```
//Data Representations
//Binary - base 2, represented by 0s and 1s
//Decimal - base 10, represented by 0-9
//Hexadecimal - base 16, represented by 0-9 and A-F
```

```
//Data Type Memory Sizes
//int, float - 4 bytes
//double - 8 bytes
//char - can only hold one char on keyboard, 1 byte
//pointers - 4 bytes
```

```
//Change from type int to double
static_cast<double>(i)
```

```
//gcc steps
//$ g++ -s hello.cpp (turns cpp into assembly language)
//$ g++ -c hello.o (makes object file)
//$ g++ -o hello hello.cpp (produces executable named hello)
//$ g++ functions.o main.o -o myhello (links multiple .o files
  into one final executable)
```

```
//Test if a pointer is valid
//assert (b != NULL);
```

```
//Char array or array of chars
char arr[] = {'J','i','l','l'};

//C-string
char arr2[] = {'J','i','l','l','\0'};
char arr3[] = "Jill"; //automatically inserts 0 char at end

//The C++ string class methods
string fruit = "Apple";
int len = fruit.length(); //5
int pos = fruit.find('l'); //3
string part = fruit.substr(1,3); //get 3 characters starting at
position 1, "ppl"
fruit.erase(2,3); //remove 3 chars starting at position 2, "Ap"
fruit.insert(2, "ricot"); //"Apricot"
friot.replace(2,5,"ple"); //"Apple"

//Check out cctype for checks and conversions on characters
fruit[0].tolower(fruit[0]);
isalpha(fruit[0]); //Checks for a-z, not case sensitive
isalnum(fruit[0]); //Checks for a-z and 0-9, not case sensitive
```

```
//References vs. Pointers
void swapWithPointer(int* a, int* b) {
	int temp = *a;
	*a = *b;
	*b = temp;
}
void swapWithReference(int &c, int &d) {
	int temp = c;
	c = d;
	d = temp;
}
```

```
//Computer Functions
Allocates the computer's resources to the different tasks that
the computer must accomplish - Operating System
Converts a program from a high-level language to machine
language - Compiler
Stores a program while it is being executed - Main memory
Executes a program stored in main memory - Processor
Converts a program from one high-level language to another high-
level language - None
```

```
//Inverted Triangle
#include <iostream>
#include <cstdlib>
using namespace std;

int main(int argc, char* argv[]) {
	if(argc != 2) {
		cerr << "Usage: " << argv[0] << " width" << endl;
		exit(1);
	}
	string out = "";
	for(int i = 0; i < atoi(argv[1]); i++) {
		//Print spaces
		for(int j = 0; j < i; j++) {
			out += " ";
		}
		//Print stars
		for(int j = 0; j < atoi(argv[1])-i; j++) {
			out += "*";
		}
		out += "\n";
	}
	cout << out;
}
```

```
//Structs
//Use . (dot operator) when passing by value
//Use -> (dereference) when passing by address
```

```
//Inputs
int input1;
cin >> inputl;
```

```
//Precondition: The address of a valid LinkedList and a char value that may be an alphabet in either lower or
upper case
//Postcondition: Adds a new node with data element set to value to the end of the linked list.
void addToEndOfList(LinkedList* list, char value) {
	Node* n = new Node;
	n -> data = value;
	n -> next = 0;
	//Check if linked list is empty
	if(list -> head) {
		//List is not empty
		list -> tail -> next = n;
		list -> tail = n;
	}
	else {
		//List is empty
		list -> head = n;
		list -> tail = n;
	}
}

//Precondition: A char array with a given length, not necessarily a C-string but containing only letters of
the alphabet
//Postcondition: The address of a new LinkedList containing all the characters of the input array in the same
order, where each node of the linked list contains one character of the array.
LinkedList* arrayToLinkedList(char* arr, int len) {
	LinkedList* list = new LinkedList;

	for(int i = 0; i < len; i++) {
		addToEndOfList(list, arr[i]);
	}

	return list;
}

//Precondition: The address of a valid LinkedList and a char value that is an alphabet in either lower or
upper case
//Postcondition: Returns the number of occurrences of the given alphabet in either lower or upper case in the
linked list
//You must use an iterative implementation (loops).
int countCharIterative(LinkedList* list, char value) {
	Node* next = list -> head;
	int count = 0;

	while(next) {
		if(tolower(next -> data) == tolower(value)) {
			count++;
		}
		next = next -> next;
	}
	return count;
}

//Precondition: The address of the first node in a linked list and a char value that is a letter of the
alphabet in either lower or upper case.
//Postcondition: Returns the number of occurrences of the given char value in the linked list using a
recursive implementation.
int countCharHelper(Node* head, char value) {
	if(!(head -> next)) {
		//Last node in list
		if(head -> data == tolower(value)) {
			return 1;
		}
		else {
			return 0;
		}
	}
	else {
		if(head -> data == tolower(value)) {
			return 1 + countCharHelper(head -> next, value);
		}
		else {
			return countCharHelper(head -> next, value);
		}
	}
}
```