Arrays

Comp 1002/1402

A Problem with Variables

- Single variables are restrictive

Problem:

We need to use 20, 200, or 2000 variables

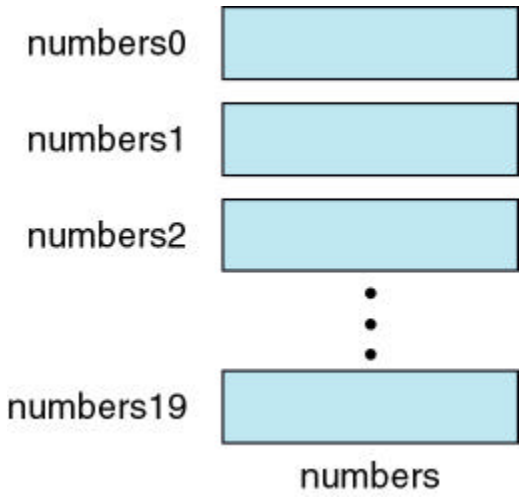
Example:

Databases access huge amounts of data

Single Variable Solution

Solution:

One Variable per data item



The Issue with that Solution

- 20 variable names makes a lot of code
- No good way to write such code
- We require a single data structure

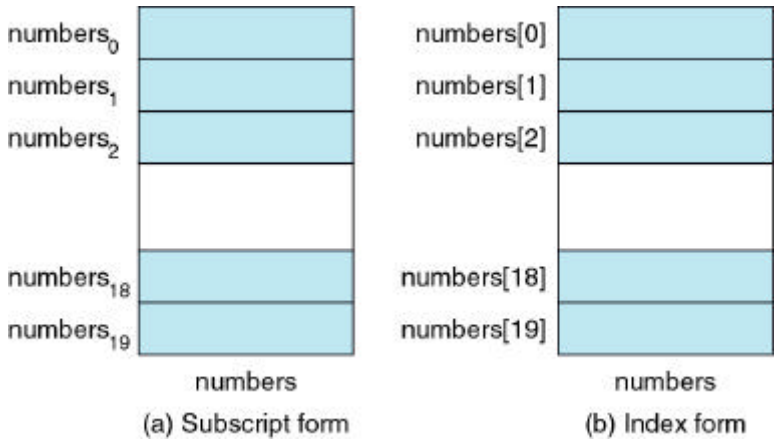
Arrays

An **array** is a fixed-size, sequenced collection of elements of the same data type

In mathematics components of an array A are subscripted: A_0, A_1, A_2, … , A_{n-1}

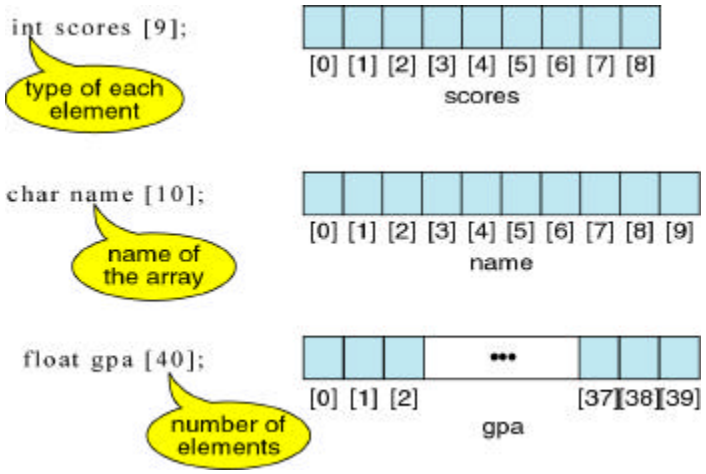
In C arrays use indexes

Array Indexing



Array Declaration

- Must be declared (like any other variable)



