Pointers

- Every storage location in the memory of a computer has a number which identifies it uniquely. This identifying number is called its **address**.

- Every variable in C is assigned a memory location and thus an address. Sometimes it is desirable to refer to a variable's address.

- A **pointer** is a variable that can hold an address as its value.

- **The Address Operator & :**
  One way to access the address of a variable is to place the **address operator**, an ampersand (**&**), in front of the variable's name.

declaration	address
int sum;	&sum
double sales;	&sales
char initials;	&initials

- **Declaring a Pointer Variable:**

```
int * p;
char * cp;
double * dp;
```

- **Assigning a Value to a Pointer Variable:**

```
p =  &sum;
cp =  &initials;
cp =  &sales;
```

- **The Dereferencing Operator * :**
  - The notation ***p** means the object that p points to (or the contents of the memory location pointed to by p).
  - This operator allows us to access memory locations through **indirect addressing**.

- **Using Pointers with * and &:**

```
int num1 = 5, num2 = 3;
int *p;

p = &num1;
printf("num1 holds %d and num2 holds %d\n",num1,num2);
printf("and the location p points to holds %d\n",*p);
p = &num2;
printf("num1 holds %d and num2 holds %d\n",num1,num2);
printf("and the location p points to holds %d\n",*p);
```

- Output:

```
num1 holds 5 and num2 holds 3
and the location p points to holds 5
num1 holds 5 and num2 holds 3
and the location p points to holds 3
```

- **Using Pointers to Change a Storage Location's Value:**

```
int num = 5;
int *p = &num;        // note initialization of p

*p = 10;
*p = *p + 1;
(*p)++;               //not *p++ which first increments p
```

Using Parameters Which are Pointers

- **Program:**

```
/* program to try to add one to function parameter */
#include < stdio.h>
void trytoadd1(int);

void main()
{
   int k = 5;

   printf("in main - before call: %d",k);
   trytoadd1(k);
   printf("in main - after call: %d",k);
}

/* function that tries to add one to its parameter */
void trytoadd1(int x)
{
   printf("in function - before adding: %d",x);
   x++;
   printf("in function - after adding: %d",x);
   return;
}
```

- Output:

```
in main - before call: 5
in function - before adding: 5
in function - after adding: 6
in main - after call: 5
```

WHAT WENT WRONG???

- **Correct Version of Program:**

```
/* program to add one to function parameter */
#include < stdio.h>
void add1(int *);

void main()
{
   int k = 5;

   printf("in main - before call: %d",k);
   add1(&k);
   printf("in main - after call: %d",k);
}

/* function that adds one to its parameter */
void add1(int *x)
{
   printf("in function - before adding: %d",*x);
   (*x)++;
   printf("in function - after adding: %d",*x);
   return;
}
```

- Output:

```
in main - before call: 5
in function - before adding: 5
in function - after adding: 6
in main - after call: 6
```

Returning More than One Value from a Function

- **Program to find both max and min of two values:**

```
/* program to find max and min of two values */
#include < stdio.h>
void findmaxmin(int,int,int * ,int * );

void main()
{
   int a= 1,b= 2,larger,smaller;

   findmaxmin(a,b,&larger,&smaller);
   printf("the larger is %d and the smaller is%d\n",
      larger,smaller);
}
/* Function findmaxmin
 * Input:
 *    x & y - two integers to be compared
 *    max & min - locations to put larger and smaller values
 * Process:
 *    store larger of x & y into *max and smaller into *min
 * Output:
 *    *max and *min contain max and min values
 */
void findmaxmin(int x, int y, int *max, int *min)
{
   if (x >  y) {
      *max =  x;
      *min =  y;
   }
   else {
      *max =  y;
      *min =  x;
   }
   return;
}
```

Function to Interchange the Value of its Parameters

- **Program to swap contents of two variables:**

```
#include < stdio.h>
void swap(int * ,int * );

void main()
{
   int x =  5, y =  3;

   printf("in main - before call: x =  %d y =  %d\n",x,y);
   swap(&x,&y);
   printf("in main - after call: x =  %d y =  %d\n",x,y);
}

/*  Function swap:
 *  Input:
 *     a & b - pointers to two integers
 *  Process:
 *     interchange the values pointed to by a and b
 *  Output:
 *     values have been interchanged
 */
void swap(int * a, int * b)
{
   int temp;

   temp =  * a;
   * a =  * b;
   * b =  temp;
   return;
}
```

- Output:

```
in main - before call: x =  5 y =  3
in main - after call: x =  3 y =  5
```

Pointers and Arrays

- **In C, a reference to the name of an array without a subscript, means the address of the array.**

- When an array is sent as a parameter to a function, only the address is sent.

