Engr 101 **Sept. 18, 2019**
Day 11

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);
  printf("%d, %d \n", (int)&d1, (int)&d2);

 return 0;
}
void RollTwoDice(int *pd1,int *pd2)
{
 *pd1 = rand() % 6 + 1;
 *pd2 = rand() % 6 + 1;
  printf("%d, %d \n", (int)pd1, (int)pd2);

}
```

Go over this example with the debugger.

```
#include<stdio.h>
int main()
{
    int a = 15;
    int *aPtr = &a;
    printf("%d\n", a);
    printf("%d\n", (int)&a);
    printf("%d\n", (int)aPtr);
    printf("%d\n", *aPtr);
    printf("%d\n",(int)(&*aPtr));
    printf("%d\n", *(aPtr));
}/*
15
7337904
7337904
15
7337904
15
Press any ke
```

```c
#include<stdio.h>
void Method1(int x, int y, int z);
int Method2(int a, int z);

void main()
      {int x, y, z;
       x = 2;
       y = 4;
       z = 12;
       printf("%d, %d, %d\n", x, y, z);
       Method1(x, y, z);
       printf("%d, %d, %d\n", x, y, z);
      }

    void Method1(int x, int y, int z)
      {int a = 7;
       int b;
       b = y + Method2(a, z);
       printf("%d, %d, %d\n", a, b, z);
      }

    int Method2(int a, int z)
      {int t;
       t = a + z;
       z = 24;
       printf("%d, %d, %d\n", t, a, z);
       return t;
      }
//2, 4, 12
//19, 7, 24
//7, 23, 12
//2, 4, 12
```

Addresses			Data
Method2	Method1	Main	

Printed Results

_____
_____
_____
_____
_____
_____
_____
_____
_____

Pass by Reference

```c
#include<stdio.h>
void Method1(int k, int *i, int *j);
void main()
 {int i = 4, j = 7, k;
  printf("In the main program:\n");
  printf("i =  %d j = %d\n", i, j);
  Method1(i, &j, &k);
  printf("In the main program:\n");
  printf("i = %d j = %d, k = %d\n", i, j, k);              }

 void Method1(int k, int *i, int *j)
 {
 printf("In Method1 program:\n");
 printf("i =  %d k = %d\n", *i, k);
 *i = 21;
 *j = 22;
  k = 23;
 printf("In Method1 program:\n");
 printf("i = %d j = %d, k = %d\n", *i, *j, k);
 }
// In the main program:
//i =  4 j = 7
//In Method1 program:
//i =  7 k = 4
//In Method1 program:
//i = 21 j = 22, k = 23
//In the main program:
//i = 4 j = 21, k = 22
```

Memory Address		
Method 1	**Main**	**Data**

Format string specifications[edit]

The formatting placeholders in `scanf` are more or less the same as that in `printf`, its reverse function.

There are rarely constants (i.e. characters that are not formatting placeholders) in a format string, mainly because a program is usually not designed to read known data. The exception is one or more whitespace characters, which discards all whitespace characters in the input.

Some of the most commonly used placeholders follow:

- `%d` : Scan an integer as a signed decimal number.
- `%i` : Scan an integer as a signed number. Similar to `%d`, but interprets the number as hexadecimal when preceded by `0x` and octal when preceded by `0`. For example, the string `031` would be read as 31 using `%d`, and 25 using `%i`. The flag `h` in `%hi` indicates conversion to a `short` and `hh` conversion to a `char`.
- `%u` : Scan for decimal `unsigned int` (Note that in the C99 standard the input value minus sign is optional, so if a minus sign is read, no errors will arise and the result will be the two's complement of a negative number, likely a very large value. See `strtoul()`.[not in citation given]) Correspondingly, `%hu` scans for an `unsigned short` and `%hhu` for an `unsigned char`.
- `%f` : Scan a floating-point number in normal (fixed-point) notation.
- `%g`, `%G` : Scan a floating-point number in either normal or exponential notation. `%g` uses lower-case letters and `%G` uses upper-case.
- `%x`, `%X` : Scan an integer as an unsigned hexadecimal number.
- `%o` : Scan an integer as an octal number.
- `%s` : Scan a character string. The scan terminates at whitespace. A null character is stored at the end of the string, which means that the buffer supplied must be at least one character longer than the specified input length.
- `%c` : Scan a character (char). No null character is added.
- whitespace: Any whitespace characters trigger a scan for zero or more whitespace characters. The number and type of whitespace characters do not need to match in either direction.
- `%lf` : Scan as a double floating-point number.
- `%Lf` : Scan as a long double floating-point number.

The above can be used in compound with numeric modifiers and the `l`, `L` modifiers which stand for "long" in between the percent symbol and the letter. There can also be numeric values between the percent symbol and the letters, preceding the `long` modifiers if any, that specifies the number of characters to be scanned. An optional asterisk (`*`) right after the percent symbol denotes that the datum read by this format specifier is not to be stored in a variable. No argument behind the format string should be included for this dropped variable.

The `ff` modifier in printf is not present in scanf, causing differences between modes of input and output. The `ll` and `hh` modifiers are not present in the C90 standard, but are present in the C99 standard.[2]

An example of a format string is

```
"%7d%s %c%lf"
```

The above format string scans the first seven characters as a decimal integer, then reads the remaining as a string until a space, new line or tab is found, then scans the first non-whitespace character following and a double-precision floating-point number afterwards.

More on Formatted I/O

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`