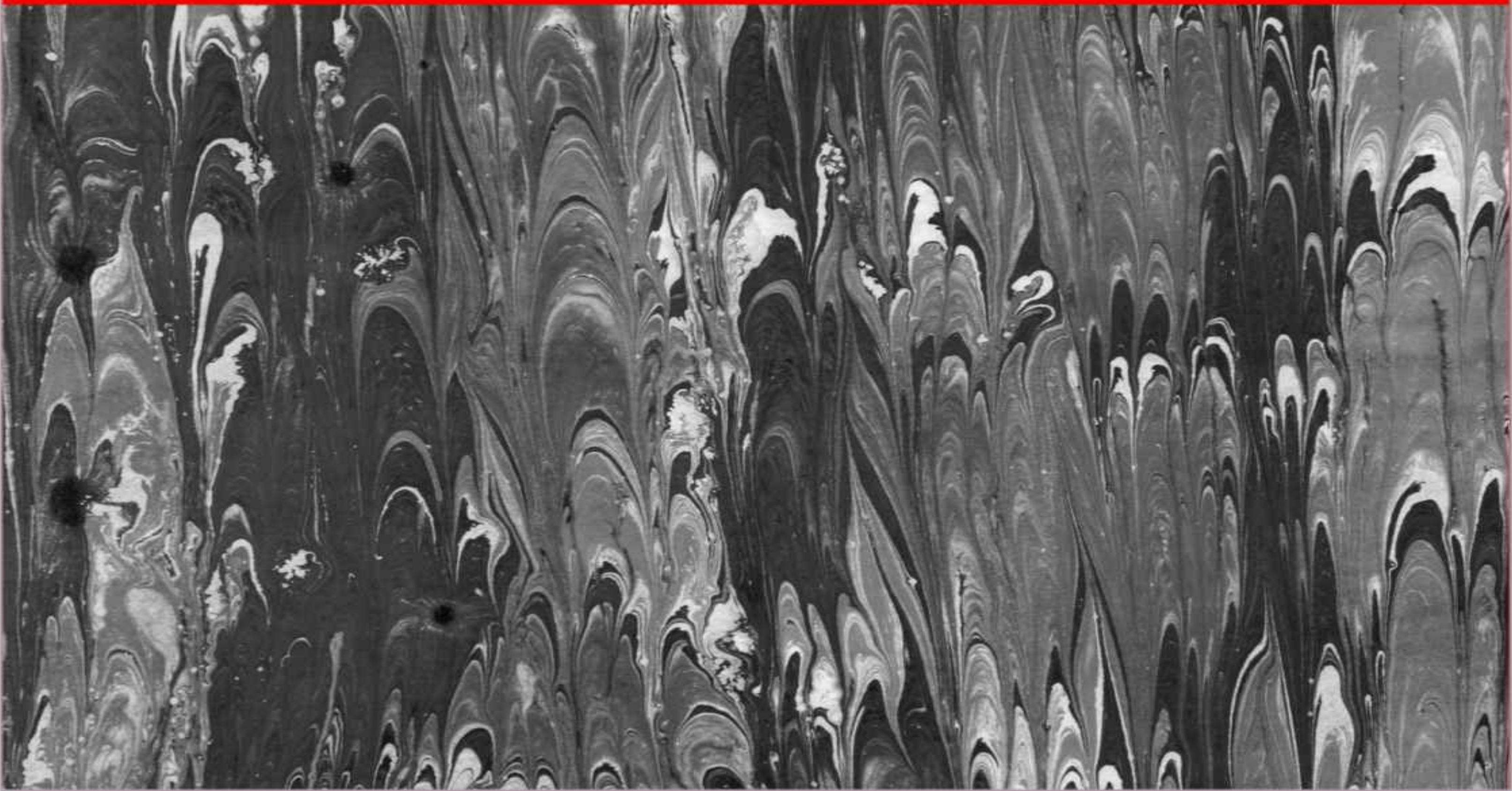
C Examples

Over 50 Examples



C-Programming Examples

Author’s Note

Thank you very much for your purchase, and your interest in learning a wonderful programming language. “C Examples” contains over 30 examples that are fully functional, easy to use, and unique. Previous programming knowledge is not required, however previous knowledge can do nothing but help. The only requirement is patience, dedication, and a passion for learning. However, as to not deter you from learning C I am going to explain how to create, compile, and run a “.c” source file.

In order to create an editable “.c” source file you first need to choose a text editor, I am going to use emacs. So to create a “.c” source file type the following command into your command line:

```
emacs SourceFile.c
```

Then to compile use the following command:

```
gcc -o SourceFile.c SourceFile
```

Then to run your file, use:

```
./SourceFile
```

New topics are presented in this book in an easy to understand way, then programs are made using the new topic as well as previously discussed topics in order to review and learn new material simultaneously. This helps you maximize learning a new language. There is much to learn and practice makes perfect. So let us begin.

Thank You

HelloWorld.c

```
#include <stdio.h> //a header file, this includes input and output
                //so we can use the printf function below
//how you comment
/*
 [author] | [email] | [date]
 [FileName]: [Description]
HelloWorld: Discusses how to set up a .c file. This is just a test to
 make sure your environment is set up properly.
*/

int main(){ //what is ran when you run a c program
  printf("Hello World!\n”);//‘\n’ is a newline character, it puts
                        //output onto a new line
    return 0;//return for main, no errors
}
```

OutPut:

```
Hello World!
```

DataTypes.c

```
#include <stdio.h>
/*
[author] | [email] | [date]
DataTypes: The types of data available to use within C.
Notice the sizeof() function, it returns the size (in bytes) of the data type. %lu is a type of
placeholder that holds an unsigned long. There are many data types not listed here, these are just
some basic types. Data types include: char, int, float, and
double.
*/

int main(){

  /*
    int and long: are whole numbers
    float and double: are floating point values (decimal numbers)
    char: is a single character
         %f -> a placeholder for floats/doubles
         %.2f -> formats a float/double output to two decimal places
         %d -> a place holder for ints
         %c -> a place holder for chars
   */
  printf("Storage size for char : %lu byte \n", sizeof(char));
  printf("Storage size for int : %lu bytes \n", sizeof(int));
  printf("Storage size for float : %lu bytes \n", sizeof(float));
  printf("Storage size for double : %lu bytes \n”, sizeof(double));
  return 0;
}
```

OutPut:

```
Storage size for char: 1 byte
Storage size for int: 4 byte
Storage size for float: 4 byte
Storage size for long: 8 byte
```

