C Review

Note: C Review DOES NOT mean I will actively go over 1310 topics. I will simply be doing programs in C at the level you are expected to be at after taking 1310…the purpose of today is just to give a refresher on C code

I will do code today that uses:

- 2D Arrays and nested loops (should be familiar with this from 1310)

Notice the use of **FUNCTIONS that I create** in all my code (we'll talk more about this later on, you should have learned this in 1310)…also, for functions that are used from the C library, get comfortable looking them up!

IF YOU ARE STRUGGLING WITH THE TOPICS COVERED IN THE NEXT FEW LECTURES, PLEASE SERIOUSLY CONSIDER DROPPING 1320 AND RETAKING 1310

Program 1 (using 2D arrays):

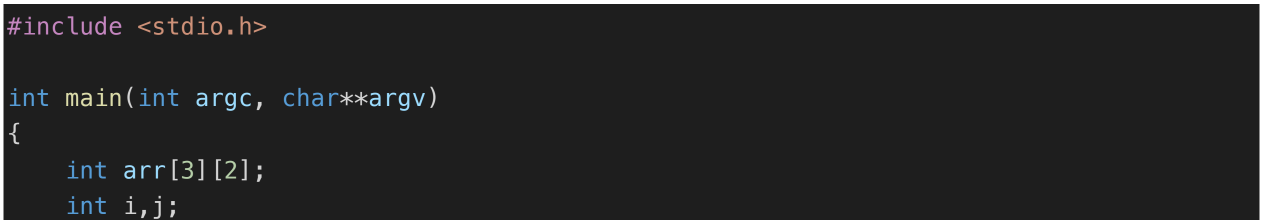
Before we start, quick review of 2D arrays (you are expected to be extremely comfortable with 2D arrays and nested loops from 1310):