- Because the array name is itself an address, we do not preface it by an & when passing it to a function.

```
int readdata(int numbers[])        // function header

num =  readdata(mark);             // function call
```

- **Using * in the Function Header for an Array Parameter:**
  `int readdata(int * numbers)        // equivalent header`

- Note:
  - a pointer variable holds a value that can change
  - an array name is a constant.

```
int a,b;
int * p;
int num[100];

p =  &a;
p =  &b;
p =  num;
p =  &num[0];
p =  &num[1];
p =  &num[99];

num =  &a;       // illegal
num =  p;        // illegal
```

- **Using & to Send an Address to an Array Parameter:**

```
num =  readdata(&mark[0]);        // same as readdata(mark)
```

- **Example:**

```
/*  ... */
int sumarray(int numbers[], int n)
{
   int count,sum= 0;

   for (count =  0; count <  n; count+ + )
      sum + =  numbers[count];
   return(sum);
}
```

- To sum the elements from 0 to num-1:

```
sum =  sumarray(mark,num);
```

- To sum the elements from 5 to 11:

```
sum =  sumarray(&mark[5],7);
```

Pointer Arithmetic

- **Pointer Arithmetic** can be used to address the elements of an array without subscripts.

- We use direct address manipulation to move from one array element to the next.

- The techniques is based on the **displacement** or **offset** of an element, which measures how far an element is from the beginning of the array.

- Example:

```
int num[5];
```

subscript	0	1	2	3	4
num	10	20	45	50	68
offset	+ 0	+ 1	+ 2	+ 3	+ 4

- num[i] is equivalent to *(num + i)

```
num[1] = 54;
*(num+ 1) = 54;

num[2] = 26;
*(num+ 2) = 26;
```

- int num[] is equivalent to int *num

- Example:

```
int i,sum= 0;
int num[100];
int * p =  num;

for (i =  0; i <  100; i+ + )
   sum + =  num[i];

for (i =  0; i <  100; i+ + )      // equivalent loop
   sum + =  * (num +  i);

for (i =  0; i <  100; i+ + ) {    // equivalent loop
   sum + =  * p;
   p+ + ;
}
```

- We can use pointer notation to send a function the address of a location offset within an array.

```
sum =  sumarray(&mark[5],7);

sum =  sumarray(mark+ 5,7);  // equivalent call
```

Modifying Some Earlier Functions

- **The Function readdata:**

```
/* ... */
int readdata(int numbers[])
{
    int count,n;

    printf("Enter the number of marks: ");
    scanf("%d",&n);

    // enter marks
    for (count = 0; count < n; count++ ) {
        printf("Enter a mark: ");
        scanf("%d",&numbers[count]);
    }
    return (n);
}
```

- **Revised Function readdata:**

```
/* ... */
void readdata(int numbers[], int * n)
{
    int count;

    printf("Enter the number of marks: ");
    scanf("%d",n);

    // enter marks
    for (count = 0; count < * n; count++ ) {
        printf("Enter a mark: ");
        scanf("%d",&numbers[count]);
    }
    return;
}
```

- **Modifying findmax to a New Function findlimits:**

```
/* Function findlimits */
...
void findlimits(int numbers[], int n, int * largest, int * smallest)
{
   int count,largest_so_far,smallest_so_far;

   largest_so_far = numbers[0];
   smallest_so_far = numbers[0];
   for (count = 1; count < n; count++ ) {
      if (largest_so_far < numbers[count])
         largest_so_far = numbers[count];
      if (smallest_so_far > numbers[count])
         smallest_so_far = numbers[count];
   }
   *largest = largest_so_far;
   *smallest = smallest_so_far;
   return;
}
```

