CS 210 **Sept. 27, 2016**
Day 9

Chapter 6 Pointers and Modular Programming

A pointer is a new variable type which holds the *address* of another variable.

Pointers are declared using the * operator.

Pointers

- pointer (pointer variable)
  - a memory cell that stores the address of a data item
  - syntax: *type *variable*

```
int m = 25;
int *itemp;     /* a pointer to an integer */
```

In this example itemp is a pointer variable.

We can make itemp hold the address of the variable *m* like this:
`itemp = &m;`

Recall that **&** is the address operator that we used in `scanf`

```
#include<stdio.h>
int main()
{
      int m = 25;
      int *mPointer;
      mPointer = &m;
      printf("m = %d\n", m);
      printf("&m = %d\n", &m);
      printf("mPointer = %d\n", mPointer);
}
```

This program prints the following:

```
m = 25
&m = 19922488
mPointer = 19922488
Press any key to continue . . .
```

Notice that the *address* of *m* is stored in *mPointer*.

If you run this program again or on another computer the address of *m* may be at a different location – it depends on where the compiler places it in memory.

We can access the data stored in variable *m* by using its name as in:
printf("%d\n", m);

or,

we can access the data *indirectly* by using the variables address stored in *mPointer*. To do this we again use the * operator as in:

```
printf("%d\n", *mPointer);
```

An indirect address is the address of an address of the data. When we use *mPointer as a variable we effectively go to the mPointer location, get the address, and go to that address to find the data.

This is somewhat confusing. Note that we now have three different meanings for the * operator and which meaning is used by the compiler depends on the context – that is, how it is used.

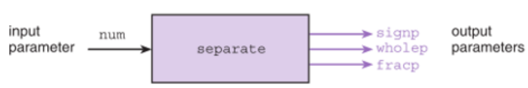
1. If we write `z = x * y;` the * operator means multiplication
2. If we write `int *x;` the * operator is used to declare x to be a pointer to an int variable.
3. If we use `y = *x;` the * operator is used for indirect address. We go to the location *x* and find the information there which is the address of the data which gets moved into y.

Functions with Output Parameters

- We've used the return statement to send back one result value from a function.
- We can also use output parameters to return multiple results from a function.

FIGURE 6.4

Diagram of Function separate with Multiple Results



For example, suppose we want to write a function which will swap two variables which are ints. If we name the function *Swap* and pass it two integers as arguments as in:

```
int x = 5, y = 9;
Swap(x, y);
```

The function will fail because *x* and *y* are passed by value.

```
int main()
   {int x = 5, y = 9;
    Swap(x, y);
    printf("%d %d\n", x, y);
    return 0;
   }
int Swap(int x, int y)
  {int tmp;
   tmp = x;
   x = y;
   y = tmp;
  }
```

Swap	main	memory
	x	5
	y	9
x		~~5~~ 9
y		~~9~~ 5
tmp		5

We can fix this program by passing the address of x and y using pointers.

```
void Swap(int *xPtr, int *yPtr);
int main()
   {int x = 5, y = 9;
    Swap(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
   }
void Swap(int *xPtr, int *yPtr)
  {int tmp;
   tmp = *xPtr;
   *xPtr = *yPtr;
   *yPtr = tmp;
  }
```

Swap	main	memory
*xPtr	x	~~5~~ 9
*yPtr	y	~~9~~ 5
tmp		5

Syntax for writing functions with output parameters

Prototype:

```
int MyFunction(int x, int *y);
//x is passed by value, y is passed by reference
```

Calling statement:

```
int z;
z = MyFunction(x, &y);
//For reference parameters you must pass the address
```

Function definition

```
int MyFunction(int x, int *y)
{
    int z;
    x = 2;
    *y = 3;
      //Reference parameter is used with dereferencing operator
    z = x + *y;
    return z;
}
```

Example

Write a function which will prompt the user for two doubles and return these to the main program.

