array

Getchar()- accepts input of a single character

- Put it in a while loop to terminate when defined size is reached or new line is entered

```
Int x[SIZE]

Do{
        x[index++]=getchar();
}while(index<SIZE && x[index-1]!='\n');
x[index-1]=0;
```

String.h functions

Strlen(string)- returns the number of characters in the string. Does not count the terminal delimiter character

- Malloc(sizeof(char)*(strlen(str)+1))

Strcpy(str1,str2)- takes the second argument and copy its value to the first (the contents of str2 are copied to str1)

- Copies characters until the delimiter is found (if there is not delimiter then the strcpy will continue past the
- Strncpy/strlcpy are safer because they stop after n characters which is another parameter passed to the function

Strcmp- accepts two strings and returns an integer indicating how the two strings are different

- If strings are equal the function return 0
- If it is different than the number outputted is the difference in the first characters that are not equal (strcmp("company", "coporate")=-5)