How Arrays Work (Basics)

```
int scores[10];
/* defines array of ints */
```

`scores` is the variable

It is a pointer to the first byte of element 0

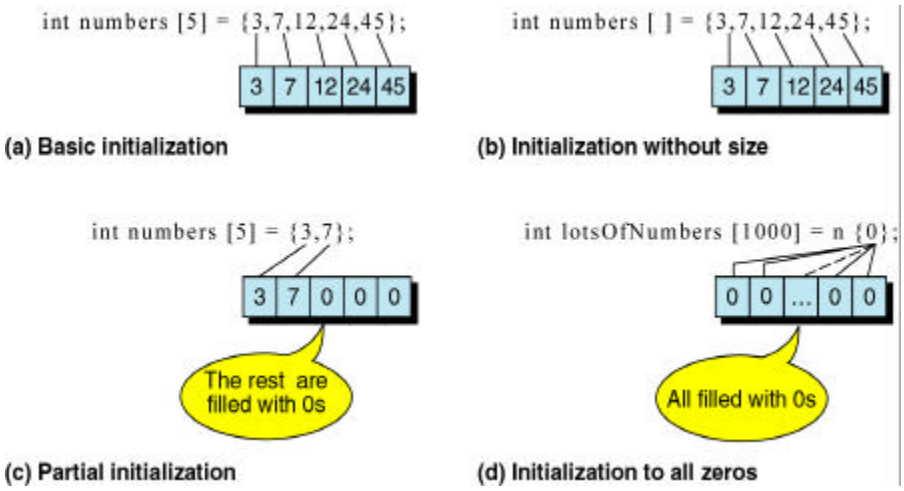
That is **`(scores == &(scores[0]))`**
is always true!

More on the ways Arrays Work

```
int scores[10];
/* defines array of ints */

scores[0]
/* looks up contents at scores */

scores[7]
/* looks up contents 7 ints from scores*/
```

This is all done with pointers (discussed later)

Initialization



Setting & Copying Arrays

Set every element:

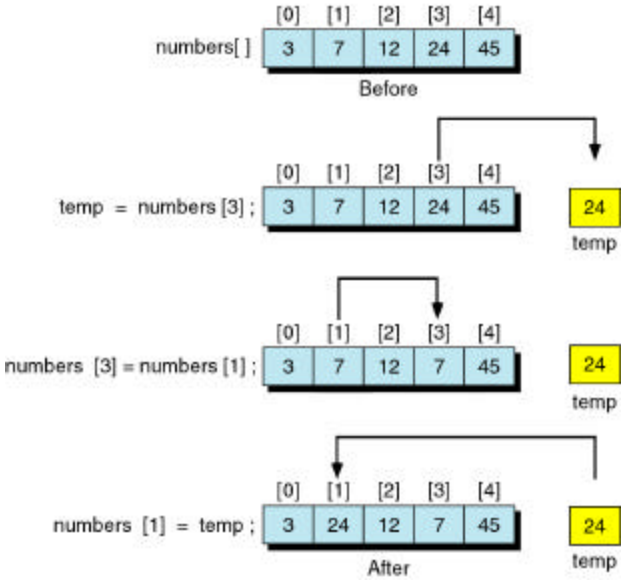
(Can only use {} notation at declaration!)

```
for (i=0; i<25; i++)
  first[i] = (i*2) + 1;
```

