Chapter7: Arrays

Computer Science Department

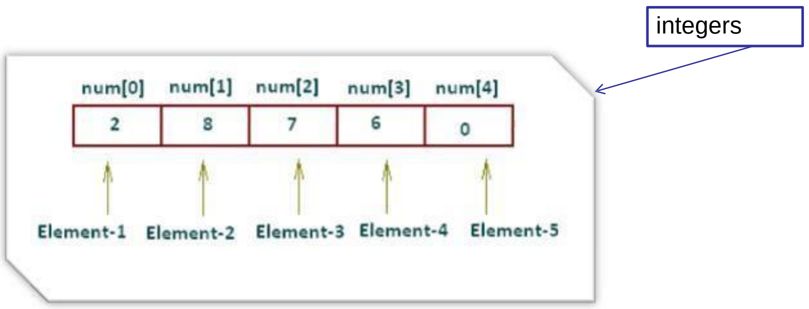
Arrays - Introduction

- Simple data types use a single memory cell to store a variable.
- Sometimes, we need to *group data items* together in main memory than to allocate an individual memory cell for each variable.
- Example: A program that processes exam scores for a class.
- Here, it would be easier to write if all the scores were stored in one area of memory and were able to be accessed as a group

Arrays - definition

Array is a collection of data items of the same type.

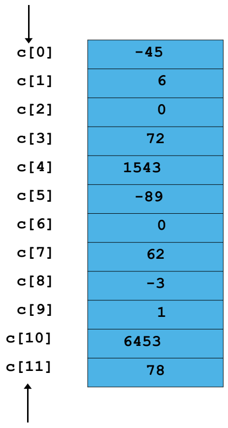
Array element is a *data item* that is part of an array.



Arrays

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

  arrayname[*position number*]
  - First element at position `0`
  - `n` element array named `c`:
    - `c[ 0 ], c[ 1 ]...c[ n – 1 ]`



Position number of the element within array c

Declaring Arrays

- When declaring arrays, specify

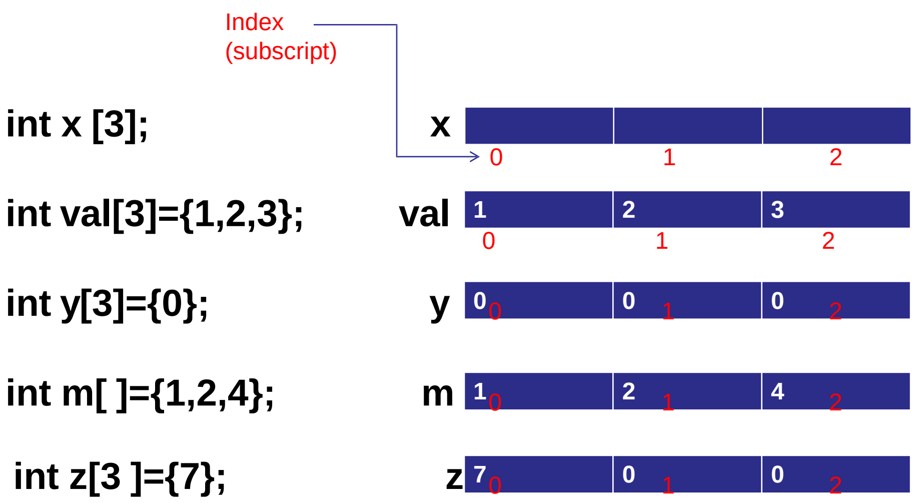
```
arrayType arrayName[numberOfElements];
```

e.g.
```
int c[ 10 ];
float myArray[ 100 ];
```

- Declaring multiple arrays of same type
  - Format similar to regular variables

e.g.
```
int b[ 100 ], x[ 27 ];
```

Declaring Arrays



Arrays

Array elements are like *normal variables*

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
c[1]= c[0]+c[2]
c[3]= c[2]+5
```

Perform operations in subscript (index).

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

Example: Operations on Arrays

