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

PTR0010

University of Idaho

Pointers

```
int x, y, *px;
```

in declarations, say "pointer to"

```
px = &x;
```

say "address of"

```
y = *px;
```

as an operator say "value at" or "value pointed to by"

is equivalent to:

```
y = x;
```

PTR0020

University of Idaho

lvalues and rvalues in C

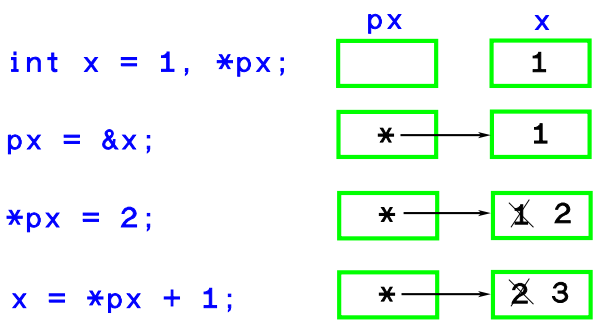
lvalue | *rvalue*

- **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**

PTR0030

University of Idaho

Pointer Examples



PTR0040

University of Idaho

Pointers and Arrays

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

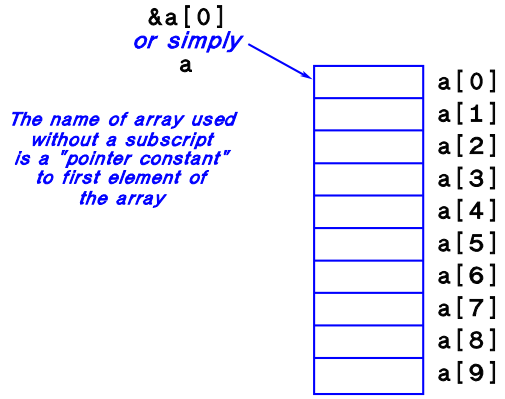
```
int a[10];
```

a ⇛ &a[0]

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

Anatomy of an Array

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



The name of array used without a subscript is a "pointer constant" to first element of the array

Pointers and Arrays

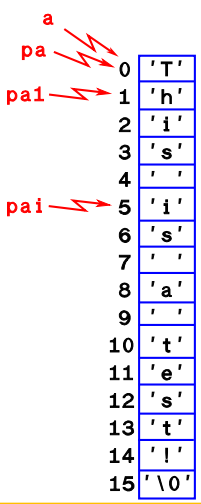
```
char a[] = "This is a test!";
char x, y, *pa, *pa1, *pai;
int i = 5;

pa = &a[0];      // addr of a[0] into pa
pa = a;          // identical to above
x = *pa;         // value of a[0] into x
pa1 = pa + 1;    // addr of a[1] into pa1
pai = pa + i;    // addr of a[i] into pai
y = *(pa + i);   // value of a[i] into y
y = pa[i];       // identical to above!
```

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

PTR0060

University of Idaho

Pointers and Arrays



PTR0065

University of Idaho

Pointer Arithmetic

Simple arithmetic operations can be performed with pointers.

Pointer arithmeteic 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; 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 + i;
```

q points to the data element i 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; p < q
```

are legal expressions, but probably only useful if p and q both point to elements within the same array.

Efficient Array Processing

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

Traditional High-Level Language Method:

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

"Assembly Language" Method:

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


Which of these are legal?

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

pa = a;
a = pa;
pa++;
a++;
qa = pa;
*qa = *pa;
pa = *qa;
pa = &a;
```

"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.

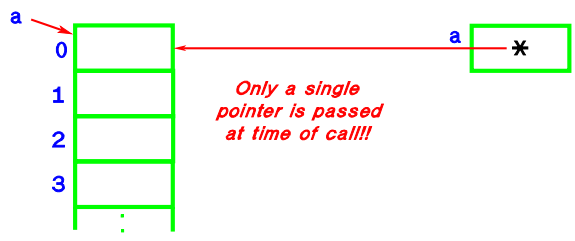
Therefore, arrays always "appear" to be
passed by reference!

Array Passing Example

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

equivalent to &a[0]!

```
int arrsum(int a[])
  {
   int s = 0;

   for(int i=0; i<100; i++)
      s += a[i];
   return(s);
  } // END arrsum
```



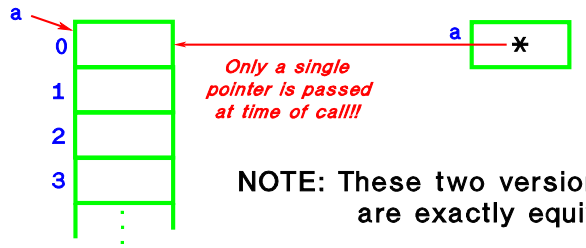
Only a single pointer is passed at time of call!!

Array Passing Example – Pointer Form

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

equivalent to &a[0]!

```
int arrsum(int* a)
  {
   int s = 0;

   for(int i=0; i<100; i++)
      s += *(a + i);
   return(s);
  } // END arrsum
```



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