Variables.c

```
#include <stdio.h>
/*
[author] | [email] | [date]
Variable: Shows how to declare and initialize a variable. Multiple declarations are legal, just
initialize on separate lines. You can only perform operations on similar data types. To initialize
means to
set the variable equal to a value.
*/

int main(){
  char letter; //declaration of a variable
  letter = 't'; //initializing the variable
          //notice char is init with ‘’
  printf("letter: %c \n", letter); //%c placeholder for a char

  int x, y, z;  //notice multiple variable
          //declarations on the same line
  x = 3;
  y = 222;
  z = x+y;
  printf("z: %d \n",z);

  return 0;
}
```

OutPut:

```
letter: t
z: 225
```

Operators.c

```
#include<stdio.h>
/*
[author] | [email] | [date]
Operators: Basic operators in C, when you perform operations with operators, the operations
must be on the same data type. Operators
include: arithmetic operators, relational/comparison operators, and
assignment operators. There are more but this is all we need for now.
*/
int main(){
  int x, y;
  x = 2;
  y = 2;

  /*Arithmetic Operators*/
  printf("2 + 2 : %d \n", x+y);
  printf("2 - 2 : %d \n", x-y);
  printf("2 * 2 : %d \n", x*y);
  printf("2 / 2 : %d \n", x/y);
  x++; //same as x = x + 1
  printf("2++ : %d \n", x);
  y—; //same as y = y+1
  printf("2-- : %d \n", y);
  int m = 3%2; //returns the remainder of a quotient
  printf("modulus : %d \n", m);

/*Relational Operators*/
//returns 0(representing false) or 1(representing true)
  printf("2 == 2 : %d \n", x==y);//not an assignment, a comparison
  printf("2 !=  2 : %d \n", x!=y);//bang operator,
  printf("2 > 2 : %d \n", x>y); //will return 0
  printf("2 >= 2 : %d \n", x>=y);
  printf("2 < 2 : %d \n", x<y);
  printf("2 <= 2 : %d \n", x<=y);

/*Assignment Operators*/
  printf("x += y : %d \n", x+=y);//x = x+y
  printf("x: %d \n",x);//x=4
  printf("x -=  y : %d \n", x-=y);//x = x-y x:4 y:2
  printf("x: %d \n",x);//x=2
  printf("x *= y : %d \n", x*=y);//x = x*y
  printf("x: %d \n",x);
  printf("x /= y : %d \n", x= x/y);//x = x/y
  printf("x: %d \n",x);
  printf("x %%= y : %d \n", x%=y); //x = x%y
         //two % signs needed to print out a % sign
  printf("x: %d \n",x);

  return 0;
}
```

Output:

```
2 + 2 : 4
2 - 2 : 0
2 * 2 : 4
2 / 2 : 1
2++ : 3
2—- : 1
modulus : 1

2 == 2 : 1
2 != 2 : 0
```

```
2 > 2 : 0
2 >= 2 : 1
2 < 2: 0
2 <= 2 : 1

x += y : 4
x : 4
x -= y : 2
x : 2
x *= y : 4
x : 4
x /= y : 2
x: 2
x %= y : 0
x : 0
```

IfStatement.c

```
#include <stdio.h>
/*
[author] | [email] | [date]
IfStatement: Illustrates if statements and decision making. If
statements controls the flow of your program (they decide which code gets ran and which code
does not based off of some true/false
```

```
statement.
*/

int main(){
  int x = 1;
  int y = 2;

  if(x==y){
    //if this statement passes all other ‘else if’ and ‘else’
    //statements will not execute
    printf("Passed the if statement \n");
  }
  else if(x!=y){
    //if this passes and previous fails this code will run
    //and the else statement will run
    x=y;
    printf("x was changed... \n");
  }
  else{
    //will run if both(or all) if and else if statements fail
    printf("in else statement \n");
  }
  printf("x: %d \n",x);

    return 0;
}
```

OutPut:

```
x was changed...
x: 2
```

CoinFlip.c

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
/*
Author | Email | Date
CoinFlip: Simulates a coin toss by random
number generation. Random number generation uses time so we need to
import time.h and stdlib.h for this to work. Random numbers make your
programs more interesting.
*/
int main(){

  srand(time(0));
  int r = rand()%2 + 1;//gives us a range of [1,2]

  if( r == 1)
    printf("Heads\n");
  else
    printf("Tails");
  return 0;
}
```

Output1:

```
Heads
```

Output2:

```
Heads
```

Output3:

```
tails
```

LogicalOperators.c

```
#include <stdio.h>
/*
DoEasy | [email] | [date]
Logical Operators: Logical operators include the and operator, &&. The or operator, ‖, and the
bang operator, !. The bang operator allows you to reverse a statement, it produces the opposite
result. Logical operators allow you to chain together true/false statements.
*/
int main(){
  int a, b, c;
  a = 100;
  b = 100;
  c = 99;
  if(a == b && a != c){//switch to ==
    printf("&& returns: %d\n", (a==b && a != c));//logical op
  }
  else if(a == b ‖ a == c){//switch to !=
    printf("‖ returns: %d\n", (a==b ‖ a == c));//logical op
  } else{
    printf("No check passed :(\n");
  }

  return 0;
}
```

Output:

```
&& returns: 1
```

TernaryOperator.c

```
/*
Author | Email | Date
TernaryOperator: A ternary operator is pretty much shorthand
notation for if blocks. They are also called conditional
operators.
*/
int main(){

  int number = 99;//this number can be anything

  int outcome;
  //below is shorthand notation for:
  //if 99%2 equals 0 outcome = 1
  //else outcome = 0
  outcome = (99%2 == 0) ? 1 : 0;
  printf("outcome: %d\n",outcome);
  //this is a test to see if number is
  //odd or even, if number is evenly divisible
  //by two (the remainder is 0) then it is an
  //even number

}
```

Output:

```
outcome: 0
```

Switch.c

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
/*
Switch: How to use switch statements in C.
Switch statements are essentially if blocks,
but they switch on some condition. We are also
going to see how to generate random numbers to
make our programs more exciting.
*/
int main(){
  //we are going to simulate a dice roll
  //lets create a random number in between 0 & 6
  srand(time(0));
  int number = rand()%6+1;
  //number is now in between 1 and 6
  //so we must adjust below to make in the proper range

  //let's declare the switch statement
  switch(number){
  case 1:
    printf("You rolled a One\n");
    break;
  case 2:
    printf("You rolled a Two\n");
    break;
  case 3:
    printf("You rolled a Three\n");
    break;
  case 4:
    printf("You rolled a Four\n");
    break;
  case 5:
    printf("You rolled a Five\n");
    break;
  case 6:
    printf("You rolled a Six\n");
    break;


  }
}
```