In the main program:

```
double a, b;
Getab(&a, &b);
```

In the function

<pre>void Getab(double *a, double *b)
{
    double x, y;
    printf("Enter a value for a... ");
    scanf_s("%lf", &x);
    *a = x;
    printf("Enter a value for b... ");
    scanf_s("%lf", &y);
    *b = y;
}</pre>
	<pre>void Getab(double *a, double *b)
	{
	    printf("Enter a value for a... ");
	    scanf_s("%lf", &*a);
	    printf("Enter a value for b... ");
	    scanf_s("%lf", &*b);
	}</pre>

Memory map example

Complete the memory map for the following program.

```
#include<stdio.h>
void Fun1(int a, int *b);
void Fun2(int a, int *b);
int main()
{
	int x = 5, y = 2;
	printf("%d, %d\n", x, y);
	Fun1(x, &y);
	printf("%d, %d\n", x, y);
}
void Fun1(int a, int *b)
{
	int c;
	printf("%d, %d\n", a, *b);
	c = *b;
	Fun2(c, &a);
	printf("%d, %d, %d\n", a, *b, c);
}
void Fun2(int a, int *b)
{
	int c;
	printf("%d, %d\n", a, *b);
	c = *b;
	printf("%d, %d, %d\n", a, *b, c);
}
```

Fun2	Fun1	Main	Data

Printed Output

Memory map example SOLUTION

Complete the memory map for the following program.

```
#include<stdio.h>
void Fun1(int a, int *b);
void Fun2(int a, int *b);
int main()
{
	int x = 5, y = 2;
	printf("%d, %d\n", x, y);
	Fun1(x, &y);
	printf("%d, %d\n", x, y);
}
void Fun1(int a, int *b)
{
	int c;
	printf("%d, %d\n", a, *b);
	c = *b;
	Fun2(c, &a);
	printf("%d, %d, %d\n", a, *b, c);
}
void Fun2(int a, int *b)
{
	int c;
	printf("%d, %d\n", a, *b);
	c = *b;
	printf("%d, %d, %d\n", a, *b, c);
}
```

Fun2	Fun1	Main	Data
		x	5
	*b	y	2
*b	a		5
	c		2
a			2
c			5

Printed Output
5, 2
5, 2
2, 5
2, 5, 5
5, 2, 2
5. 2

Engr 123 **September 26, 2016**
Pointers and output parameters

Polar coordinates are written in the form of $r\angle\theta$ and rectangular coordinates are expressed as the pair *(x, y)*. Write a program that prompts the user to enter *r* and θ (in degrees) and calls a function named *ConvertToXY* which converts the polar coordinates to Cartesian coordinates and returns them to the main program for printing. You will need to pass your function *r* and *theta* by value and *x* and *y* by reference.

To convert polar to Cartesian use:

$x = r\cos(theta)$

$y = r\sin(theta)$

$\pi = 3.141592653589793$

Turn in a printed copy of your source file.