```
double x[8];
```

Array `x`

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Statements that manipulate the value of x

Statement	Explanation
`printf("%.1f", x[0]);`	Displays the value of `x[0]`, which is `16.0`.
`x[3] = 25.0;`	Stores the value `25.0` in `x[3]`.
`sum = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]`, which is `28.0` in the variable `sum`.
`sum += x[2];`	Adds `x[2]` to `sum`. The new `sum` is `34.0`.
`x[3] += 1.0;`	Adds `1.0` to `x[3]`. The new `x[3]` is `26.0`.
`x[2] = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]` in `x[2]`. The new `x[2]` is `28.0`.

The array x is now:

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	28.0	26.0	2.5	12.0	14.0	–54.5

Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
char alphabet[5] = {'A','B','C','D','E'};
```

- All elements 0

```
int n[ 5 ] = { 0 }
```

- If size omitted, initializers determine it

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

5 initializers, therefore 5 element array

Example Array

```
/*  prints the days for each month */

#include <stdio.h>
#define MONTHS 12

int main(void)
{
int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
int index;
for (index = 0; index < MONTHS; index++)
   printf("Month %d has %2d days.\n", index +1,days[index]);
return 0;
}
```

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

Example: Fill and Print Array

```c
#include <stdio.h>

int main ()
{
   int n[ 10 ];  // n is an array of 10 integers
   int i,j;

    // initialize elements of array      0 (Fill Array)
   for ( i = 0; i < 10; i++ )
   {
      n[ i ] = i + 1; /* set element at location i to i + 1 */
   }

    // output each array element's value (Print Array)
   for (j = 0; j < 10; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Output:

```
Element[0] = 1
Element[1] = 2
Element[2] = 3
Element[3] = 4
Element[4] = 5
Element[5] = 6
Element[6] = 7
Element[7] = 8
Element[8] = 9
Element[9] = 10
```

Example: Fill and Print Array

```c
#include <stdio.h>
#define size 5 //   array size= 5
int main ()
{
   int n[ size ]; //  n is an array of 5 integers
   int i,j;

   //  initialize elements of array n (Fill Array)
   for ( i = 0; i < size; i++ )
   {
      scanf ("%d",&n[ i ]);
   }

   //  output each array element's value (Print Array)
   for (j = 0; j < size; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Input:

```
1 2 3 4 5
```

Output:

```
Element[0] = 1
Element[1] = 2
Element[2] = 3
Element[3] = 4
Element[4] = 5
```

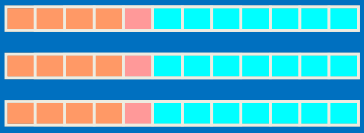
Multi- Dimensional Arrays

- Multidimensional arrays are derived from the basic or built-in data types of the C language.
- Two-dimensional arrays are understood as rows and columns:
  - two-dimensional tables,
  - parallel vectors,
  - two-dimensional matrices.
- Mostly Two-dimensional array are used in Multi-dimensional array