Simple programs showing 2D arrays

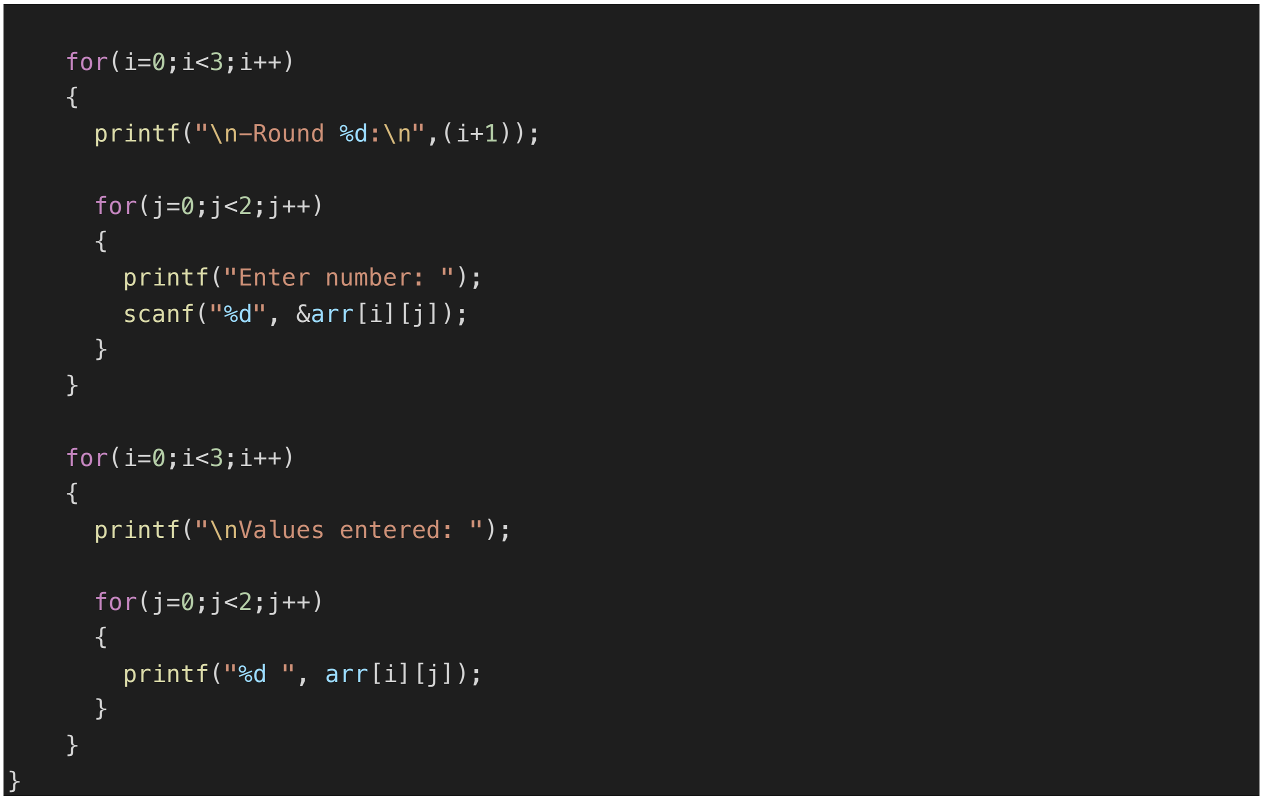
Print out numbers

Sample Run:

```
-Round 1:
Enter number: 1
Enter number: 2

-Round 2:
Enter number: 3
Enter number: 4

-Round 3:
Enter number: 5
Enter number: 6

Values entered: 1 2
Values entered: 3 4
Values entered: 5 6
```



```c
#include <stdio.h>

int main(int argc, char**argv)
{
    int arr[3][2];
    int i,j;
```



```
    for(i=0;i<3;i++)
    {
      printf("\n-Round %d:\n",(i+1));

      for(j=0;j<2;j++)
      {
        printf("Enter number: ");
        scanf("%d", &arr[i][j]);
      }
    }

    for(i=0;i<3;i++)
    {
      printf("\nValues entered: ");

      for(j=0;j<2;j++)
      {
        printf("%d ", arr[i][j]);
      }
    }
}
```

2d arrays of chars (printing out strings)

Sample Run:

```
cat
dog
bird
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
   char arr[3][20]={"cat","dog","bird"};
   int i,j;

   for(i=0;i<3;i++)
   {
     printf("%s\n",arr[i]); /*each word is printed out here*/
   }
}
```

2d arrays of chars (strings)-find a specific letter

Sample Run:

```
Found the letter a in cat!
Found the letter a in day!
--------------------------
```

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

int main(int argc, char**argv)
{
  char arr[3][20]={"cat","day","bird"};
  int i,j;

  for(i=0;i<3;i++)
  {
   for(j=0;j<strlen(arr[i]);j++)
   {
    if(arr[i][j]=='a') /*getting a single char*/
    {
     printf("Found the letter a in %s!\n",arr[i]); /*getting the whole string*/
    }
   }
  }
}
```

Davoud owns a very exclusive restaurant with three tables, labeled *1, 2* and *3*. In order to keep track of reservations, he has asked you to make a program that allows people to reserve and unreserve tables.

When coding remember that you are trying to convert the world around you into code. This means thinking "how can I represent some concept and idea in code?"

For example, we have a real-world scenario of a restaurant…how can we represent three tables? We can use an array, where each index represents a specific table. How can we represent the idea of a reserved table vs unreserved table? We can use the value 0 to represent unreserved and the value 1 to represent reserved. How can we represent the action of reserving/un-reserving a table? We can change these values between 0 and 1 at a specific index of the array. Always remember when coding that we are holding information (either in a variable or data structure like an array) or manipulating that information (held in either a variable or data structure) with some operation. Always try to reduce word problems to information you need to hold and actions you need to do with that information.

Sample run on my computer:

```
C computer$ ./a.out

--reserve a table, unreserve a table or exit? reserve
Table 1 is now reserved!

--reserve a table, unreserve a table or exit? reserve
Table 2 is now reserved!

--reserve a table, unreserve a table or exit? reserve
Table 3 is now reserved!
```

```
--reserve a table, unreserve a table or exit? reserve
Sorry, no tables available.

--reserve a table, unreserve a table or exit? unreserve
Table 1 is reserved.  Would you like to unreserve? y or n
n
Table 2 is reserved.  Would you like to unreserve? y or n
y
Table 3 is reserved.  Would you like to unreserve? y or n
y
Total unreserved: 2

--reserve a table, unreserve a table or exit? reserve
Table 2 is now reserved!