Copy every element:

```
for (i=0; i<25; i++)
  second[i] = first[i];
```

Exchanging Values

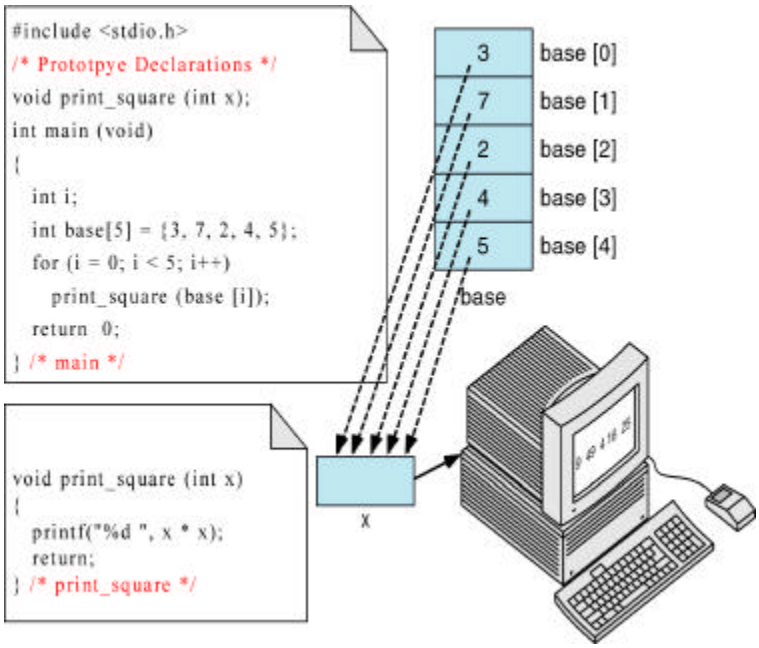


Reading into Arrays

- Use the address of the variable

```
unsigned int grades[10];
int i;
for (i = 0; i<= 9; i++)
  scanf("%u",&(scores[i]));
```

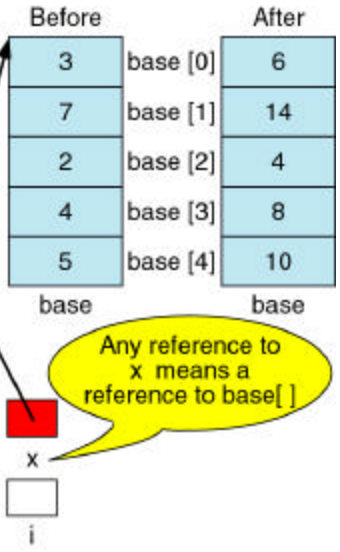
Passing Individual Elements



Passing the Whole Array

```
/* Prototype Declarations */
void multiply2 (int x[]);
#include <stdio.h>
int main (void)
{
  int base[5] = {3, 7, 2, 4, 5};
  multiply2 (base);
  return 0;
} /* main */
```

```
void multiply2 (int x[])
{
  int i;
  for (i = 0; i < 5; i++)
      x[i] *= 2;
  return ;
} /* multiply2 */
```



Things **NOT** to do with Arrays

- Usually pass length to functions!!
- Never go past the end of the array!!
- Be aware that a function can change your array

Things to do with Arrays

- Find min or max value
- Compute sum or average
- Sort for item
- Search for item

Max

```
int values[10]=
  {0,-2, 3, 6,-4, 8, 1, 0, 1, -1};
int i;
int max = values[0];
for (i=0;i<10;i++)
  if (values[i] > max)
     max = values[i];
printf("The max is: %d",max);
```

Min

```
int values[10]=
  {0,-2, 3, 6,-4, 8, 1, 0, 1, -1};
int i;
int min = values[0];
for (i=0;i<10;i++)
  if (values[i] > min)
     min = values[i];
printf("The min is: %d",min);
```

Sum

```
int values[10]=
  {0,-2, 3, 6,-4, 8, 1, 0, 1, -1};
int i;
int sum = 0;
for (i=0;i<10;i++)
  sum += values[i];
printf("The sum is: %d",sum);
```

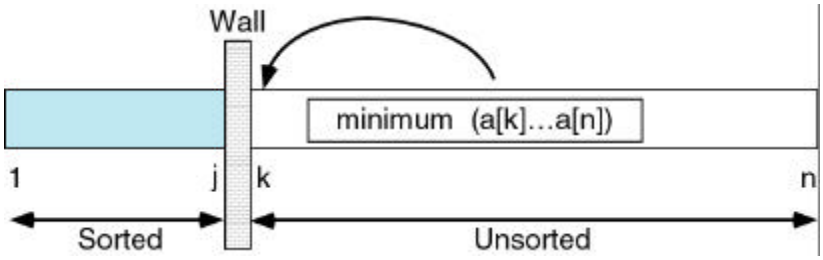
Average

```
int values[10]=
  {0,-2, 3, 6,-4, 8, 1, 0, 1, -1};
int i, sum = 0;
double ave;
for (i=0;i<10;i++)
  sum += values[i];
ave = sum / 10.0;
printf("The average is: %f",ave);
```

