CSE 142 Programming I

Arrays and Pointers: Review and Examples

Array Type Quiz

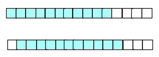
```
AFunction (int a1[ ], int *sp) {
int a2[MAXA];
int N;
...
a1 = a2;                /*1. */
a1[MAXA] = a2[MAXA]; /*2. */
N = a1[0];              /*3. */
a1[01] = sp;            /*4. */
*sp = a1[0];            /*5. */
printf (“%d”, a1);      /*6. */
printf (“%d”, a2[0]);   /*7. */
a1[a2[*sp]] = 1;        /*8. */
scanf (“%d%d%d%d%d”, a1[1], &a1[1], *a1[1], sp, *sp);  /*9. */
```

Pointer Type Quiz

```
QFunct (int i,  int * ip, double x,
          double * xp)
{
...
x = i;     /* 1. */
i = x;     /* 2. */

ip = 30;  /* 3. */
ip = i;     /* 4. */
ip = &i;  /* 5. */
ip = &x; /* 6. */
xp = ip;  /* 7. */
&i =  ip; /* 8. */
```

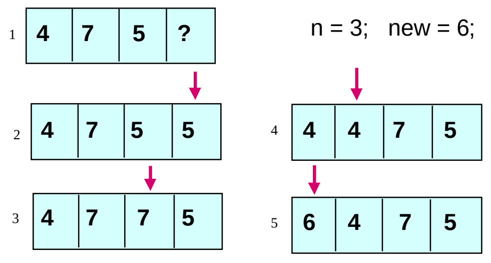
Shifting Array Elements

```
/* Shift x[0], x[1], ..., x[n-1] one position upwards
    to make space for a new element at x[0].
    Insert  the value new at x[0].
    Update the value of n.
*/
for ( k = n ;   k >= 1 ;   k = k - 1 )
    x[k] = x[k-1] ;
x[0] = new ;
n = n+1 ;
```



Shifting Array Elements

n = 3;   new = 6;



Searching

- Searching = looking for something
- Searching an array is particularly common
  - Goal: determine if a particular value is in the array
- If the array is unsorted:
  - start at the beginning and look at each element to see if it matches

Linear Search

```
/* If x appears in a[0..n-1], return its index, i.e.,
   return k so that a[k]==x.  If x not found, return -1 */
int search (int a[ ], int n, int x) {
   int index = 0;
   while (index < n && a[index] != x) {
     index++;
     }
   if (index < n)
     return index;
   else return -1;
}
```

Linear Search

v	3	12	-5	6	142	21	-17	45

- **Test:**
  - **search(v, 8, 12)**
  - **search(v, 8, 15)**
- **Note: Condition in *while* relies on short-circuit evaluation of && (i.e., a[index] might not be defined if index>=n).**

Can we do better?

- **"Binary search" works *if the array is already sorted***
  1. **Look for the target in the middle.**
  2. **If you don't find it, you can ignore half of the array, and repeat the process with the other half.**
- **Example: Find first page of Pizza listings in the yellow pages**

Is it worth the trouble?

- **Suppose you had 1000 elements**
- **Ordinary search would require maybe 500 comparisons on average**
- **Binary search**
  - **after 1st compare, throw away half, leaving 500 elements to be searched.**
  - **after 2nd compare, throw away half, leaving 250. Then 125, 63, 32, 16, 8, 4, 2, 1 are left.**
  - **After at most 10 steps, you're done!**

What if you had 1,000,000 elements??

Whole Arrays as Parameters

```
#define ARRAY_SIZE  200
double average ( int a[ARRAY_SIZE] ) {
     int i, total = 0 ;
     for ( i = 0 ;  i < ARRAY_SIZE ;  i = i + 1 )
        total = total + a[i]  ;
     return ((double) total /  (double) ARRAY_SIZE) ;
}

int x[ARRAY_SIZE] ;
...
x_avg = average ( x ) ;
```

Arrays as Output Parameters

```
          /* Sets vsum to sum of  vectors a and b. */
          void VectorSum( int a[3], int b[3], int vsum[3])  {
               int i ;
note:          for ( i = 0 ;  i < 3 ;  i = i + 1 )
no *              vsum[i] = a[i] + b[i] ;
no &      }

          int main(void)  {
               int x[3] = {1,2,3},  y[3] = {4,5,6},  z[3] ;
               VectorSum( x , y , z );
               printf( "%d %d %d", z[0], z[1], z[2] ) ;
          }
```

