**Pointer Variables in C**

- A pointer variable is a variable whose value is the address of another variable.
- "A pointer *points to* another variable"
- A pointer is not an *int*. It is a completely separate data type.
- Two pointer operators (both unary):
  & - produces the address of a variable
  * - retrieves the value pointed to by pointer

**Pointers**

```c
int x, y, *px;
px = &x;
y = *px;
```

In declarations, say "pointer to"

say "address of"

say "value of" or "value pointed to by"

_is equivalent to:_

```c
y = x;
```
Ivalues and rvalues in C

- The "value of a variable" is an rvalue
- The address of a variable is an lvalue
- Arithmetic expressions in C produce rvalues
- Pointer expressions produce lvalues

Pointer Examples

```c
int x = 1, *px;
px = &x;
*px = 2;
x = *px + 1;
```
Pointers and Arrays

An array name (used without a subscript) IS a pointer value, by definition:

```c
int a[10];
```

```
a ⇒ &a[0]
```

It is a pointer constant, ALWAYS containing the address of the first element (element 0).

Anatomy of an Array

```c
int a[10];
```

```
&a[0] or simply a
```

The name of array used without a subscript is a “pointer constant” that points to the first element of the array.

```
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]
```
Pointers and Arrays

char a[] = "This is a test!";
char x, y, z, *pa, *pal, *pan;
int n = 5;

pa = &a[0];  // addr of a[0] into pa
pa = a;     // identical to above!
x = *pa;    // value of a[0] into x

pal = pa + 1;  // addr of a[1] into pal
y = *(pa + 1); // value of a[1] into y
y = a[1];    // identical to above!
y = *pal;    // also identical to above!

pan = pa + n;  // addr of a[n] into pan
z = *pan;     // value of a[n] into z
z = pa[n];    // identical to above!

NOTE: The forms a[], pa[], *a + 1, and *(pa + 1) are completely equivalent and interchangeable!

Pointers and Arrays

Pictorial representation of previous code

```
0 'T'
1 'h'
2 'i'
3 'c'
4 'e'
5 'i'
6 'a'
7 'r'
8 'o'
9 'a'
10 'T'
11 'a'
12 'a'
13 't'
14 '1'
15 '
```
**Pointer Arithmetic**

Simple arithmetic operations can be performed with pointers.

Pointer arithmetic is generally only useful with, and is "system independent" only with, pointers which point to elements of the same array.

Pointers are NOT ints!! The actual value of a pointer will be adjusted by the length (in bytes) of the data type being pointed to.

It is generally good NOT to know or assume anything about the actual value of a pointer -- only assume that it is a value that in some way contains the address of another variable.

---

**Pointer Arithmetic**

\[ p = p + 1; \quad p++; \]

Causes \( p \) to point to the next data element following the one that \( p \) originally pointed to, regardless of the size of each element.

\[ q = p + 1; \]

\( q \) points to the data element 1 positions away from the one that \( p \) points to.

\[ n = q - p; \]

\( n \) is the number of elements between \( p \) and \( q \), an int

\[ p == q; \quad p < q \]

are legal expressions, but probably only useful if \( p \) and \( q \) both point to elements within the same array.
Array Processing with Pointers

```c
int a[100], *pa;

Traditional High-Level Language Method:

```c
for (i = 0; i < 100; i++)
    a[i] = 0;
```

"Assembly Language" Method:

```c
pa = &a[0];  // or simply, pa = a;
while (pa < a + 100)
    *pa++ = 0;
```

Which of these are legal?

```c
int a[100], *pa, *qa;

pa = a;       // Line 1
a = pa;       // Line 2
pa++;         // Line 3
a++;          // Line 4
qa = pa;      // Line 5
*qa = *pa;    // Line 6
pa = *qa;     // Line 7
pa = &a;      // Line 8
```

Ans: Lines 1, 3, 5, 6 are legal.
"Passing" Arrays

Arrays are NEVER actually passed to functions! Instead, the array name (without a subscript) is specified – this is actually a pointer value.

So, a pointer to the start of the array is passed to the function; this pointer can be used to access elements of the array.

Therefore, arrays always "appear" to be passed by reference!

Array Passing Example

```c
void main()
{
    int arrsum(int a[])
    {
        int s = 0;
        //define a
        sum = arrsum(a);
        cout << sum;
    } // END main
}
```

Only a single pointer is passed at time of call!!
Array Passing Example – Pointer Form

```c
void main()
{
    int arrsum(int* a)
    {
        int s[100], sum;
        int s = 0;
        //define a
        sum = arrsum(a);
        cout << sum;
        // END main
    } // END arrsum

    int a[100], *p;
    p = a;
    for(int i=0; i<100; i++)
        a[i] = *p++;
    cout << *p;
    return 0;
}
```

**NOTE:** These two versions of arrsum are exactly equivalent!!