CS 210 **Sept. 29, 2016**
Day 10

Example – Parameter passage
Write a function to simulate throwing two dice. The function will returns a void and the value of the two dice will come back as output parameters. Your main program will print the value of the two dice on the console.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void RollTwoDice(int *d1, int *d2);
int main()
{
    int d1, d2;
    //seed random number genrator with present time
    srand((unsigned)time(NULL));
    RollTwoDice(&d1, &d2);
    printf("%d, %d \n", d1, d2);
    return 0;
}
void RollTwoDice(int *d1,int *d2)
{
    *d1 = rand() % 6 + 1;
    *d2 = rand() % 6 + 1;
}
```

Ch 11 File pointers and text files

Input/Output Files

- text file
  - a named collection of characters saved in secondary storeage
- input (output) stream
  - continuous stream of character codes representing textual input (or output) data

The keyboard and Screen as Text Streams

- stdin
  - system file pointer for keyboard's input stream

- stdout, stderr
  - system file pointers for screen's output stream

TABLE 11.1 Meanings of Common Escape Sequences

Escape Sequence	Meaning
`'\n'`	new line
`'\t'`	tab
`'\f'`	form feed (new page)
`'\r'`	return (go back to column 1 of current output line)
`'\b'`	backspace

TABLE 11.2 Placeholders for printf Format Strings

Placeholder	Used for Output of	Example	Output
`%c`	a single character	`printf("%c%c%c\n", 'a', '\n', 'b');`	a b
`%s`	a string	`printf("%s%s\n", "Hi, how ", "are you?");`	`Hi, how are you?`
`%d`	an integer (in base 10)	`printf("%d\n", 43);`	43
`%o`	an integer (in base 8)	`printf("%o\n", 43);`	53
`%x`	an integer (in base 16)	`printf("%x\n", 43);`	2b
`%f`	a floating-point number	`printf("%f\n", 81.97);`	81.970000
`%e`	a floating-point number in scientific notation	`printf("%e\n", 81.97);`	8.197000e+01
`%E`	a floating-point number in scientific notation	`printf("%E\n", 81.97);`	8.197000E+01
`%%`	a single % sign	`printf("%d%%\n", 10);`	10%

TABLE 11.3 Designating Field Width, Justification, and Precision in Format Strings

Example	Meaning of Highlighted Format String Fragment	Output Produced
`printf("%5d%4d\n", 100, 2);`	Display an integer right-justified in a field of five columns.	`■■100■■■2`
`printf ("%2d with label\n", 5210);`	Display an integer in a field of two columns. Note: Field is too small.	`5210■with■label`
`printf("%-16s%d\n", "Jeri R. Hanly", 28);`	Display a string left-justified in a field of 16 columns.	`Jeri■R.■Hanly■■■28`
`printf("%15f\n", 981.48);`	Display a floating-point number right-justified in a field of 15 columns.	`■■■■■981.480000`
`printf("%10.3f\n", 981.48);`	Display a floating-point number right-justified in a field of ten columns, with three digits to the right of the decimal point.	`■■■981.480`
`printf("%7.1f\n", 981.48);`	Display a floating-point number right-justified in a field of seven columns, with one digit to the right of the decimal point.	`■■981.5`
`printf("%12.3e\n", 981.48);`	Display a floating-point number in scientific notation right-justified in a field of 12 columns, with three digits to the right of the decimal point and a lowercase `e` before the exponent.	`■■■9.815e+02`
`printf("%.5E\n", 0.098148);`	Display a floating-point number in scientific notation, with five digits to the right of the decimal point and an uppercase `E` before the exponent.	`9.81480E-02`

TABLE 11.4 Comparison of I/O with Standard Files and I/O with User-Defined File Pointers

Line	Functions That Access stdin and stdout	Functions That Can Access Any Text File
1	`scanf("%d", &num);`	`fscanf(infilep, "%d", &num);`
2	`printf ("Number = %d\n", num);`	`fprintf(outfilep, "Number = %d\n", num);`
3	`ch = getchar();`	`ch = getc(infilep);`
4	`putchar(ch);`	`putc(ch, outfilep);`

Pointers to Files

- C allows a program to explicitly name a file for input or output.
- Declare file pointers:
  - FILE *inp; /* pointer to input file */
  - FILE *outp; /* pointer to output file */
- Prepare for input or output before permitting access:
  - inp = fopen(“infile.txt”, “r”);
  - outp = fopen(“outfile.txt”, “w”);

Pointers to Files

- fscanf
  - file equivalent of scanf
  - fscanf(inp, “%1f”, &item);
- fprintf
  - file equivalent of printf
  - fprintf(outp, “%.2f\n”, item);
- closing a file when done
  - fclose(inp);
  - fclose(outp);

CS 210 **Fall, 2016**

Reading and writing files with VS 2015

The textbook (pp. 320-322) shows how to read and write files using pointers with `fopen`. If you use `fopen` in Visual Studio you will get an error indicating that `fopen` is not safe and suggesting you use `fopen_s` in its place. You can still use fopen but you need to turn off the errors and warnings first. To do this you need to add the following line at the top of your program which uses pointers to files:

```
#pragma warning(disable:4996)
```

Here is an example of a C program which uses `fopen` to create a file, write some ints into it, read the file, and print its contents to the console.