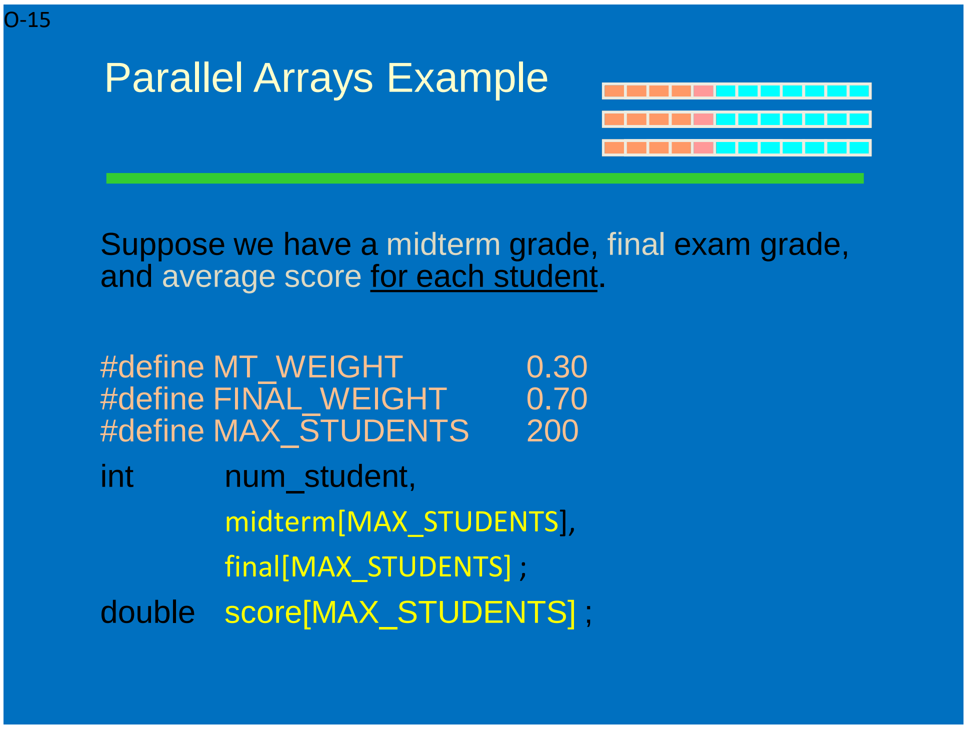
O-14

"Parallel" Arrays

A set of arrays may be used in parallel when more than one piece of information must be stored for each item.



Example: we are keeping track of a group of students. For each item (student), we might have several pieces of information such as scores

