Pointers

a ***pointer*** is a variable that holds a memory address.

There are two main operators associated with pointers, the address operator and the dereference operator:

& - the address operator retrieves the memory address of the operand.

* - the derefence operator retrieves the value stored at the address held in the pointer.

Recall that information information in a computer system is *bits in context*, so all data types have different sizes and are stored in varying formats in memory. If a pointer variable were only associated with a memory address, then it would not give us enough context to fully describe a data type. Therefore we must declare a pointer variable using the specific C data type that is stored at the memory address.To designate a pointer variable we use the * modifier on the variable name.

```
char c; // declares a variable of character data type
char *cp; // declares a pointer that holds the memory address of a character value

int i; // declares a variable of integer data type
int *ip; // declares a pointer that holds the memory address of an integer value
```

We can assign a value to a pointer by providing it with a memory address. The easiest way to do this is by using the **address operator**, &, on an existing variable.

```
char c='A'; // declares a variable of character data type, with 'A' as value
char *cp=&c; // declares a pointer that holds the memory address of the value stored in the
variable c

int i=45; // declares a variable of integer data type, with 45 as the value
int *ip=&i; // declares a pointer that holds the memory address of the value stored in the
integer variable i
```

We can access the value at the memory address stored in a pointer variable by using the **dereference operator** *

```
printf("%c %c\n", c, *cp);     // prints "A A"
printf("%d %d\n", i, *ip);     // prints "45 45"
```

Q:Since pointers hold memory addresses (bits) of a certain data type (context), what happens if we try to treat an integer pointer as if it were character pointer?

But what does it mean?

Among other things, pointer variables give us the ability to:

1. Return more than one value from a function call:

```
int divide(int dividend, int divisor) {
```

```
    return divident/divisor;
}
```

What if we want to return both the quotient and the remainder? We can pass pointers to the function.

```
void divide(int dividend, int divisor, int *quotient, int *remainder) {
    *quotient=dividend/divisor;    // stores the quotient at the memory address contained in
the quotient variable
    *remainder=dividend%divisor;    // stores the remainder at the memory address containted
in the remainder variable
}
```

Example:Swapping Values

```
                                  variable  memory    value
                                  name      address

    void swap_em( int *, int *);

    int main() {
         int x = 8;               x         7040      8
         int y = 18;              y         7042      18

         /* pass addresses, swap values */
         swap_em( &x, &y );
         return 0;
    }

    void
    swap_em(int *a, int *b) {     a         8000      7040
                                  b         8002      7042

        int temp;                 temp      9000      0

         temp = *a;               temp      9000      8
         *a = *b;                 *a        8000      18
         *b = temp;               *b        8002      8
    }
```

2. Create data structures like linked lists.

```
// define a data structure for a doubly linked list of integers
typedef struct i_ll {
   int value;
   struct i_ll *previous;    // pointer to previous element in linked list
   struct i_ll *next;    // pointer to next element in linked list
} int_linked_list_t;
```

3. Efficient manipulation of ordered collections.

   Example: Insertion sort of the following integers using an array vs. parallel array vs. pointers

10,9,8,7,6,5,4,3,2,1

4. Allows us to store additional memory that we request via dynamic memory allocation

Pointer Arithmetic

The arithmetic operators defined on pointers are ++, --, +, -, +=, and -=. When using arithmetic operators on a pointer, the address value is incremented or decremented based on the size of the data type. For example an integer pointer, *ip, containing the memory address of 2000 becomes 2004 when incremented. This is because the size of an integer 4 bytes. If the pointer where a character pointer, it would be incremented to 2001.

We can derefence values at memory addresses without modifying the address stored in the pointer variable by using arithmetic with the derefence operator.

```
  int array[100];
  int *ip=array;
  int i;

  for(i=0;i<100;i++) {
    printf("%d\n", *(ip+i));
  }
```

the NULL pointer

NULL is a constant that is defined in stdlib.h and is a valid value for a pointer variable. If a pointer is defined but holds the NULL value, then the pointer does not point to anything:

```
int *p = NULL, *q;
if (p == q) printf("p and q are the same!\n");
```

This code would print *nothing* because q is undefined. An *undefined pointer* is a pointer whose value has either not been initialized or one that no longer points to a valid place in memory. We often use the NULL pointer to initialize a pointer or set it to NULL once the memory it points to is no longer available. We can also use the NULL pointer to denote either an empty list or the end of the list. The following code loops through a linked-list structure:

```
struct node {
  char letter;
  struct node *next;
};

void loop_it(struct node *ltr) {

  while(ltr != NULL) {
    printf("%c",ltr->letter);
    ltr=ltr->next;
  }
  printf("\n");
}
```