```
#include<stdio.h>
#pragma warning(disable:4996)
int main()
{
    int i;
    FILE *inp;  //Declare pointers to in
    FILE *outp; //  and out files
    int dataIn, status;
    //Open the file for output
    outp = fopen("MyFile.txt", "w");
    //Write five ints to the file
    for(i=0;i<5;i++)
       fprintf(outp, "%d\n", i);
    fclose(outp);   //close the file
    //reopen for input
    inp = fopen("MyFile.txt", "r");
    status = fscanf(inp, "%d", &dataIn);
    //read the file and print to console
    while(status == 1)
       {
        printf("%d\n", dataIn);
        status = fscanf(inp, "%d", &dataIn);
       }
    fclose(inp);
}
```

We can also use `fopen_s` which is a secure open statement. The syntax is slightly different. The program below is the same as that above but it uses `fopen_s` in place of `fopen`.

```
#include<stdio.h>
int main()
{
    int i, err;
    FILE *inp;  //Declare pointers to in
    FILE *outp; //  and out files
    int dataIn, status;
    //Open the file for output
    err = fopen_s(&outp, "MyData.txt", "w");
    //Write five ints to the file
    for(i=0;i<5;i++)
       fprintf(outp, "%d\n", i);
    fclose(outp);   //close the file
    //reopen for input
    err = fopen_s(&inp, "MyData.txt", "r");
    status = fscanf_s(inp, "%d", &dataIn);
    //read the file and print to console
    while(status == 1)
       {
        printf("%d\n", dataIn);
        status = fscanf_s(inp, "%d", &dataIn);
       }
    fclose(inp);
}
```

Example

Write a function which writes 1000 random integers in the range $0 \leq x \leq 100$ to file called "Numbers.txt". Your function should return a status variable that is 0 only if the file is successfully written. Otherwise it should return a 1. Write a main program which calls your function. The main program should write a message to the screen to indicate whether or not the file write was successful.

```
#include<stdio.h>
#include<stdlib.h>
int WriteNumbers();
int main()
{
    if(WriteNumbers() == 0)
        printf("File written successfully. \n");
    else
        printf("Error writting file.\n");
}
int WriteNumbers()
{
    //Writes 1000 random ints between 0 and 100
    //  to "Numbers.txt"
       int err, i, r;
    FILE *outp; //  and out files
       err = fopen_s(&outp, "Numbers.txt", "w");
    if(err != 0)
        return 1;
    srand(23);
    for(i=0;i<1000;i++)
       {
        r = rand();
        r = r % 101;  //0 to 100
        fprintf(outp, "%d\n", r);
    }
    fclose(outp);
    return 0;

}
```

CS 210 **September 29, 2016**
File Operations

Down load the source file for the example above from the website at:
http://csserver.evansville.edu/~blandfor/CS210/Day10WriteNumbers.docx

Create a project with this source file, compile it, and verify that it runs successfully.

Add a second function to the project which reopens the file "Numbers.txt" for input. Read the numbers in the file and print their average. Your function should add the numbers in the file and count the number of random numbers that were read in. Your function should return the average value of all of the ints in the file as a double. Print this double in the main program.

Turn in a printed copy of your source file.

CS 210 **Oct. 4, 2016**
Day 11

Ch. 7 – Array Pointers

Basic Terminology

- data structure
  - a composite of related data items stored under the same name
- array
  - a collection of data items of the same type

In mathematics we see the notation:

$y = \sum_{i=0}^{5} x_i$ which is taken to mean $y = x_0 + x_1 + x_2 + x_3 + x_4 + x_5$

The variable name *x* is a single name which is used to stand for multiple variable by use of a subscript.

In C we write a subscript using brackets as in x[0] + x[1] + …

The number inside the brackets is the *subscript* and is always in int or an expression that evaluates to an int. This allows us to calculate the name of a variable.

Declaring and Referencing Arrays

- array element
  - a data item that is part of an array
- subscripted variable
  - a variable followed by a subscript in brackets, designating an array element
- array subscript
  - a value or expression enclosed in brackets after the array name, specifying which array element to access