--reserve a table, unreserve a table or exit? exit
```

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

void reserve(int all_tables[][2],int size)
{
  int i;

  for(i=0;i<size;i++)
  {
    if(all_tables[i][1]==0)
    {
      all_tables[i][1]=1;
      printf("Table %d is now reserved!\n",(i+1));
      break;
    }
  }

  if(i==size)  /*we went through all available tables, nothing available (since the for loop ran completely and didn't enter the if statement, the value of i will be equal to size.  Note here btw that since we declared the i variable outside of the loop, the i value will exist after…if it was declared in the for loop, it wouldn't' be available here…this is a scope concept*/
  {
    printf("Sorry, no tables available.\n");
  }

}

/*returns total number of tables that were unreserved*/
int unreserve(int all_tables[][2],int size)
{
  int i, total=0;
  char answer[20];
  for(i=0;i<size;i++) /*go through each table*/
  {
    if(all_tables[i][1]!=0) /*If the table is reserved, ask if the user wants to unreserved…can also just say if (all_tables[i][1]) as
mentioned in class-if the value is 1 (reserved) it would count as true and the if would execute, otherwise if it is 0 it would count
as false and not execute*/
    {
      printf("Table %d is reserved.  Would you like to unreserve? y or n\n", all_tables[i][0]);
      fgets(answer, 20,stdin);

      if(answer[0]=='y') /*could also use strcmp since fgets takes everything as a string and even “y” would be a string*/
      {
        all_tables[i][1]=0;
        total++;  /*keep track of the total number of tables that get unreserved*/
      }
    }
  }

  return total;

}


int main(int argc, char **argv)
{
  char answer[20];
  int all_tables[][2]={{1,0},{2,0},{3,0}};  /*you don’t always have to intialize arrays, just showing you*/
  int full=1;

  while(full)  /*see note 1 below*/
  {
    printf("\n--reserve a table, unreserve a table or exit? ");
    fgets(answer, 20,stdin);   /*see note 2 below*/
    strtok(answer,"\n");  /*We will learn about this function more later this semester*/

    if(!strcmp(answer,"exit"))  /*could also do strcmp(answer,”exit”)==0…look up strcmp to see return value to see why
these work*/
    {
      full=0;
    }

    else if(!strcmp(answer,"reserve")) /*could also do strcmp(answer,”reserve”)==0…look up strcmp to see return value to
see why these work*/
    {
      reserve(all_tables,3);
    }

    else if(!strcmp(answer,"unreserve")) /*could also do strcmp(answer,”reserve”)==0…look up strcmp to see return value
to see why these work*/
    {
      int n=unreserve(all_tables,3);
      printf("Total unreserved: %d\n",n);
  }

  else
  {
   printf("Unknown response.\n");
  }
 }
}
```

Note 1:

Remember in if statements (and stopping condition of loops), we conceptually encode true and false as 0 and 1 (note that any other value other than 0 can count as true). So:

if(0) ←this is false, so if won't execute
if(1) ← this is true, so if will execute…also if(9) would be true, since any other value other than 0 counts as true

NULL also counts as false, so if fp==NULL (see the return value of fopen-NULL is returned if the file doesn't open), then if(fp) is the same as if(0)…false

Note that other languages, like Java for example, may have an actual true/false built in the form of a boolean variable. The concept is the same as mentioned above.

Also remember how relational and logical operators work. Relational operators can build up a true/false statement:
***if('c' == 'b')…in this example, the == relational operator creates an overall true/false statement (false) since c does not equal b.

Logical operators build up an overall true/false statement composed of smaller true/false statements:
***if('c'=='b' && 45<100)…in this example, the && logical operator creates an overall true/false statement from the two smaller true/false statements c'=='b' and 45<100. In this case, the overall true/false statement is False since:

c'=='b' is false
45<100 is true

false && true == false

Note 2:

I am using fgets here…don't worry if you didn't see it in 1310-we will go over in depth in 1320. It is just another way to get user input…note that it automatically inputs \n at the end of input. So if you type in *dog* or read *dog* from a file, the computer will actually save *dog\n*

Same with strtok-we will cover it in depth later on. Here, it is just used to "chop off" the \n at the end of the input read in by fgets.