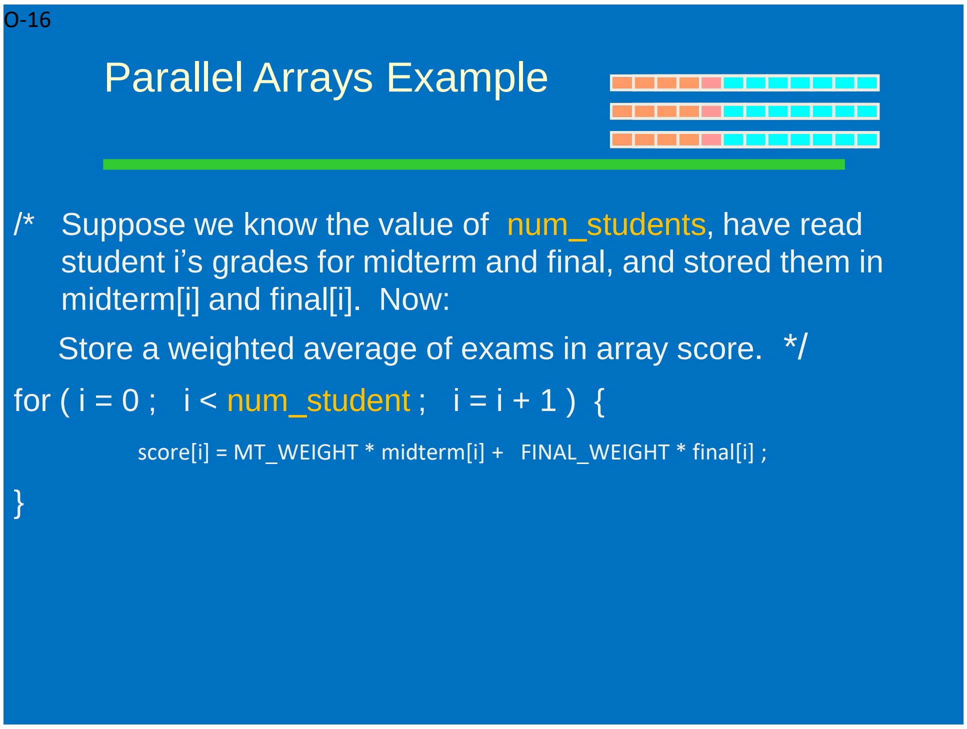


O-15

Parallel Arrays Example

Suppose we have a midterm grade, final exam grade, and average score for each student.

```
#define MT_WEIGHT           0.30
#define FINAL_WEIGHT        0.70
#define MAX_STUDENTS        200
int         num_student,
            midterm[MAX_STUDENTS],
            final[MAX_STUDENTS] ;
double      score[MAX_STUDENTS] ;
```

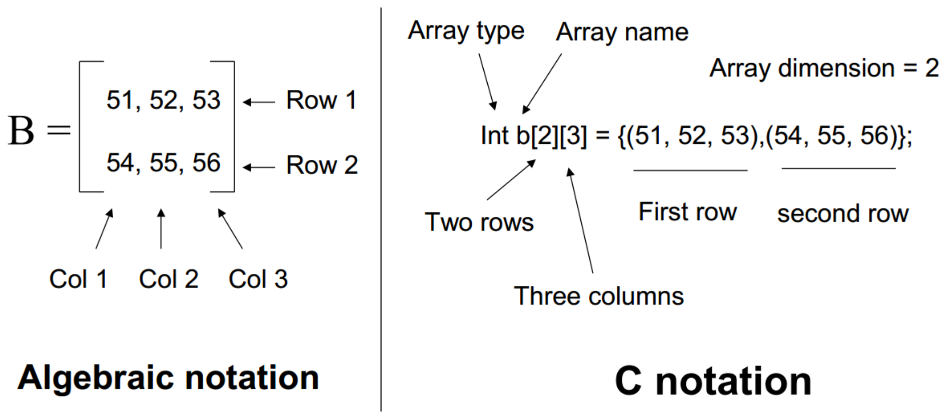


O-16

Parallel Arrays Example

```
/*   Suppose we know the value of  num_students, have read
     student i's grades for midterm and final, and stored them in
     midterm[i] and final[i].  Now:
     Store a weighted average of exams in array score.   */
for ( i = 0 ;   i < num_student ;   i = i + 1 )  {
        score[i] = MT_WEIGHT * midterm[i] +   FINAL_WEIGHT * final[i] ;
}
```

What is a Two-dimensional array?



How to initialize a Two-Dimensional array?

- Initialized directly in the declaration statement
  - int b[2][3] = {51, 52, 53, 54, 55, 56};
  - b[0][0] = 51   b[0][1] = 52   b[0][2] = 53
- Use braces to separate rows in 2-D arrays.
  - int c[4][3] = {{1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}};
  - int c[ ][3] = {{1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}};

  Implicitly declares the number of rows to be 4.

2-Dimenstional Array

Create array elements by telling how many ROWS and COLUMNS

Example:

```
int grades[5][3];
```

grades is a *two-dimensional array*, with 5 rows and 3 columns. One row for each student. One column for each test.

Declare & Initialize

Example:

```
int grades[5][3] =
  { { 78, 83, 82 },
    { 90, 88, 94 },
    { 71, 73, 78 },
    { 97, 96, 95 },
    { 89, 93, 90 } };
```

A Two-D Array is an *array of arrays*.

Each row is itself a One-D array.

Row, Column Indices

	0	1	2
0	78	83	82
1	90	88	94
2	71	73	78
3	97	96	95
4	89	93	90

Give both the ROW and COLUMN indices to pick out an individual element.

The fourth student's third test score is at ROW 3, COLUMN 2

Example

```
int a[2][4];

a[1][0]=9;

a[0][3]=5;

a[0][1]=a[0][3]+ a[1][0];
```

	0	1	2	3
0		14		5
1	9			

R-23

Representation of Arrays

A computer's memory is a *one dimensional array of cells*.