```
double x[8];
```

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Note that
1. The first subscript is always 0 as in x[0].
2. In the array declaration double x[8] give the array type (double) and the size of the array (8).
3. The array size must be a constant.
4. The last subscript is always one less than the array size.

TABLE 7.1 Statements That Manipulate Array x

Statement	Explanation
`printf("%.1f", x[0]);`	Displays the value of `x[0]`, which is `16.0`.
`x[3] = 25.0;`	Stores the value `25.0` in `x[3]`.
`sum = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]`, which is `28.0` in the variable `sum`.
`sum += x[2];`	Adds `x[2]` to `sum`. The new `sum` is `34.0`.
`x[3] += 1.0;`	Adds `1.0` to `x[3]`. The new `x[3]` is `26.0`.
`x[2] = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]` in `x[2]`. The new `x[2]` is `28.0`.

Initializing an array

```
int x[] = {1, 2, 3, 4, 5};  //compiler fills in the right size
char vowels[] = {'a', 'e', 'i', 'o', 'u'};
```

You can also store a string in an array. In fact, a string is just an array of char where the last character is 0 (NULL).

```
char hello[] = "Hello Mom!";
```

In memory this becomes:

[10]	0
[9]	'!'
[8]	'm'
[7]	'o'
[6]	'M'
[5]	' '
[4]	'o'
[3]	'l'
[2]	'l'
[1]	'e'
hello[0]	'H'

We will do strings in Ch 8

Sometimes arrays are initialized using loops:

```
int i;
int x[100];
for(i=0;i<100;i++)
   x[i] = i*i;
```

Example

Suppose we declare an array of doubles as:

```
double x[] = {16.0, 12.0, 6.0, 8.0, 2.5, 12.0, 14.0, -54.5];
```

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

TABLE 7.2 Code Fragment That Manipulates Array x

Statement	Explanation
`i = 5;`	
`printf("%d %.1f\n", 4, x[4]);`	Displays 4 and 2.5 (value of x[4])
`printf("%d %.1f\n", i, x[i]);`	Displays 5 and 12.0 (value of x[5])
`printf("%.1f\n", x[i] + 1);`	Displays 13.0 (value of x[5] plus 1)
`printf("%.1f\n", x[i] + i);`	Displays 17.0 (value of x[5] plus 5)
`printf("%.1f\n", x[i + 1]);`	Displays 14.0 (value of x[6])
`printf("%.1f\n", x[i + i]);`	Invalid. Attempt to display x[10]
`printf("%.1f\n", x[2 * i]);`	Invalid. Attempt to display x[10]
`printf("%.1f\n", x[2 * i - 3]);`	Displays -54.5 (value of x[7])
`printf("%.1f\n", x[(int)x[4]]);`	Displays 6.0 (value of x[2])
`printf("%.1f\n", x[i++]);`	Displays 12.0 (value of x[5]); then assigns 6 to i
`printf("%.1f \n", x[--i]);`	Assigns 5 (6 - 1) to i and then displays 12.0 (value of x[5])
`x[i - 1] = x[i];`	Assigns 12.0 (value of x[5]) to x[4]
`x[i] = x[i + 1];`	Assigns 14.0 (value of x[6]) to x[5]
`x[i] - 1 = x[i];`	Illegal assignment statement

Example

Generate an array of 1000 random ints in the range 0 <= x <= 100. Calculate the average, mean, and standard deviation of the data in the array.

The standard deviation is given by:

$$\textit{standard deviation} = \sqrt{\frac{\sum_{\texttt{i=0}}^{\texttt{MAX\_ITEM-1}} \texttt{x[i]}^2}{\texttt{MAX\_ITEM}} - \textit{mean}^2}$$

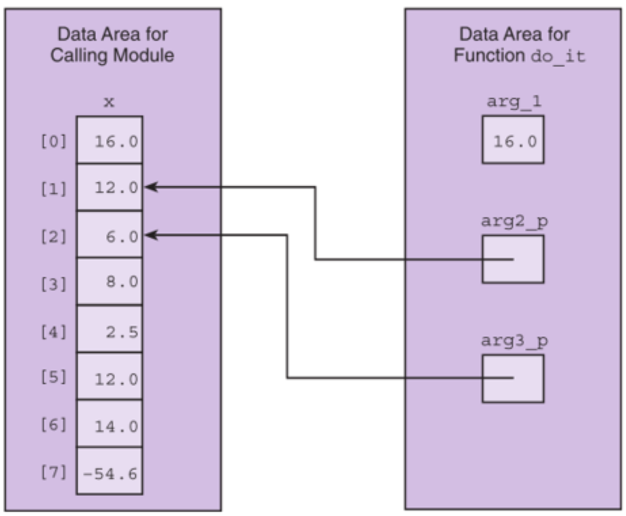
```
#define SIZE 1000
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
{
    int i;
    int data[SIZE];
    double mean, stdev, sum, sumSqr;
    srand(23);
    for(i=0;i<SIZE;i++)
       data[i] = rand() % 101;
    sum = 0;sumSqr = 0;
    for(i=0;i < SIZE;i++)
    {
        sum += data[i];
        sumSqr += data[i]*data[i];
    }
    mean = sum/SIZE;
    stdev = sqrt(sumSqr/SIZE - mean*mean);
    printf("The mean is %6.3f\n", mean);
    printf("The standard deviation is %6.3f\n", stdev);
}
```