- **Modifying the Function countmarks:**

```
void countmarks(int numbers[], int n, double avgnum,
   int * num_below, int * num_above, int * num_equal)
{
   int count;

   *num_below = *num_above = *num_equal = 0;
   for (count = 0; count < n; count++ )
      if ((double)numbers[count] < avgnum)
         (*num_below)++ ;
      else if ((double)numbers[count] > avgnum)
         (*num_above)++ ;
      else
         (*num_equal)++ ;
   return;
}
```

- **Revision of the Main Program:**

```
/* Program to process exam grades */
#include <stdio.h>
#define SIZE 40
/* Function Prototypes */
void readdata(int [], int *);
int sumarray(int [], int);
double avgarray(int [], int);
void findlimits(int [], int, int *, int *);
void countmarks(int [], int, double, int *, int *, int *);

void main()
{
    int count,num,hi_mark,low_mark;
    int mark[SIZE];
    double avgmark;
    int marks_below,marks_above,marks_equal;

    // call function to read the marks
    readdata(mark,&num);

    // print the mark array
    for (count = 0; count < num; count++)
        printf("mark[%d] = %d\n",count,mark[count]);

    // find and print the average mark
    avgmark = avgarray(mark,num);
    printf("The average is %6.2f\n",avgmark);

    // find and print the highest mark & the lowest mark
    findlimits(mark,num,&hi_mark,&low_mark);
    printf("The highest mark is %d, the lowest mark is %d\n",
        hi_mark,low_mark);

    // classify marks w.r.t. average mark
    countmarks(mark,num,avgmark,&marks_below,&marks_above,
        &marks_equal);
    printf("below = %d above = %d equal = %d\n", marks_below,
        marks_above, marks_equal);
}
```

Modifying Some Functions to Use Pointer Notation

- **Revised Function readdata:**

```
/* ... */
void readdata(int * numbers, int * n)
{
    int count;

    printf("Enter the number of marks: ");
    scanf("%d",n);

    // enter marks
    for (count = 0; count < * n; count++ ) {
        printf("Enter a mark: ");
        scanf("%d", numbers+ count);
    }
    return;
}
```

- **Modifying Function findlimits:**

```
/* ... */
void findlimits(int * numbers, int n, int * largest, int * smallest)
{
    int count,largest_so_far,smallest_so_far;

    largest_so_far = smallest_so_far = * numbers;
    for (count = 1; count < n; count++ ) {
        if (largest_so_far < * (numbers+ count))
            largest_so_far = * (numbers+ count);
        if (smallest_so_far > * (numbers+ count))
            smallest_so_far = * (numbers+ count);
    }
    * largest = largest_so_far;
    * smallest = smallest_so_far;
    return;
}
```

- **Another Revision of the Main Program:**

```
/* Program to process exam grades */
#include <stdio.h>
#define SIZE 40
/* Function Prototypes */
void readdata(int *, int *);
int sumarray(int *, int);
double avgarray(int *, int);
void findlimits(int *, int, int *, int *);
void countmarks(int *, int, double, int *, int *, int *);

void main()
{
    int count,num,hi_mark,low_mark;
    int mark[SIZE];
    double avgmark;
    int marks_below,marks_above,marks_equal;

    // call function to read the marks
    readdata(mark,&num);

    // print the mark array
    for (count = 0; count < num; count++)
        printf("mark[%d] = %d\n",count,mark[count]);

    // find and print the average mark
    avgmark = avgarray(mark,num);
    printf("The average is %6.2f\n",avgmark);

    // find and print the highest mark and the lowest mark
    findlimits(mark,num,&hi_mark,&low_mark);
    printf("The highest mark is %d, the lowest mark is %d\n",
        hi_mark,low_mark);

    // classify marks w.r.t. average mark
    countmarks(mark,num,avgmark,&marks_below,&marks_above,
        &marks_equal);
    printf("below = %d above = %d equal = %d\n", marks_below,
        marks_above, marks_equal);
}
```