How is a 2-D array stored?

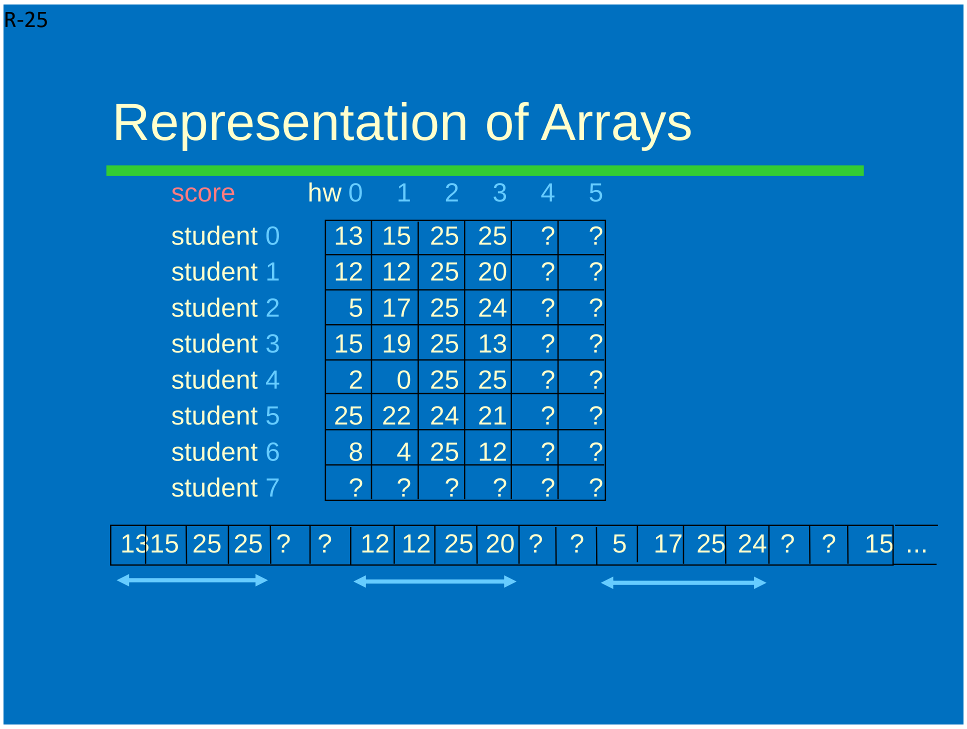
Answer: In C, the array rows are stored **sequentially**: row 0, 1, 2, …

R-24

Representation of Arrays

score	hw 0	1	2	3	4	5
student 0	13	15	25	25	?	?
student 1	12	12	25	20	?	?
student 2	5	17	25	24	?	?
student 3	15	19	25	13	?	?
student 4	2	0	25	25	?	?
student 5	25	22	24	21	?	?
student 6	8	4	25	12	?	?
student 7	?	?	?	?	?	?