```
The mean is 50.218
The standard deviation is 29.421
Press any key to continue . . .
```

Using Array Elements as Function Arguments

scanf("%lf", &x[i]);

do_it(x[0], &x[1], &x[2]);



Array Arguments

- We can write functions that have arrays as arguments.
- Such functions can manipulate some, or all, of the elements corresponding to an actual array argument.

Array names are actually pointers to where the first element of an array is stored.in memory. If we index an array the index is added to the pointer to get the data.

This also means that, by default, all arrays are passed by reference and changing an array in a function that has been passed as a parameter also changes the array in the calling program.

Example

Write a function called FillArray which three parameters: the first is the array, the second is the number of items in the array, and the third is the value used to fill the array.

```
#include<stdio.h>
void FillArray(int data[], int n, int fill);
int main()
{
    int myData[20];
    int i;
    FillArray(myData, 5, 25);
    for(i=0;i<5;i++)
       printf("%d\n", myData[i]);
}
void FillArray(int data[],int n,int fill)
{
    int i;
    for(i=0;i<n;i++)
        data[i] = fill; //data and myData are the same array
}
/* printed output
25
25
25
25
25
Press any key to continue . . .
*/
```

Note that the array in FillArray is called `int data[]` but this array name is just a pointer and we could write this parameter as a pointer instead of as an array. The following also works.

```
void FillArray(int *d, int n,int fill)
{
    int i;
    for(i=0;i<n;i++)
        d[i] = fill;
}
```

In some cases, such as when multiple programmers are working on various functions that receive an array parameter you may want to pass an array but disallow the array to be changed. You can do this by declaring the array in the parameter list to be constant like this:

```
void FillArray(const int data[],int n,int fill)
{
    int i;
    for(i=0;i<n;i++)
        data[i] = fill; //this will create an error
}
```

The const declaration will prevent the array from being changed.

Example

Write a program which creates a list of 1000 simulated throws of a dice. Call a function which will count the number of ones, twos, … sixes. The function should have two parameters: The first will contain the simulation of the 1000 throws of the dice and the second will have six entries that contain the number of ones, twos, etc. The first is an input array and should be declared const, the second is an output array. The output array should be printed in the main program.