Output:

```
You rolled a Two
```

Loops.c

```
#include <stdio.h>
/*
[author] | [email] | [date]
Loops: Outlines the types of loops. There are for, while, and do while loops, although do while
loops are not presented because they are very similar to while loops, with one exception, the
```

```
body is guaranteed to
execute at least once.
*/

int main(){

  /*
    To set up a for loop, we must initialize some sort of counter
    variable, then we must set up a condition to test each time,
    then we need some sort of step to keep the for loop going.
   */
  for(int i =0; i<=10;i++){//initialization, condition, step
    printf("i : %d \n",i);
  }

  //Multiplication table
  //this is called a nested for loop
  for(int i=1; i<=12;i++){
    for(int j=1; j<=12; j++){
      printf("%d \t", i*j);//will run 144 times
    }
    printf("\n");
  }

  //while loops are good for when you don’t know how long
  //your loop will be running for
  int x = 0;
  while(x<10){
    printf("x : %d \n", x);
    x++;//watch out for infinite loop, comment out to see what
           //happens…, terminate with cntrl+c
  }
    return 0;
}
```

OutPut:

```
i : 0
i : 1
i : 2
i : 3
i : 4
i : 5
i : 6
i : 7
i : 8
i : 9
i : 10
1       2       3       4       5       6       7       8       9       10      11      12
2       4       6       8       10      12      14      16      18      20      22      24
3       6       9       12      15      18      21      24      27      30      33      36
4       8       12      16      20      24      28      32      36      40      44      48
5       10      15      20      25      30      35      40      45      50      55      60
6       12      18      24      30      36      42      48      54      60      66      72
7       14      21      28      35      42      49      56      63      70      77      84
8       16      24      32      40      48      56      64      72      80      88      96
9       18      27      36      45      54      63      72      81      90      99      108
10      20      30      40      50      60      70      80      90      100     110     120
11      22      33      44      55      66      77      88      99      110     121     132
12      24      36      48      60      72      84      96      108     120     132     144
x : 0
x : 1
x : 2
x : 3
x : 4
x : 5
x : 6
x : 7
x : 8
x : 9
```

Scope.c

```
#include <stdio.h>
/*
[author] | [email] | [date]
Scope: Illustrates variable scope with a for loop. Scope is where you can use variables, methods,
etc, based off of where they were
declared.
*/

//variables declared out here have global scope
int main(){
  int i; //within the scope of the main method, i has local scope

  for(i = 5;i>0;i—-){     //try declaring i within for loop
    printf("i: %d \n”,i);
  }

  printf("i: %d \n", i);//will work because i is in scope of main
  //if i was declared in for loop, printf wouldn't run
  return 0;
}
```

Output:

```
i: 5
i: 4
i: 3
i: 2
i: 1
i: 0
```

Functions.c

```
#include <stdio.h>
/*
[author] | [email] | [date]
Functions: How to declare, define, and use your own functions.
*/

/*
To Set Up a function you must use this format:
  -Declare function above main
  -declare return type, method, name and parameter list
  -return type can be any legal data type
  -function name should be in lower case
  -param list can be as large or small as needed
  -define the function below the main method
  -call the function within the main method
*/
//declaring function
void add(int i);//global scope
int a = 0;//global scope


int main(){
  //local scope within the main function
  int a = 100;
  printf("a before function: %d\n", a);
  add(223); //calling(using) the function
  printf("a after function: %d\n", a);
  return 0;
}

//defining the function
void add(int i){ //must be declared outside of the main function
  a += i;//a = a+i;
}
```

Output:

```
a before function: 100
a after function: 323
```

Recursion.c

```
#include<stdio.h>
/*
Recursion: A recursive function is a function that calls
itself. Infinite recursion is possible so beware of this.
Also, you must have a proper base case in order to avoid this.
Recursive functions are somewhat easier to read and write
however they do take up more space on the heap so they do take longer to execute. However, for this size program we will be alright.
*/

int factorial(int n);
int main(){
  int n = factorial(5);
  printf("5!: %d\n",n);
  printf("6!: %d\n", factorial(6));

  return 0;
}
```

```
/*
Calculates the factorial of n.
*/
int factorial(int n){
  //base case
  if (n == 0)
    return 1;
  //recursive step
  else
    return n * factorial(n-1);
}
```

Output:

```
5!: 120
6!: 720
```

FibRecursion.c

```
#include<stdio.h>
/*
Author | Email | Date
FibRecursion: Calculates the fibonacci number up
to the desired place.
*/

int fibonacci(int n);

int main(){

  int fibArray[10];
  printf("Populating Array\n");
  for(int i = 0; i < 10; i++){
    fibArray[i] = fibonacci(i);
    printf("fibArray[%d]: %d\n",i, fibArray[i]);
  }
  return 0;
}

int fibonacci(int n){
  //base case
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Output:

```
Populating Array
fibArray[0]: 0
fibArray[1]: 1
fibArray[2]: 1
fibArray[3]: 2
fibArray[4]: 3
fibArray[5]: 5
fibArray[6]: 8
fibArray[7]: 13
fibArray[8]: 21
fibArray[9]: 34
```

Array.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
Array: Shows two ways to initialize arrays, and how to access array data members. Also, you can make 2-D arrays, and so on as needed.
*/
```

```
int main(){
  /*Two Ways to Initialize arrays*/
  //below will make array just big enough
  int list[] = {1, 2, 3, 4, 5, 6, 7, 8,9 ,10};
  printf("byte size of int[]: %lu\n", sizeof(list));
  //second way to initialize an array
  int forList[5];//set aside memory in space
  for(int index = 0; index < 5; index++){
    forList[index] = index*2;
    printf("Array[%d] : %d\t", index, index*2);
  }
  printf("\n");
  list[0] = forList[4];//legal
  printf("list[0]: %d\n", list[0]);
  list[0]=10001; //legal to initialize array
  printf("list[0]: %d\n", list[0]);

/*
2-D Array, an array within an array. Organized by row, column, notice double for loop.
*/
//init with row and column
int box[4][4]={{1, 2, 3, 4},
              {11, 12, 13, 14},
              {21, 22, 23, 24},
              {31, 32, 33, 34}};
  for(int r = 0; r<4;r++){
    for(int c = 0; c<4; c++){//nested for loop
      printf("box[%d][%d]: %d\t", r, c, box[r][c]);
    }
    printf("\n");
  }

    return 0;
}
```

