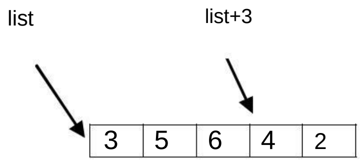
Pointers and One-dimensional Arrays

- Arrays and pointers are closely related in C and may be used almost interchangeably.
- An array name represents the address of the first element of an array.

Example:

```
int list[5] = {3, 5, 6, 4, 2};
```



- The array name `list` is just equivalent to `&list[0]`. To reach to the address of any other element, we can add the subscript of that element to the array name. For example `list+3` represents `&list[3]`.
- We can use the indirection operator to reach to the array elements. For example, both `list[3]` and `*(list+3)` represent the value **4**.

Example: Express 3 in the list

a) `list[0]`

b) `*(list+0) ≡ *list`

Example:

```
char b[50];
char *bptr;
```

- Since the array name without a subscript is a pointer to the first element of the array, we can set `bptr` equal to the address of the first element in array `b` with the statement:

  ```
  bptr = b;
  ```

  This statement is equivalent to taking the address of the first element of the array as follows:

  ```
  bptr = &b[0];
  ```

- Therefore, to reach to the first element of `b` array, we can use `*bptr`:

  ```
  *bptr = ‘A’;
  ```

- What about the following assignment statement?

  ```
  *(bptr + 25) = ‘B’;
  ```

- The **25** in the above expression is the *offset* to the pointer. When the pointer points to the beginning of an array, the offset indicates which element of the array should be referenced, and the offset value is identical to the array subscript. The preceding notation is referred to as *pointer/offset notation*.

- The parantheses are necessary because the precedence of ***** is higher than the precedence of **+**. Without the parantheses, the above expression would add **25** to the value of the expression `*bptr`, thus, **25** would be added to `b[0]`, and this would cause an error since the left side of the assignment statement would be an arithmetic expression.

```
*bptr + 25     ≠ b[25]
               ≡ b[0]+ 25
```

- What does the following `printf` statement display?

```
printf (“%c %c %c %c\n”, b[0], b[25],*(bptr+25),*bptr + 25);
```

Output:

Example: What does the following program display?

```
int main (void)
{int ref[4] = {14, 8, 22, 5}; int
     *ref_ptr;
     int i;
     ref_ptr = ref;
     for (i = 0; i < 4; i++)
     { printf("%3d%3d%3d", ref[i], *(ref + i), *ref + i);
         printf("%3d%3d%3d\n", ref_ptr[i], *(ref_ptr + i),
                                     *ref_ptr + i);
     }
     return(0);
}
```

Output:

- As in the above example, all subscripted array expressions can be written with a pointer and an offset using either the name of the array as a pointer, or a separate pointer that points to the array.

- An array name is a constant pointer that always points to the same location in memory. Array names can not be modified as conventional pointers can. For example, in the above example, we can increment `ref_ptr` as `ref_ptr++` so that it points to `ref[1]`, but `ref++` is an invalid expression.

Example: Write a function that initializes all elements of a one-dimensional integer array to **n**. Use pointer-notation instead of subscript notation.

```
void init_array (int *ap, int size, int n)
{int k;
     for (k = 0; k < size; k++)
         *(ap + k) = n;        // ap[k] = n;
}
```

- When we need a one-dimensional array as the parameter of a function, in fact we need its starting address, which is a pointer. So it is possible to declare the `a` array as `int *ap` in the formal parameter list.

Example:

```
char arr[9] = {'H', 'E', 'L', 'L', 'O', ' ', 'A', 'L',
'I'}; int ind;

ind = 0;
while (arr[ind] != ' ')
     ind++;
printf("Number of elements before blank= %d\n", ind);
```

can also be written as

```
char arr[9] = {'H', 'E', 'L', 'L', 'O', ' ', 'A', 'L',
'I'}; char *arr_ptr;

arr_ptr = arr;
while (*arr_ptr != ' ')
     arr_ptr++;
printf("Number of elements before blank= %d\n",arr_ptr-arr);
```

- Array subscripting notation is converted to pointer notation during compilation, so writing array subscripting expressions with pointer notation, can save compile time. But array subscripting notation will probably be much clearer.

Example: What does the following program segment display?

```
int a[5] = {2, 5, 8, 9,
3}; int *b;

b = a;
printf("%d\n", b[2]);
*b = 3;
*(b + 2) = 4;
printf("%d %d\n", b[0], b[2]);
printf("%d %d\n", a[0], a[2]);
```

Output:

➢ *READ Sec. 7.8 (pg 272 – 276) from Deitel & Deitel.*

Home Exercise: What does the following program do? Complete the output.

```
void f (double **p)
{
     (*p)++;
}

int  main (void)
{
     double num[3] = {7.5, 5.0, 2.5};
     double *pp = num;
     printf("%p %0.1f\n", pp, *pp);
     f(&pp);
     printf("%p %0.1f\n", pp, *pp);
     return(0);
}
```

```
0012FF0C   ?

?          ?
```