Basic Sorting Techniques

- Many techniques available to sort
- Most basic (and long way):
  - Place an element in sorted part at front
  - Sort remainder
  - Three sorting algorithms
    (Bubble, Selection, Insertion)

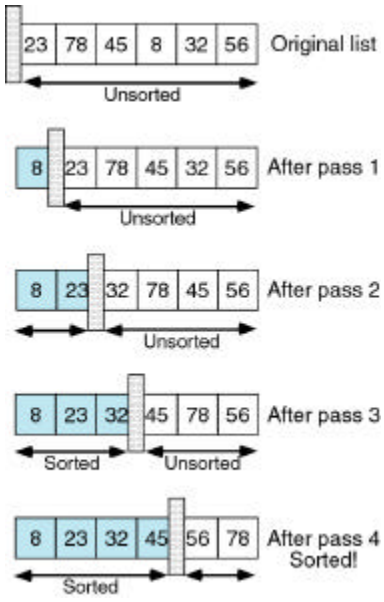
Bubble & Selection



Bubble Sort

- Start at the last element
- Swap any two out of order neighbours
- Every pass ensures one more in order

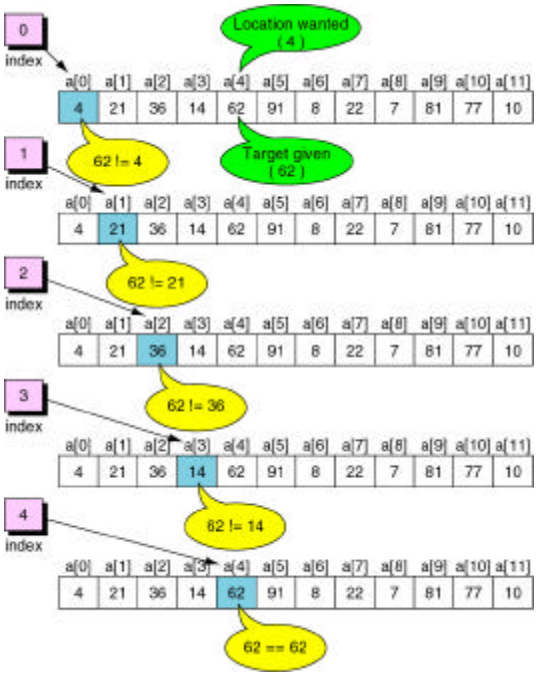
Bubble Sort Example



Bubble Sort Code

```
const int size = 5;
int array[size]= {34, 1245, 992, 3, 1};
int pass, i, temp;
   /* requires (size -1) passes to complete */
for (pass = 0; pass > size; pass++)   {
  for (i = size; i>=pass; i--) {
      if (array[i] > array[i+1]) {
                  temp = array[i];
                  array[i] = array[i+1];
                  array[i+1] = temp;
            }
      }
  }
```

Searching



Sequential Searching

```
int find(int num, int values[], int size){
  int found = 0, i = 0;
  while ((i<size)&&(found==0)) {
    if (values[i++] == num)
      found = 1;
  }
  if (found == 1) return (i-1);
  else return –1;
}
```

Searching a Sorted Array

Problem: Find the element in the array

Solution:

Continually half the search space

If the midpoint is the number…

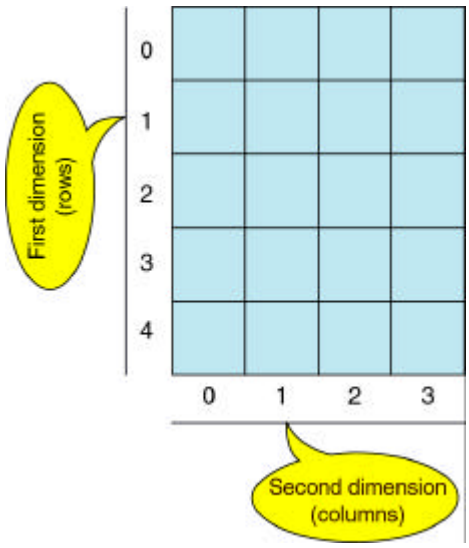
If the midpoint is too high…

If the midpoint is too low…

Binary Search

```
int bsearch(int list[], int size, int find, int *locn) {

  int first = 0, last = size, mid;

  while (first <= last) {
      mid = (first+last)/2;
      if (target > list[mid])  first = mid + 1;
      else if (target < list[mid])  last = mid –1;
      else first = last + 1;
  }

  *locn = mid;
  return (target == list[mid]);
}
```

Two Dimensional Arrays



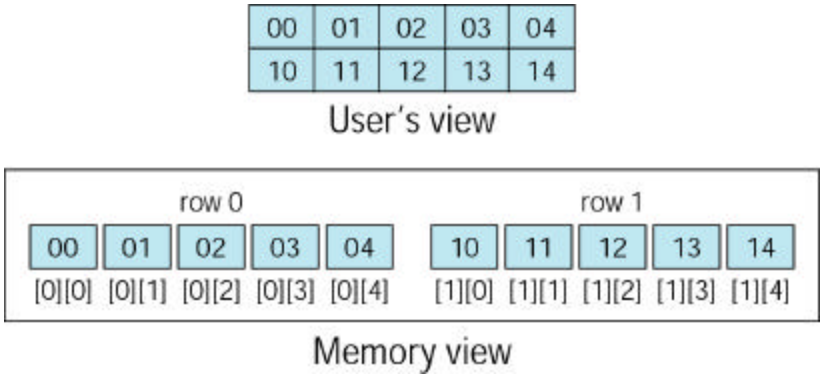
Declaration & Initialization

```
int table[5][4]; /* no initialization */

int table[5][4] =
  {0,1,2,3,10,11,12,13,20,21,22,23,30,31,32,33,40,4
  1,42,43}; /* poor style in initialization */

int table[5][4] = {
  { 0, 1, 2, 3},
  {10,11,12,13},
  {20,21,22,23},
  {30,31,32,33},
  {40,41,42,43}}; /*table*/

int table [5][4] = {0}; /* all zeroes*/
```

Looping Through 2d Arrays

```
for (row=0;row<5;row++) {
  for (column = 0; column < 4; column++)
     printf("%8d",table[row][column]);
  printf("\n");
} /* for */
```