General Vector Sum

```
void VectorSum( int a[ ] , int b[ ] ,
                         int vsum[ ] , int length)  {
    int i ;
    for ( i = 0 ;  i < length ;  i = i + 1 )
      vsum[i] = a[i] + b[i] ;
}
```

```
int x[3] = {1,2,3},  y[3] = {4,5,6},  z[3] ;
VectorSum( x , y , z , 3 );
```

Array Parameter Summary

Array elements:

Just like simple variables of that type, both input & output parameters

Whole arrays:

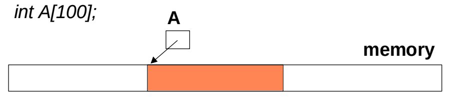
Arrays are not passed by value, i.e. not copied

Formal parameter: *type array_name [SIZE]*
Or : *type array_name [ ]*
no *

Actual parameter: *array_name*
no [ ] , no &

An Array as a Pointer

int A[100];



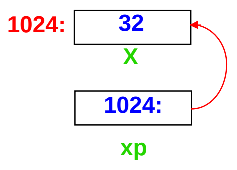
A[0]	**equivalent to**	***A**
A[i]	**equivalent to**	***(A + i)**

pointer addition

Review: Pointer

A pointer contains a reference to another variable; that is, the pointer contains the address of a variable.

xp is a pointer to int



Addresses and Pointers

Three new types:

int *	“pointer to int”
double *	“pointer to double”
char *	“pointer to char”

Two new (unary) operators:

& “address of”
& can be applied to any variable (or param)

* “location pointed to by”
* can be applied only to a pointer

Vocabulary

- Dereferencing or indirection:
  - following a pointer to a memory location
- Output parameter:
  - a pointer parameter of a function
  - can be used to provide a value ("input") as usual, and/or store a changed value ("output")
  - Don’t confuse with printed output (printf)

Sorting

Problem: Sort 3 integers

Three-step algorithm:

1. *Read in three integers: x, y, z*
2. *Put smallest in x:*

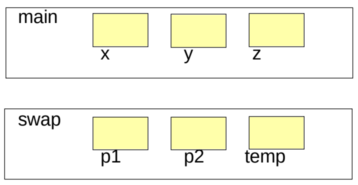
   Swap x, y if necessary; then swap x, z, if necessary.
3. *Put second smallest in y:*

   Swap y, z, if necessary.

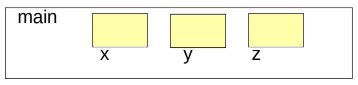
Sort 3 Integers as a Program

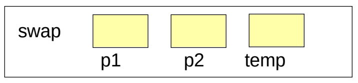
```
int main (void) {
int x, y, z, scanStatus ;
…
scanStatus = scanf("%d%d%d", &x, &y, &z) ;
if scanStatus == 3 {
   if ( x > y )  swap(&x, &y) ;
   if ( x > z )  swap(&x, &z) ;
   if ( y > z )  swap(&y, &z) ;
```

Trace



Trace





sort3 as a Function

```
/* interchange values as needed to establish */
/* *xp <= *yp <= *zp  */
void sort3 (int *xp, int *yp, int *zp)  {
    if ( *xp > *yp )  swap(xp, yp) ;
    if ( *xp > *zp )  swap(xp, zp) ;        ← NO &s!
    if ( *yp > *zp )  swap(yp, zp) ;
}

int main(void) {
    int x, y, z ;
    ...  /*scanf the values, then: */
    sort3(&x, &y, &z) ;
    ...
}
```

Why no & in swap call?

Real reason

xp and yp are already pointers that point to the variables that we want to swap

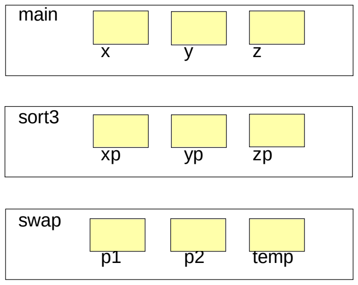
Alternative explanation using alias idea

*xp and *yp are aliases for the variables we want to swap

We want to allow swap to use aliases for *xp and *yp so we should use &(*xp) and &(*yp) in the call

BUT   xp==&(*xp) and yp==&(*yp)   !!!!