Output:

```
byte size of int[]: 40
Array[0] : 0 Array[1] : 2 Array[2] : 4 Array[3] : 6 Array[4] : 8
list[0]: 8
list[0]: 10001
box[0][0]: 1    box[0][1]: 2    box[0][2]: 3    box[0][3]: 4
box[1][0]: 11   box[1][1]: 12   box[1][2]: 13   box[1][3]: 14
box[2][0]: 21   box[2][1]: 22   box[2][2]: 23   box[2][3]: 24
box[3][0]: 31   box[3][1]: 32   box[3][2]: 33   box[3][3]: 34
```

Average.c

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
/*
  Author | Email | Date
  Average: Calculates the average of a random int[].
*/
int main(){

  srand(time(0));
  int numbers[10];
  int total = 0;

  for(int i = 0; i < 10; i++){
    numbers[i] = rand()%101;// includes [0,100]
    total += numbers[i];
  }

  printf("Array Populated:\n");
  double average = total/10;//ten total numbers, or use sizeof
  printf("Average: %.2f\n", average);

  return 0;
}
```

Output:

```
Array Populated:
Average: 39.00
```

String.c

```
#include <stdio.h>
#include <string.h> //notice new header file
/*
[Author] | [Email] | [Date]
String: A string is a one dimensional array of chars.
There are a couple different ways to init a char[].
There are many functions to run on strings.
A string is terminated by a null character, or a null byte '\0'
*/

int main(){

  char name[5] = {'F','r','e','d','\0'};
  //make sure to leave one space for the null byte
  char nameTwo[] = "Robert";
  //will automatically make array just big enough,
  //will also add null byte
  printf("name: %s\n", name);//place holder for string is %s
  printf("nameTwo: %s\n", nameTwo);

  //there are many functions in string.h header file
  //one is strcat(s1, s2) concatenates s2 onto s1
  char nameAndSurname[14];//make sure this is big enough to hold what
                          //you are concatenating on
  strcat(nameAndSurname, "Tommy Pickles");
  printf("nameAndSurname: %s\n",nameAndSurname);
  return 0;
}
```

Output:

```
name: Fred
nameTwo: Robert
nameAndSurname: Tommy Pickles
```

StringArray.c

```
#include <stdio.h>
#include <string.h>
/*
[Author] | [Email] | [Date]
StringArray: How to declare an array of strings.
*/

int main(){
  //declared just like this, or with a for loop, must know size
  char *array[6]={"This","is", "an", "array", "of", "strings"};
  for(int i = 0; i<6; i++){
    printf("Word %d: %s\n", i, array[i]);
  }

  return 0;
}
```

Output:

```
Word 0: This
Word 1: is
Word 2: an
Word 3: array
Word 4: of
Word 5: strings
```

Struct.c

```
#include <stdio.h>
#include <string.h> //notice the include statement
/*
[Author] | [Email] | [Date]
Struct: Short for data structure a struct is  grouping of separate data types, just like a basic variable(like an int), memory space is set aside when you declare a struct.
*/

//declaration outside of main
struct Person{
  char name[20];
  int age;
};//notice the semi colon at the end

int main(){

  //declaring a struct variable
  struct Person personOne;
  //length == 24 bytes
  printf("size of personOne: %lu", sizeof(personOne));
  //initializing a struct variable
  strcpy(personOne.name,"DoEasy Productions!");
  personOne.age = 1;
  //dot operator allows you to access
  //data members within a dat structure
  return 0;
}
```

Typedef.c

```
#include <stdio.h>
#include <string.h>
/*
[Author] | [Email] | [Date]
```

```
TypeDef: Explains how to use typedef, a keyword in C. Typedef allows you to rename a type in
order to call a type by its new name. They are useful with structs.
*/
int main(){
  //typedef is a keyword, so it can't be used anywhere else
  //it is used to give a type a new name

  typedef unsigned char BYTE;
  //now we can use BYTE instead of ‘unsigned char’
  BYTE b;//8 bits
  b = (2*2*2*2*2*2*2*2)-1;
  printf("Max value BYTE can hold: %d\n",b);
  BYTE four[4];//32 bits

  //can also do something like:
  typedef struct Person{
    char name[10];
    int age;
  }Person;

  Person doEasy; //notice no struct
  strcpy(doEasy.name,"DOEASY");
  doEasy.age = 100;
  printf("Name: %s\nAge: %d\n",doEasy.name, doEasy.age);

  return 0;
}
```

Output:

```
Max value BYTE can hold: 255
Name: DOEASY
Age: 100
```

BitField.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
BitField: How to change the bit length of data types.
*/

//will change the width of an int to 1 bit,
//instead of 8 bytes (32 bits)
struct{
unsigned int b : 1;
}Boolean;

int main(){

        Boolean.b = 0;//legal
        Boolean.b = 1;//legal
        //Boolean.b = 2//not legal unless you assign a width of 2 bits
        //because it takes two bits to express 2 in binary
        //Boolean.b = 7 // illegal for same reason above, except you
            //would have to change the bit width to 3
  return 0;
}
```

Pointer.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
Pointer: A pointer is a special variable which stores the memory address of a data type. A pointer
points to the memory address where a variable’s data is stored. When you declare a variable in
C, memory space is allocated to store that variable. So a pointer can retrieve the data stored at a
memory address. When you access the value at a memory address, this is called dereferencing the
pointer. You can have a pointer to a pointer.
*/

int main(){

  int number = 1;
  int two;

  //to access a memory address use the & symbol
  //next to variable name
  printf("Address of number: %p\n", &number);
  printf("Address of two: %p\n", &two);//placeholder for pointer is %p
  //these two addresses will be separated by 4 bytes, or 32 bits
  //in memory and since they were declared one after the other they
  //are stored in a “linear” fashion
  //to declare a pointer use the same data type
  //in this case int then an * next to the name
  int *pointerToInt = &number; //declaring a pointer variable
                               //that points to address of number

  //just referencing the pointer will return an address
  printf("Address of where int is stored: %p\n", pointerToInt);

  //to access what the pointer is pointing to use a * on the pointer
  //this is called referencing a pointer
  printf("What is stored in number address: %d\n", *pointerToInt);
return 0;
}
```

Output:

```
Address of number: 0x7fff5f315be8
Address of two: 0x7fff5f315be4
Address of where int is stored: 0x7fff5f315be8
What is stored in number address: 1
```

PointerToNull.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
PointerToNull:Null in C is nothing, it literally does not have a value and is used in place of data that is unknown.
Pointers to null are legal within C. When a pointer is assigned
to null the pointer does not point to anything.
Two null pointers will always be equal.
Pointers to null do not point to anything.
The address stored in a pointer to null is an invalid memory address.
They are useful to perform checks.
*/
int main(){
  int *p;
  p = NULL;
  int *cp = NULL;
  printf("p: %p\n",p);//%p is a placeholder for a pointer
  printf("p==cp: %d\n",p==cp);//1 is true, 0 is false

  //can use a pointer to null in an if statement
  if(p){//can use p!=NULL
    printf("p is not null\n");
  }else {
    printf("p is null\n");
  }

  //NULL is nothing, pointer to null are useful
  //in data structures to signify
  //the end of the data structure, for example a linked list
return 0;
}
```

Output:

```
p: 0x0
p==cp: 1
p is null
```

PointerToVoid.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
PointerToVoid: Pointers to void are legal within c. A pointer to void
can be used to hold a pointer to any data type. Pointers to void
display traits of polymorphism.
*/
int main(){
  //Pointer to void can hold any data type
  int a = 100;
  int *p = &a;
  printf("Address of p: %p\n",p);
  printf("sizeof(p): %lu\n", sizeof(p));
  printf("Dereferencing of p: %d\n",*p);
  void *vp;
  vp = p;//this pointer can hold any data type,
         //useful for function parameters
  /*
    -a pointer to void will have the same representation and memory
     alignment as a char
    -a ptr to void will never == another ptr, yet two pointers to void
     assigned to NULL will ==
   */
  printf("Address of vp: %p\n",vp);
  printf("sizeof(vp): %lu\n", sizeof(vp));
  return 0;
}
```

Output:

```
Address of p: 0x7fff5c39cba8
sizeof(p): 8
Dereferencing of p: 100
Address of vp: 0x7fff5c39cba8
sizeof(vp): 8
```

PointerToConstant.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
Pointers to a constant are legal within c. Once a pointer
to a constant is initialized it cannot be changed. Constant is a
modifier you can add to a variable. Once a variable has been defined
as constant it cannot be change later. Constants help to secure your code.
*/
int main(){

  int x = 100;
  const int *pci = &x;
  printf("Address: %p\n",pci);
  printf("Dereference: %d\n", *pci);
  //this is not legal so it is commented out
  //*pci = 200;//illegal
  //if pci was not a pointer to a constant you could
  //change the value by dereferencing

  return 0;
}
```

Output:

```
Address: 0x7fff54958ba8
Dereference: 100
```

MultiplePointers.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
You can have multiple pointers pointing to the same memory address.
But each of those pointers will have different addresses.
Be careful doing this because memory addresses can become lost
if done improperly. Please note in this example, there is also a
reference of a pointer to a pointer.
*/
int main(){
  int x = 1000;
  int *p1 = &x;
  int *p2 = &x;
  //pointers have addresses themselves
  //and can be retrieved using & operator
  printf("Memory address of x: %p\n", p1);
  printf("Memory address of p1: %p\n", &p1);
  printf("Dereference of p1: %d\n", *p1);

  printf("Memory address of x: %p\n", p2);
  printf("Memory address of p1: %p\n", &p2);
  printf("Dereference of p2: %d\n", *p2);

  return 0;
}
```

Output:

```
Memory address of x: 0x7fff5bf59ba8
Memory address of p1: 0x7fff5bf59ba0
Dereference of p1: 1000
Memory address of x: 0x7fff5bf59ba8
Memory address of p1: 0x7fff5bf59b98
Dereference of p2: 1000
```

PointerToArray.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
PointerToArray: When you declare an array, and try to access this
array by name an address to the first index will automatically return. You can use the technique below to iterate through an array.
*/

int main(){
  int *p; //declare a pointer before
  int list[]= {777, 222, 100};//initialize array
  //if you use list, instead of list[n], a pointer to the first memory
  //address in the array is returned.

  p = list;//will give us an address, same as p = &list[0]
  for(int i=0; i<3; i++){
    printf("Memory Address of index %d : %p\n", i, (p+i));
    //this is called pointer arithmetic
    printf("*(p + %d) : %d\n",  i, *(p + i) );
    //same as printing list[i]
    printf("\n");
  }

    return 0;
}
```

Output:

```
Memory Address of index 0 : 0x7fff5ec02bbc
*(p + 0) : 777

Memory Address of index 1 : 0x7fff5ec02bc0
*(p + 1) : 222

Memory Address of index 2 : 0x7fff5ec02bc4
*(p + 2) : 100
```

PointerToStruct.c

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

```
/*
[Author] | [Email] | [Date]
PointerToStruct: You can have a pointer to a struct the same way you can have a pointer to any other data type. When you declare a pointer the memory address belongs to the first data type in the struct. You can iterate through a struct the same way we learned before, however printing them out may be more difficult.
*/

struct Person{
  char name[20];
  int age;
}; //notice the semi-colon

void printStruct(struct Person *person);

int main(){
  struct Person person;
  printf("Size of Struct: %lu\n\n",sizeof(person));
  strcpy(person.name, "DoEasy Prod.");
  person.age = 1;
  struct Person *p = &person;//memory address of name[]
  for(int i =0; i<2; i++){
    printf(“address[%d] : %p\n”,i,(p+i));
  }
  printStruct(p);
  return 0;
}
/*
Since we are using a pointer to a struct we pass a pointer to a struct as a parameter, to see what the pointer is pointing to we must use the -> operator, not the * operator. This is still dereferencing a pointer
however to dereference pointers of structs you must use the -> operator.
*/
void printStruct(struct Person *person){
  printf("Name: %s\nAge: %d\n", person->name, person->age);
}
```

Output:

```
Size of Struct: 24

address[0] : 0x7fff55633bb0
address[1] : 0x7fff55633bc8
```

Name: DoEasy Prod.
Age: 1

PointerToFunction.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
Explains what a pointer to a function is, and how to use them. You can have a pointer to a function
just like you can have a pointer to
anything else. You can even call a function by the pointer, below
illustrates how.
*/
//declaring a function
int add(int x, int y);
int main(){

  /*
   So when we declare a pointer to a function, we must do so with a
   specified format. First, declare the return type(it must be the
   same return type as the function. Next, name the pointer in our
   case the pointer name is ‘ptr’. Third, in parenthesis declare the
   parameter list the exact same as the function. In our case our
   functions takes in two arguments, both ints. So our pointer must do
   the same. Finally, set the pointer equal to the address where the
   function is stored.
   */
  int (*ptr)(int, int) = &add;
  int r = (*ptr)(10,3);
  printf("r: %d\n",r);
  return 0;
}

