Pointers

- Pointers and Addresses
- Pointers
- Using Pointers in Call by Reference
- Swap - A Pointer Example
- Pointers and Arrays
- Operator Precedence Example
Variables

- Variable names correspond to memory locations in memory. Every variable has a type, a name and a value.

```c
int i;           i
i = 4;

32212242

(the address of i )  &i

4
```
int main ()
{
    int i;
    i = 4;
    printf("i = %d, address of i = %u\n", i, &i);
    return 0;
}

$./ptr1
i = 4, address of i = 3220392980
Pointers

- What is a pointer?
  - a variable that contains a memory address as its value.
  - Pointers contain the address of a variable that has a specific value (an indirect reference).

- Pointers in C are **typed**.
  - a pointer to a variable of type int
  - a pointer to a variable of type char
  - a pointer to a defined type or an object.
Fig. 7.1 Directly and Indirectly Referencing a Variable

- `count` directly references a variable that contains the value 7.
- Pointer `countPtr` indirectly references a variable that contains the value 7.
/* Welcome to the world of Pointers!
Pointers are a powerful tool */

int main ()
{
    int i;
    int *ptr; /* pointer declaration */

    i = 4;
    ptr = &i;
    printf(" i = %d
 address of i = %u
 address of pointer = %u
",
           i, ptr, &ptr);
    return 0;
}

./ptr2
i = 4
address of i = 3219352564
address of pointer = 3219352560
/* Do you think in Hex ?*/

```c
int main ()
{
    int i;
    int *ptr;

    i = 4;
    ptr = &i;
    printf(" i = %d
    \n address of i = %p
    \n address of pointer = %p
", 
    i, ptr, &ptr);
    return 0;
}
```

./ptr3
```bash
i = 4
address of i = 0xbfe07244
address of pointer = 0xbfe07240
```
/* Never trust a Compiler. */
int j, i; /* think globally! */
int *ptr1, *ptr2;
void printit ()
{
    printf(" i = %2d, ptr1 = %p\n", i, ptr1);
    printf(" j = %2d, ptr2 = %p\n", j, ptr2);
}
int main ()
{
    i = 4; j = 8;
    ptr1 = &i;
    ptr2 = &j;
    printit ();
    *ptr2 = *ptr2 + 1;
    ptr1 = ptr1 - 2; /* You cannot know this */
    printit ();
    i = 6;
    *ptr1 = *ptr1 + 10;
    printit ();
    return 0;
}

The unary (or indirection) operator returns the value of the object to which its operand points.
All arguments in C are passed by value!!

Call by reference is done with pointer arguments.
- Pass address of argument using & (address operator).
- Allows you to change the value of the variable in the caller.
- Arrays are not passed with & because the array name is already a pointer.

* indirection operator
- Used as alias/nickname for variable inside of function
  ```c
  void double(int *number)
  {
      *number = 2 * (*number);
  }
  - *number used as nickname for the variable passed.
/* Fig. 7.7: fig07_07.c  
 Cube a variable using call-by-reference with a pointer argument */

#include <stdio.h>

#include <stdio.h>

void cubeByReference( int *nPtr ); /* prototype */

int main( void )
{
    int number = 5; /* initialize number */
    printf( "The original value of number is %d", number );
    /* pass address of number to cubeByReference */
    cubeByReference( &number );
    printf( "\nThe new value of number is %d\n", number );
    return 0; /* indicates successful termination */
}

/* end main */

void cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
}

/* end function cubeByReference */

The original value of number is 5
The new value of number is 125

Using Pointers in Call by Reference

Function prototype takes a pointer argument

Function cubeByReference is passed an address, which can be the value of a pointer variable.

In this program, *nPtr is number, so this statement modifies the value of number itself.
/* A simple memory swap using pointers */

void swap (int *i, int *j)
{
    int temp;

    temp = *i;
    *i   = *j;
    *j   = temp;
}
int main ( )
{
    int i;
    int mem1;
    int mem2;
    int ray1[4];
    mem1 = 12;
    mem2 = 81;
    swap (&mem1, &mem2); /* swap two integers */
    printf("mem1:%4d mem2:%4d\n", mem1, mem2);
    for (i = 0; i < 4; i++)
    {
        ray1[i] = 10*i;
        printf("ray1[%d] =%4d ", i, ray1[i]);
    }
    printf("\n");
}
swap (&mem1, &ray1[3]);
swap (&mem2, &ray1[2]);
printf("mem1:%4d mem2:%4d\n", mem1, mem2);

for (i = 0; i < 4; i++)
    printf("ray1[%d] =%4d ", i, ray1[i]);

printf("\n");
return 0;

mem1:  30 mem2:  20
```c
int main ()
{
    int i, r[6] = {1,1,1};
    int *ptr;
    ptr = &r;
    *ptr = 83;
    *(ptr + 2) = 33;
    for (i=0; i < 6; i++)
        printf (" r[%d] = %d\n", i, r[i]);
}```
```c
void r[6] = {83, 1, 33, 0, 0, 0};

int main()
{
    int *ptr = &r[0];
    printf("r[0] = %d, r[1] = %d, r[2] = %d, r[3] = %d, r[4] = %d, r[5] = %d\n", r[0], r[1], r[2], r[3], r[4], r[5]);
    r[4] = *ptr;
    ptr++;
    *ptr = 6;
    *(ptr + 2) = 7;
    for (int i = 0; i < 6; i++)
    {
        printf(" r[%d] = %d\n", i, r[i]);
    }
    return 0;
}
```
```c
int main()
{
    int r[6] = {83, 1, 33, 0, 83, 0};
    int *ptr = r;

    r[4] = *ptr;
    ptr++;
    *ptr = 6;
    *(ptr + 2) = 7;
    for (int i = 0; i < 6; i++)
    {
        printf("r[%d] = %d\n", i, r[i]);
    }
    return 0;
}
```
Pointers and Arrays

```c
int r[6] = {83, 6, 33, 7, 83, 0};

int main()
{
    int ptr;
    for (ptr = 0; ptr < 6; ptr++)
    {
        printf("r[%d] = %d\n", ptr, r[ptr]);
    }
    return 0;
}
```
/* An example of operator precedence trouble */

int main ()
{
    float x, y, z;
    float *ptr1, *ptr2, *ptr3;

    x = 2.0; y = 8.0; z = 4.0;
    ptr1 = &x;
    ptr2 = &y;
    ptr3 = &z;
    printf(" %u %u %u\n", ptr1, ptr2, ptr3);

    *ptr3++;
    printf(" %f %f %f\n", x, y, z);
    printf(" %u %u %u\n", ptr1, ptr2, ptr3);
    printf(" %f %f %f\n", *ptr1, *ptr2, *ptr3);
}

$ ./prec
3220728372 3220728368 3220728364
2.000000 8.000000 4.000000
3220728372 3220728368 3220728368
2.000000 8.000000 8.000000
```c
(*ptr1)++;  
printf(" %f %f %f\n", *ptr1, *ptr2, *ptr3);

--*ptr2;  
printf(" %f %f %f\n", *ptr1, *ptr2, *ptr3);
printf(" %f %f %f\n", x, y, z);
return 0;
}
```

```
3.000000 8.000000 8.000000
3.000000 7.000000 7.000000
3.000000 7.000000 4.000000
```
Review of Pointer Basics

- This section demonstrated the relationship between pointers and addresses and introduced the respective operators & and *
- Showed the use of pointers in simple examples.
- Introduced call by reference with pointers.
- Detailed the relationship between pointers and arrays.