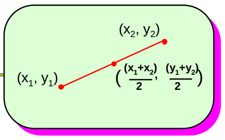
Trace



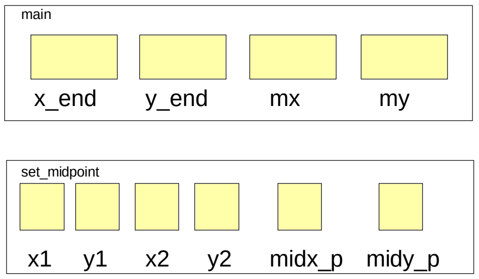
Midpoint Of A Line

/* Given 2 endpoints of a line, "return" coordinates of midpoint */

```
void set_midpoint(
    double x1, double y1,              /* 1st endpoint          */
    double x2, double y2,              /* 2nd endpoint          */
    double * midx_p, double * midy_p ) /* Pointers to midpoint  */
{
    *midx_p = (x1 + x2) / 2.0;
    *midy_p = (y1 + y2) / 2.0;
}

double x_end, y_end, mx, my;
...
set_midpoint(0.0, 0.0, x_end, y_end, &mx, &my);
```



Trace



Example: Coordinates

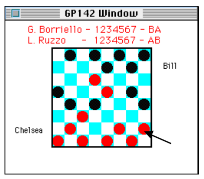
Board Coordinates

row, column

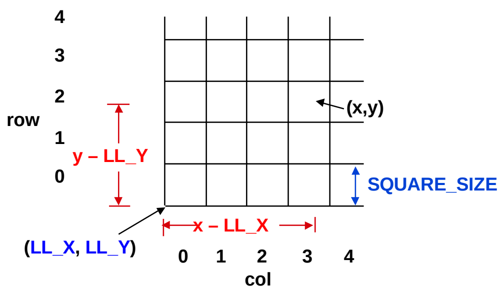
Screen Coordinates

x, y

used by graphics package



Coordinate Conversion



Coordinate Conversion

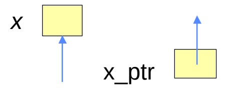
```
#define LL_X      40
#define LL_Y      20
#define SQUARE_SIZE  10

void screen_to_board (
    int screenx, int screeny,       /* coordinates on screen */
    int * row_p,  int * col_p)      /* position on board        */
{
    *row_p =  (screeny - LL_Y) / SQUARE_SIZE;
    *col_p  =  (screenx - LL_X) / SQUARE_SIZE;
}
```

```
screen_to_board (x, y, &row, &col);
```

2/16/00 K3-31

Pointers vs. Values

	in caller	in callee
Declaration:	*int x*	*int * x_ptr*
To get the address of x:	*&x*	*x_ptr*
To get the value of x:	*x*	**x_ptr*



2/16/00 K3-32

Array Type Quiz: Answers

```
AFunction (int a1[ ], int *sp) {
int a2[MAXA];
int N;
...
a1 = a2;           /*1.  no way*/
a1[MAXA] = a2[MAXA]; /*2.  logic error*/
N = a1[0];         /*3.  OK*/
a1[01] = sp;     /*4.  nope*/
*sp = a1[0];     /*5.  OK*/
printf (“%d”, a1);        /*6.  nope*/
printf (“%d”, a2[0]);   /*7.  OK */
a1[a2[*sp]] = 1             /*8. OK*/
scanf (“%d%d%d%d%d”,
a1[1], &a1[1], *a1[1], sp, *sp);/*9.  no, yes, meaningless, yes, no*/
```

2/16/00 K3-33

Pointer Type Quiz: Answers

```
QFunct (int i,  int * ip, double x,
            double * xp)
{
...
x = i;      /* 1. no problem */
i = x;      /* 2. not recommended */

ip = 30;  /* 3. No way */
ip = i;    /* 4. Nope */
ip = &i;  /* 5. just fine */
ip = &x; /* 6. forget it! */
xp = ip;  /* 7. bad */
&i =  ip; /* 8. meaningless */
```

2/16/00 K3-34