```
#define SIZE 1000
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void CountDice(const int dice[], int numCnt[]);
int main()
{
    int dice[SIZE];
    int numCnt[7];
    int i;
    srand((unsigned)time(NULL));
    for(i=0;i < SIZE;i++)
       dice[i] = rand() % 6 + 1;
    CountDice(dice, numCnt);
    for(i=1;i<7;i++)
      printf("Number of %ds = %d\n", i, numCnt[i]);
    return 0;
}
void CountDice(const int dice[],int numCnt[])
{
    int i;
    for(i=0;i<7;i++)
        numCnt[i] = 0;
    for(i=0;i < SIZE;i++)
    {
        switch(dice[i])
        {
            case 1:
              numCnt[1]++;
              break;
            case 2:
              numCnt[2]++;
              break;
            case 3:
              numCnt[3]++;
              break;
            case 4:
              numCnt[4]++;
              break;
            case 5:
              numCnt[5]++;
              break;
            case 6:
              numCnt[6]++;
              break;
            default:
              break;
        }
    }
}
```

Note that in general, for case I we increment numCnt[i]. This allows us to write it like this:

```
void CountDice(const int dice[],int numCnt[])
{
    int i;
    for(i=0;i<7;i++)
        numCnt[i] = 0;
    for(i=0;i<SIZE;i++)
        numCnt[dice[i]]++;

}
```

The statement `numCnt[dice[i]]++` is calculating the name of a variable.

You may be tempted to write a function which returns an array. In C functions may not return an array so they must be used as output parameters.

Example
Write a function which will accept and array as a parameter and return the index of the first negative entry in the array. If no negative entries are found, your function should return -1.

```
#include<stdio.h>
int FirstNegative(int d[], int n);
int main()
{
    int d[] = {1, 2, 3, 4, -1, -2, -3, -4};
    int indxNeg;
    indxNeg = FirstNegative(d, 8);
    printf("Index of first negative is %d\n", indxNeg);
}
int FirstNegative(int d[], int n)
{
    int indx = 0;
    while(indx < n && d[indx] >= 0)
        indx++;
    if(indx == n)
       indx = -1;
    return indx;
}
```

CS 210 **October 4, 2016**
Arrays

Write a method called Rotate which shifts all the values in an array argument to the left 1 place with the value at index 0 becoming the last value in the array. For example, if the array is defined by `int [] data = {4, 5, 6, 7},` your method should return the array with the data `{5, 6, 7, 4}.`

Use the following sequence in the main program to call your function.

```
#include<stdio.h>
void Rotate(int d[], int n);
int main()
{
    int d[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i;
    for(i=0;i<8;i++)
       printf("%d ", d[i]);
    printf("\n");
    Rotate(d, 8);
    for(i=0;i<8;i++)
       printf("%d ", d[i]);
    printf("\n");
}
```

Turn in a printed copy of your source file.

Press any key to continue . . .

CS 210 **Oct. 6, 2016**
Day 12

More of Ch. 7 – Array Pointers

Stacks

- A stack is a data structure in which only the top element can be accessed.
- pop
  - remove the top element of a stack
- push
  - insert a new element at the top of the stack

FIGURE 7.13 Functions push and pop

```c
void
push(char stack[],    /* input/output - the stack */
     char item,       /* input - data being pushed onto the stack */
     int  *top,       /* input/output - pointer to top of stack */
     int  max_size)   /* input - maximum size of stack */
{
     if (*top < max_size-1) {
         ++(*top);
         stack[*top] = item;
     }
}

char
pop(char stack[],    /* input/output - the stack */
    int *top)        /* input/output - pointer to top of stack */
{
     char item;      /* value popped off the stack */

     if (*top >= 0) {
          item = stack[*top];
          --(*top);
     } else {
          item = STACK_EMPTY;
     }