R-25

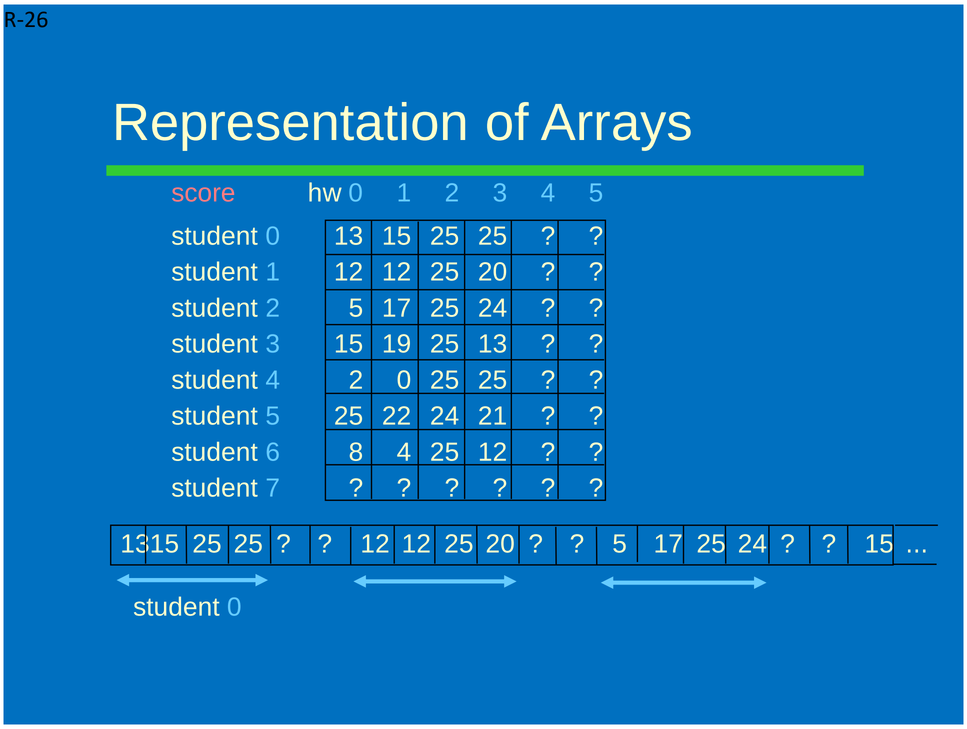


Representation of Arrays

score	hw 0	1	2	3	4	5
student 0	13	15	25	25	?	?
student 1	12	12	25	20	?	?
student 2	5	17	25	24	?	?
student 3	15	19	25	13	?	?
student 4	2	0	25	25	?	?
student 5	25	22	24	21	?	?
student 6	8	4	25	12	?	?
student 7	?	?	?	?	?	?

13	15	25	25	?	?	12	12	25	20	?	?	5	17	25	24	?	?	15	...