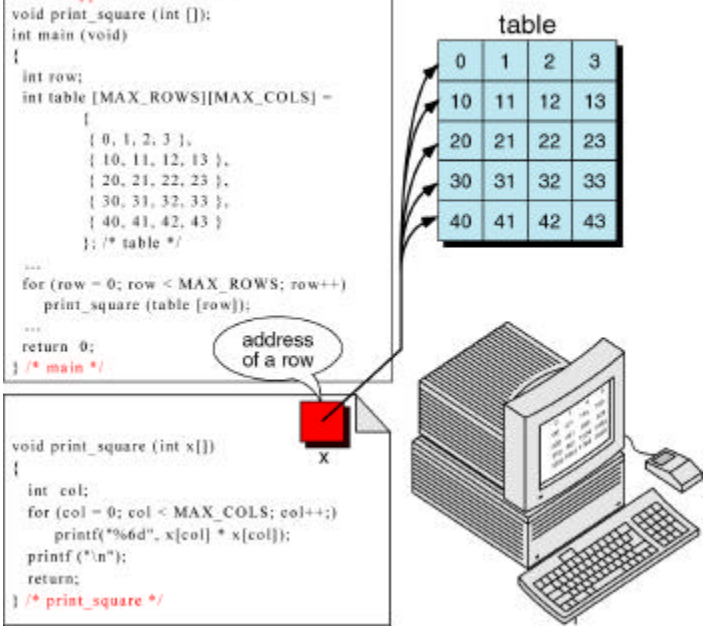
Memory View



Passing a Row

```
#define MAX_ROWS  5
#define MAX_COLS  4
/* Prototype Declarations */
void print_square (int []);
int main (void)
{
  int row;
  int table [MAX_ROWS][MAX_COLS] =
            {
              { 0, 1, 2, 3 },
              { 10, 11, 12, 13 },
              { 20, 21, 22, 23 },
              { 30, 31, 32, 33 },
              { 40, 41, 42, 43 }
            }; /* table */
  ...
  for (row = 0; row < MAX_ROWS; row++)
     print_square (table [row]);
  ...
  return  0;
} /* main */
```

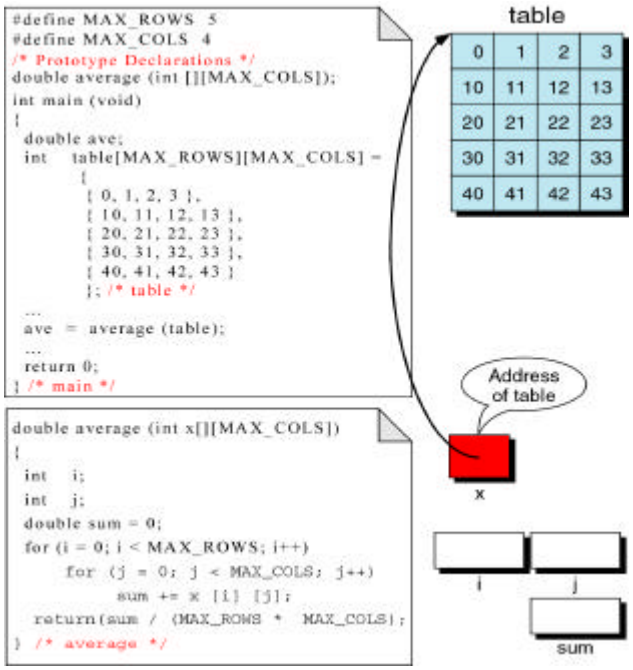
```
void print_square (int x[])
{
  int  col;
  for (col = 0; col < MAX_COLS; col++;)
      printf("%6d", x[col] * x[col]);
  printf ("\n");
  return;
} /* print_square */
```



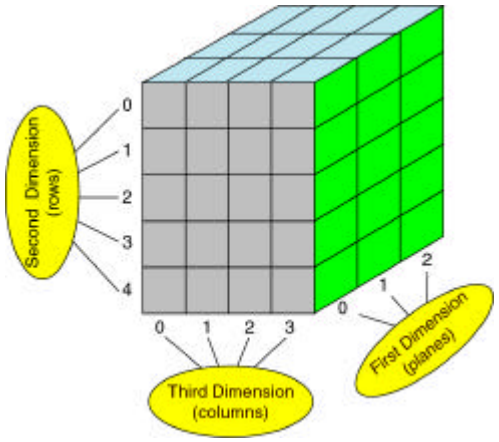
Passing a 2D Array

```
#define MAX_ROWS  5
#define MAX_COLS  4
/* Prototype Declarations */
double average (int [][MAX_COLS]);
int main (void)
{
  double ave;
  int    table[MAX_ROWS][MAX_COLS] =
           {
             { 0, 1, 2, 3 },
             { 10, 11, 12, 13 },
             { 20, 21, 22, 23 },
             { 30, 31, 32, 33 },
             { 40, 41, 42, 43 }
           }; /* table */
  ...
  ave  =  average (table);
  ...
  return 0;
} /* main */
```

```
double average (int x[][MAX_COLS])
{
  int    i;
  int    j;
  double sum = 0;
  for (i = 0; i < MAX_ROWS; i++)
       for (j = 0; j < MAX_COLS; j++)
            sum += x [i] [j];
  return(sum / (MAX_ROWS *  MAX_COLS));
} /* average */
```



3D



Memory Representation