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

Output:

```
r: 13
```

Binary.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
Binary: Converts a char[], representing an 8 bit binary number, into a base 10 number, covers all topics discussed thus far. Good practice to try out some of the topics we have covered.
*/
//global scope, can be used anywhere within .c source file
char binary[8]= {‘0','0','0','0','1','1','0','1'}; //arrays
int binaryToDecimal(char binary[]); //functions

int main(){
  printf("decimal: %d \n", binaryToDecimal(binary));
    return 0;
}

int binaryToDecimal(char binary[]){
    char *p = binary; //pointer to first address in binary[]
    int sum = 0;
    int base = 1;
    for(int i = 7; i>0; i--){ //loops and operator
      //try i < sizeof(binaryNumber)/sizeof(char)
      if(*(p+i) == ‘1’){//pointer to array, and if statement
             sum += base;//assignment operator
      }
      base *= 2;//assignment operator
    }
    return sum;
  }
```

Output:

```
decimal: 13
```

TypeCasting.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
TypeCasting: How to type cast, and the subtleties of do so. When you type cast you convert one data type to the specified data type, but only for the one line in which you cast, unless you assign that cast to a variable.
*/
int main(){
  /*
    -Can use casting to pass different data types to functions
    -Can cast any data type to another data type
    -Watch out for data loss, like casting a float to an int
   */
  double x = 89.99;
  printf("x before cast: %f\n", x);
  printf("sizeof(x): %lu\n", sizeof(x));
  printf("x after cast: %d\n", (int)x);
  printf("sizeof((int)x): %lu\n", sizeof((int)x));

  //or instead of casting x to an int each time
  //we could assign it to a variable,
  //just make sure that variable is the same as the cast
  int castVariable = (int)x;
  return 0;
}
```

Output:

```
x before cast: 89.990000
sizeof(x): 8
x after cast: 89
sizeof((int)x): 4
```

IO.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
IO: We’ve been dealing with output all along with the printf()
function, now let’s try some input using the scanf() function.
Standard input comes from the keyboard, so now we can start to
create some really fun and exciting programs.
*/
int main(){
  //we have already been seeing input, but what if you
  //want to get user input from the keyboard?
  /*
    getchar() and putchar()
   */
  char c;
  printf("Enter a value>>> ");
  c = getchar();//only retrieves one character
  printf("\nChar Entered: ");
  putchar(c);
  printf("\n");
  /*
    scanf() and printf()
   */
  char input[100];
  int i;
  printf("Enter a value>>> “);
  //retrieves any pattern of data types you like
  scanf("%s %d", input, &i);//this can be any pattern you like
  printf("\nInput Entered: ");
  printf("%s %d", input, i);
  printf("\n");
  return 0;
}
```

Output:

```
Enter a value>>> s

Char Entered: s
Enter a value>>> string 4
Input Entered: string 4
```

GuessThePassPhrase.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/*
[Author] | [Email] | [Date]
A game to test out while loops, user input and
logic, the game will end when the user guesses our
password.
*/

     /*
    Returns 1 if guess == password
    Otherwise 0 is returned.
   */
```

```
int equals(char* guess, char* password);

int main(){
  char PASSWORD[] = “Password”;//this can be any password you like
  char* pp = PASSWORD;//this pointer holds
          //the first memory address of PASSWORD
  char guess[15];//guess can be no longer than 14 characters
          //because we need to account for the null byte
  do{
    printf("Guess the password: \n");
    scanf("%s", guess);
  }while(equals(guess, pp) == 0);
  printf("You win the password was: %s\n",PASSWORD);
  return 0;
}

     /*
    Returns 1 if guess == password
    Otherwise 0 is returned.
   */
  int equals(char* guess, char* password){
    int i = 0;
    while(*(password+i) != '\0'){
      if (*(password+i)!= *(guess+i))
             return 0;//it is ok to leave out curly brackets
           //for one line of code
      i++;
    }
    return 1;
  }
```

Output:

```
Guess the password:
these
Guess the password:
are
Guess the password:
some
Guess the password:
```

```
guesses
Guess the password:
Password
You win the password was: Password
```

TempConverter.c

```
#include<stdio.h>
#include<stdlib.h>
/*
[Author] | [Email] | [Date]
Converts a temperature in Fahrenheit to Celsius and Kelvin.
*/
int main(){
  printf("Enter degrees in F:\n");
  double tempF = 0;
  scanf("%lf", &tempF);

  double tempC = (tempF-32)*(double)5/9;
  double tempK = tempC + 273.15;

  printf("tempF: %.2f\n",tempF);;
  printf("tempC: %.2f\n",tempC);;
  printf("tempK: %.2f\n",tempK);;
}
```

Output:

```
Enter degrees in F:
0
tempF: 0.00
tempC: -17.78
tempK: 255.37
```

FileIO.c

```
#include <stdio.h>
#include <string.h>
/*
[Author] | [Email] | [Date]
  FileIO: In C you can read input from a file, and also write to a
  file from your program. fgets reads in one line from a file, fputs
  puts a line onto a file, when you do this however all content in the
  file is erased, and replaced with what is inside fputs, so be sure
  to use the append option if you want to avoid this.
*/
int main(){
  /*
    fgetc(fp) -->reads in a char
    fscanf(fp, char[], sourcefile)—> reads in a pattern
    fgets --> reads in lines
   */
  FILE *fp; //first declare a pointer to a file
  char contents[50];//can’t be longer than 50
  fp = fopen("tester.txt","r");
  //if fp == NULL {
  fscanf(fp, "%s", contents);
  printf("Word 1: %s\n",contents);
  fscanf(fp, "%s", contents);
  printf("Word 2: %s\n",contents);
  fscanf(fp, "%s", contents);
  printf("Word 3: %s\n",contents);

  for(int i = 0; i<4; i++){
  fgets(contents, 50,  fp);
  printf("Line %d: %s\n", i+1, contents);
  }
  fclose(fp);
  fp = fopen("output.txt","w");
  //doesn’t add a new line character
  fprintf(fp,"This will write into file above");
  fputs("This will also fo in the file",fp);
  return 0;
}
```

Output:

```
Word 1: This
Word 2: scans
Word 3: words
Line 1:

Line 2: this is another line

Line 3: file input and output

Line 4: can be very useful
```

ErrorHandling.c

```
#include <stdio.h>
#include <stdlib.h>
/*
[Author] | [Email] | [Date]
ErrorHandling: How to deal with errors, what to look for,
and stderr, part of the stdlib. We are going to use a simple example
of trying to divide by zero. When you divide by zero, a non real
number is returned. So we want to set up some precautions to avoid this, error handling is very
specific to your program.
*/

int main(){
  int numerator = 4;
  int denominator = 1;
  int quotient;

  //below is the error check
  if(denominator == 0){
    fprintf(stderr,"Error: Denominator == 0\n”);
    fprintf(Exiting with return status 1.\n");
    return 1;
  }
  quotient = numerator/denominator;
  fprintf("Quotient: %d", quotient);
  return 0;

}
```