     return (item);
}
```

Stacks are used almost universally to save the return address when a function is called. If you ever used an HP calculator with Reverse Polish Notation, you have a better idea of what a stack can do in terms of arithmetic.

Selection Sort

1. for each value of fill from 0 to n-2
   2. Find index_of_min, the index of the smallest element in the unsorted subarray list[fill] through list[n-1]
   3. if fill is not the position of the smallest element (index_of_min)
      4. Exchange the smallest element with the one at position fill.

Selection Sort

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void SelectionSort(int d[], int n);
void Swap(int *x, int *y);
void FindMin(int d[], int n, int start, int *minIndx);
int main()
{
   int i;
   int data[100];
   srand(23);
   for(i=0;i<100;i++)
      data[i] = rand() % 101;
   for(i=0;i<100;i++)
      printf("%d, ", data[i]);
   printf("\n\n");
   SelectionSort(data, 100);
   for(i=0;i<100;i++)
      printf("%d, ", data[i]);
   printf("\n");
}
void SelectionSort(int d[], int n)
    {int i, j, swpCnt, minIndx;
     swpCnt = 0;
     for(i=0;i<n-1;i++)
        {for(j=i;j<n;j++)
           {FindMin(d, n, i, &minIndx);
            Swap(&d[i], &d[minIndx]);
            swpCnt++;
            }
        }
     printf("Selection sort done with %d swaps.\n", swpCnt);
    }
void FindMin(int d[], int n, int start, int *minIndx)
   {int i;
    *minIndx = start;
    for(i=start;i<n;i++)
       {if(d[i] < d[*minIndx])
           *minIndx = i;
       }
   }
void Swap(int *a, int *b)
   {int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
   }
```

Bubble Sort

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void BubbleSort(int d[], int n);
void Swap(int *x, int *y);
int main()
{
   int i;
   int data[100];
   srand(23);
   for(i=0;i < 100;i++)
      data[i] = rand() % 101;
   for(i=0;i < 100;i++)
      printf("%d, ", data[i]);
   printf("\n\n");
   BubbleSort(data, 100);
   for(i=0;i < 100;i++)
      printf("%d, ", data[i]);
   printf("\n");
}
void BubbleSort(int d[], int n)
   {int i, swpCnt, fDone;
    fDone = 0;
    swpCnt = 0;
    while(!fDone)
        {fDone = 1;
         for(i=0;i<n-1;i++)
             {if(d[i] > d[i+1])
                 {Swap(&d[i], &d[i+1]);
                  fDone = 0;
                  swpCnt++;
                 }
             }
         }
     printf("Bubble sort done with %d swaps.\n", swpCnt);
     }
void Swap(int *a, int *b)
   {int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
   }
```

Enumerated Types

- enumerated type
  - a data type whose list of values is specified by the programmer in a type declaration
- enumeration constant
  - an identifier that is one of the values of an enumerated type

```
typedef enum
        {Monday, Tuesday, Wednesday, Thursday,
         Friday, Saturday, Sunday}
day_t;
```

```
#include <stdio.h>
enum day { sunday, monday, tuesday, wednesday, thursday, friday, saturday };
int main()
{
    enum day today = wednesday;
    printf("Day %d\n",today+1);
    return 0;
}
//Day 4
//Press any key to continue . . .
```

Alternatively, you can write it like this:

```
#include <stdio.h>

typedef enum
{sunday, monday, tuesday, wednesday, thursday, friday, saturday
}day;

int main()
{
    day today = wednesday;
    printf("Day %d\n",today+1);
    return 0;
}
//Day 4
//Press any key to continue . . .
```

This is a way to create a Boolean variable type:

```
#include<stdio.h>
enum Boolean {false, true};  //false is 0 and true is 1
int main()
{
    enum Boolean flag;
    flag = false;
    printf("%d\n",  flag);
}
```

CS 210 **October 6, 2016**

Sorting characters

The code segment below creates an array named `data` which has 100 random lower case characters. Use this segment in your own main program to create an array of 100 random lower case characters. Print your characters to the screen with one space between each character.

```
int i;
char data[100];
srand(23);
for(i=0;i < 100;i++)
   data[i] = (char)((rand() % 26) + 'a');
for(i=0;i < 100;i++)
   printf("%c, ", data[i]);
printf("\n\n");
```

Write a sorting function (either a bubble sort or a section sort) to sort your data. Call the sorting program from your main program and again print the characters with one space between each to show that they were successfully sorted.

Turn in a printed copy of your source file.