Output:

```
Quotient: 4
```

MyHeader.h

```
/*
   In C you can create your own header files, which you then can
   include in any other C/C++ program. Structs are a common thing
   to place in a header file.

  */
  typedef struct Person{
  char name[10];
  int age;
}Person;

//can put methods in here,
//can put error handling within those methods as well
```

Header.c

```
#include <stdio.h>
#include <string.h>
#include “MyHeader.h”//notice the include
/*
[Author] | [Email] | [Date]
Header: Uses our self made MyHeader.h file.
*/
int main(){
  Person me;
  strcpy(me.name,"DoEasy");
  me.age = 100;
  printf("Name: %s\nAge: %d\n",me.name, me.age);
  return 0;
}
```

Output:

```
Name: DoEasy
Age: 100
```

ChemicalElement.h

```
/*
[Author] | [Email] | [Date]
A header file that contains a data structure
that mimics a chemical element.
*/
```

```
typedef struct ChemicalElement{
  //names cannot be longer that 19 characters
  //can you make this String dynamic?
  char name[20];
  int number;
  double mass;
  //anymore members you can think of?
}ChemicalElement;
```

PeriodicTable.c

```
#include"ChemicalElement.h"
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
/*
[Author] | [Email] | [Date]
PeriodicTable: A C source file where you can save chemical elements,
and maybe start creating molecules.
*/
int main(){

  ChemicalElement hydrogen;
  strcpy(hydrogen.name, "Hydrogen");
  hydrogen.number = 1;
  hydrogen.mass = 1.001;

  ChemicalElement oxygen;
  strcpy(oxygen.name, "Oxygen");
  oxygen.number = 8;
  oxygen.mass = 15.9994;

  ChemicalElement h2o[3];
  h2o[0] = hydrogen;
  h2o[1] = hydrogen;
  h2o[2] = oxygen;

  printf("Printing Water:\n");
  double totalMass=0;
  for(int i = 0; i < 3; i++){
    printf("%s: %.2f\n", h2o[i].name, h2o[i].mass);
    totalMass += h2o[i].mass;
  }
  printf("Total Mass: %.2f\n", totalMass);

  return 0;
}
```

Output:

```
Printing Water:
Hydrogen: 1.00
Hydrogen: 1.00
Oxygen: 16.00
Total Mass: 18.00
```

CommandLine.c

```
#include <stdio.h>
/*
[Author] | [Email] | [Date]
CommandLine: How to pass in command line arguments.
Please note they are passed as an array of strings
so to accept a different data type takes more work.
*/
int main(int argc, char *argv[]){

  if (argc == 1){
    printf("NO COMMAND LINE ARGS\n");
    return 0;
  } else {
    int i =1;
    while(i<argc){
      printf("Command Line Arg[%d]:%s\n", i, argv[i]);
      i++;
    }//end while
  }//end else


  return 0;
}
```

```
. /CommandLine These Are the Args
```

```
Command Line Arg[1]:These
Command Line Arg[2]:Are
Command Line Arg[3]:the
Command Line Arg[4]:Args
```

MemoryManagement.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*
[Author] | [Email] | [Date]
How to create memory space dynamically. This means how to create
memory space during run time. Useful for when you do’t know how much memory you will need.

void *calloc(int num, int size)-->allocates an array of num, elements, each with given size

void free(void *address)-->release a block of memory given by address

void malloc(int num)-->allocates an array of num bytes, which are uninitialized

void *realloc(void *address, int newsize)-->reallocates memory, extending to size

So we can define a pointer to a data type without defining how much memory is required
*/
int main(){

  char name[10];
  char *about_name;
  strcpy(name,"doeasy");

  about_name = malloc(100*sizeof(char));
  printf("sizeof(): %lu\n", sizeof(about_name));
  printf("address: %p\n", &about_name);
  strcpy(about_name,"This string can be as big as the malloc...");
  //so if you want to make this bigger, like n an error check or
  //based off of some user input you can use realloc()
  //then finally free up the space
  printf("name: %s\n",name);
  printf("about_name: %s\n", about_name);
  free(about_name);

  return 0;
}
```

Output:

```
        sizeof(): 8
        address: 0x7fff5c8e3b90
        name: doeasy
        about_name: This string can be as big as the malloc…
```

DynamicString.c

```
/*
Author | Email | Date
DynamicString: How to create a string dynamically.
Most times going into a program you do not know
how long your input is going to be, we will deal with
that right now.
*/
int main(){


  //first declare a pointer to a char,
  //using malloc set your pointer equal to one char length
  //malloc returns a void* so you must cast to the
  //data type we are woking with
  char* input = (char*)malloc(sizeof(char));
  int count = 0;

  //next we will get some user input of unknown length
  printf("Enter a sentence: ");
  char token;
  token = getchar();
  //can only read in one sentence, hit enter from command line
  //to keep program going
  while(token != '\n'){
    //realloc returns a void* so you must cast
    //to the type of pointer you are working with
    input = (char*)realloc(input, sizeof(char)*(count+1));
    input[count] = token;//add to array
    token = getchar();//get the next token
    count++;//increment our size
  }
  printf("You entered: %s\n", input);
  printf("Counted %d tokens.",count);
  return 0;
}
```

Output:

```
Enter a sentence: this sentence can be any length
You entered: this sentence can be any length
```

DynamicIntArray.c

```
#include<stdio.h>
#include<stdlib.h>
/*
[Author] | [Email] | [Date]
DynamicIntArray: A dynamic int array can grow or shrink
during run time. We are going to use much the same approach
as before, but with integers this time.
*/
int main(){

  int* numbers = (int*)malloc(sizeof(int));
  printf("Enters numbers, press enter to add a number:\n");
  printf("Enter a -1 to exit.\n");
  int input = 0;
  int count = 0;
  while(input != -1){
    scanf("%d", &input);
    numbers = (int*)realloc(numbers, sizeof(int)*(count+1));
    numbers[count] = input;
    count++;
  }
  printf("\nPrinting Numbers:\n");
  for(int i = 0; i<count-1;i++){
    printf("number[%d]: %d\n",i, numbers[i]);
  }
  printf("\nThat was %d numbers.\n",count-1);

  return 0;
}
```

Output:

```
Enters numbers, press enter to add a number:
Enter a -1 to exit.
0
22
222
-1

Printing Numbers:
number[0]: 0
number[1]: 22
number[2]: 222

That was 3 numbers.
```

ArrayAnalysis.c

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
/*
  Author | Email | Date
  ArrayAnalysis: Calculates the average, biggest, and smallest number
  of a randomly generated array. We are going to discuss how to return
  an array from a function, and how to pass an array as a parameter
*/

int* initArray(int size, int range);
double average(int* numbers, int size);
int biggest(int* numbers, int size);
int smallest(int* numbers, int size);

int main(){
  int* pi = initArray(10, 100);
  double ave = average(pi, 10);
  int big = biggest(pi, 10);
  int small = smallest(pi, 10);

  printf("Analyzing Array:\n");
  printf("\tAverage %.2f\n",ave);
  printf("\tBiggest %d\n",big);
  printf("\tSmallest %d\n", small);

  return 0;
}

/*
Creates an array of length size, with
random numbers in between [0, range].
When you return an array you must return
a pointer to an array, which you can later
iterate through.
*/
int* initArray(int size, int range){

  srand(time(0));
  int* num = (int*)malloc(sizeof(int));
  for(int i = 0; i < size; i++){
    num = realloc(num, sizeof(int)*(i+1));
    *(num+i) = rand()%range;
  }
  //this returns a pointer to the first
  //address of numbers
  return num;
}
```

```
/*
  average: Calculates the average of our array.
  What we are doing is passing an array into our function.
  More specifically we are passing a pointer into our function
  which we then can use pointer arithmetic on to
  dereference it's members. When you do this it is good practice
  to pass in the size as well.
*/
double average(int* numbers, int size){
  double total = 0;
  for(int i = 0; i < size; i++)
    total += *(numbers+i);
  return total/(double)size;
}

/*
  biggest: finds the largest number in an array.
  We will use the same technique as above.
*/
int biggest(int* numbers, int size){
  int biggest = *numbers;
  //initialize biggest to the first element
  for(int i = 1; i < size; i++){
    if(*(numbers+i) > biggest)
      biggest = *(numbers+i);
  }
  return biggest;
}

/*
  smallest: finds the smallest number in an array.
  We will use the same technique as above.
*/
int smallest(int* numbers, int size){
  int smallest = *numbers;
  //initialize biggest to the first element
  for(int i = 1; i < size; i++){
    if(*(numbers+i) <smallest)
      smallest = *(numbers+i);
  }
  return smallest;
}
```

Output:

```
Analyzing Array:
        Average 60.30
        Biggest 90
        Smallest 0
```

TicTacToe.h

```
/*
[Author] | [Email] | [Date]
The beginning to a TicTacToe game. It needs some work but that would be great practice.
*/
char board[3][3];
int player1 = 1;
int player2 = 0;

void initBoard(){
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      board[i][j] = ' ';
    }
  }
}

void printBoard(){
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 3; j++){
      printf("[%c]",board[i][j]);
    }
    printf("\n");
  }
}

/*
Put functions in here to check to see whose turn it is.
And to make moves.
*/
```

TicTacToeGame.c

```
#include <stdio.h>
#include "TicTacToe.h"
/*
[Author] | [Email] | [Date]
Exercises our TicTacToe.h header file. This file is where
the game is actually played. You can choose to make the game
for one or two players.
*/
int main(){

  initBoard();
  printBoard();
  board[0][1] = 'X';
  printBoard();

}
```

Output:

```
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][ ]
[ ][X][ ]
[ ][ ][ ]
[ ][ ][ ]
```

pieces.h

```
/*
 [Author] | [Email] | [Date]
pieces: Contains the pieces to a chess game.
*/
#include <stdio.h>
char *board[8][8];
char pawn = 'p';
char rook = 'r';
char night = 'n';
char bishop = 'b';
char queen = 'q';
char king = 'k';
char block = ' ';
char *lostBlackPieces[16];
char *lostWhitePieces[16];
void initBoard(){
  for(int i = 0; i<8;i++){
    for(int j = 0; j<8; j++){
      if(i==1 ‖ i == 6){
          board[i][j] = &pawn;
      }
      else if (i != 7 ‖ 1 !=0 ‖ i != 1 ‖ i != 6)
        board[i][j]=&block;
    }
  }
  board[7][0]=&rook;
  board[7][1]=&night;
  board[7][2]=&bishop;
  board[7][3]=&queen;
  board[7][4]=&king;
  board[7][5]=&bishop;
  board[7][6]=&night;
  board[7][7]=&rook;
  board[0][0]=&rook;
  board[0][1]=&night;
  board[0][2]=&bishop;
  board[0][3]=&king;
  board[0][4]=&queen;
  board[0][5]=&bishop;
  board[0][6]=&night;
  board[0][7]=&rook;
}
void printBoard(){
  for (int r = 0; r<8; r++){
    for(int c = 0; c<8; c++){
      printf("[%c]", *(board[r][c]));
    }
      printf("\n");
  }
}
```

moves.h

```
/*
 [Author] | [Email] | [Date]
movess: Contains the moves allowed in a chess game.
*/
int checkPawnMove(char *p, int fromr, int fromc, int tor, int toc){
  if(p != &pawn){
    return 0;
  }
  else if(tor-fromr == 1 || tor - fromr!=-1){
    return 0;
  }
  else if(tor < 0 || tor > 7){
    return 0;
  }
  else if(toc-fromc != 0){
    return 0;
  }
  else{
    return 1;
  }
}

/*
int jumpWithPawn(char *piece, int r, int c, int tor, int toc){

}
*/
void movePawn(char *piece, int r, int c, int tor, int toc){
  if(checkPawnMove(piece, r, c, tor, toc)==1){
    board[tor][toc] = piece;
    board[r][c] = &block;
  }
}
```

Game.c

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

```
#include "moves.h"
/*
Game: A game of chess. Here is where you will implement the header files to actually play a
game of Chess. The game is far from complete but it is a nice start. Have fun :)
*/
int main(){

  initBoard();
  printBoard();
  movePawn(board[6][0], 6, 0, 5, 0);
  printBoard();
  movePawn(board[5][0], 5, 0, 4, 0);
  printBoard();
  return 0;
}
```

Conclusion

In conclusion, we have only scratched the surface with these examples. However, now you have a great base on which to build. So get out there and start building, breaking, and having fun with your own code. Please check out my website for more information:

tfoss0001.github.io

For fun and interactive video tutorials check out my youtube channel:

DoEasy Productions

There are playlists on that channel to help guide you through these examples and to give you ideas for your own programs. Thank you very much.

-Torin Foss