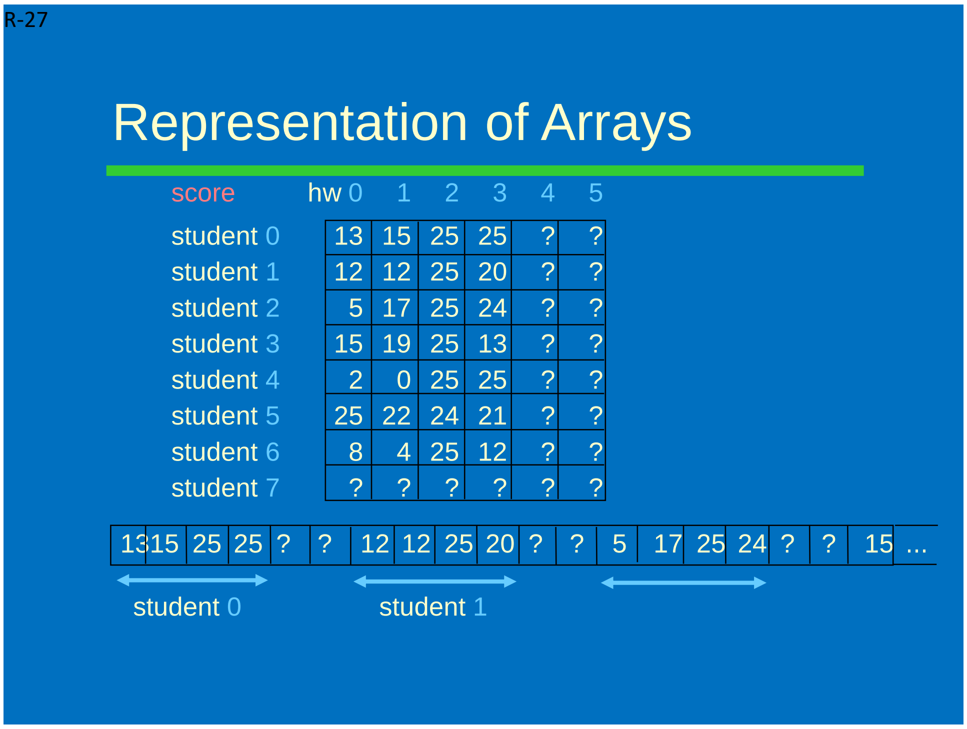
R-26



Representation of Arrays

score	hw 0	1	2	3	4	5
student 0	13	15	25	25	?	?
student 1	12	12	25	20	?	?
student 2	5	17	25	24	?	?
student 3	15	19	25	13	?	?
student 4	2	0	25	25	?	?
student 5	25	22	24	21	?	?
student 6	8	4	25	12	?	?
student 7	?	?	?	?	?	?

13	15	25	25	?	?	12	12	25	20	?	?	5	17	25	24	?	?	15	...



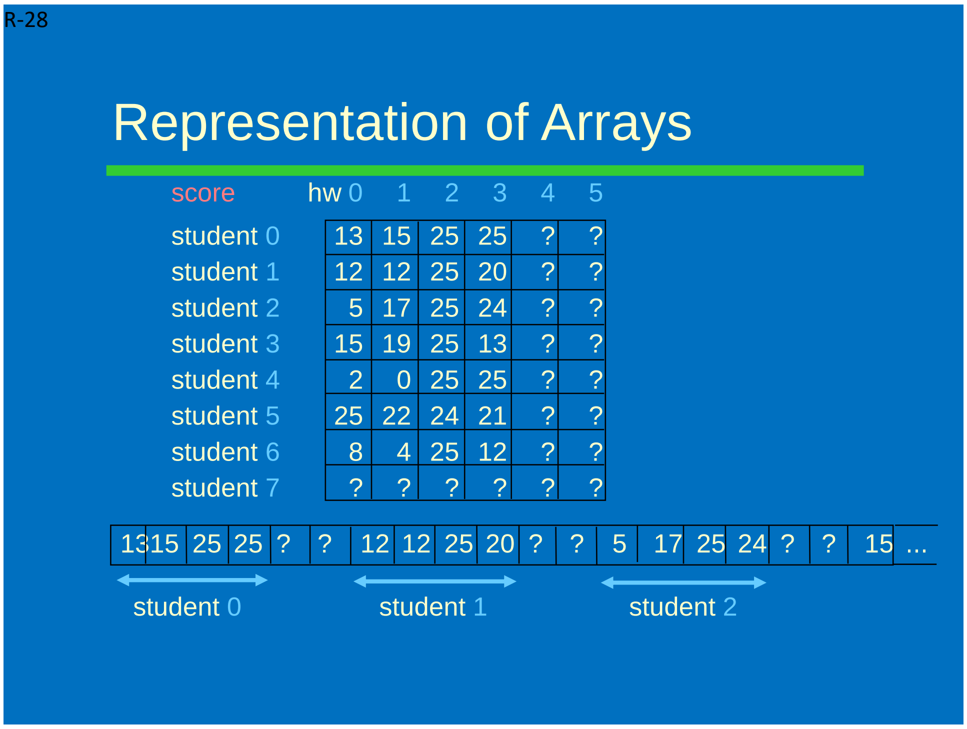
R-27

Representation of Arrays

score	hw 0	1	2	3	4	5
student 0	13	15	25	25	?	?
student 1	12	12	25	20	?	?
student 2	5	17	25	24	?	?
student 3	15	19	25	13	?	?
student 4	2	0	25	25	?	?
student 5	25	22	24	21	?	?
student 6	8	4	25	12	?	?
student 7	?	?	?	?	?	?

13	15	25	25	?	?	12	12	25	20	?	?	5	17	25	24	?	?	15	...

student 0 student 1



R-28

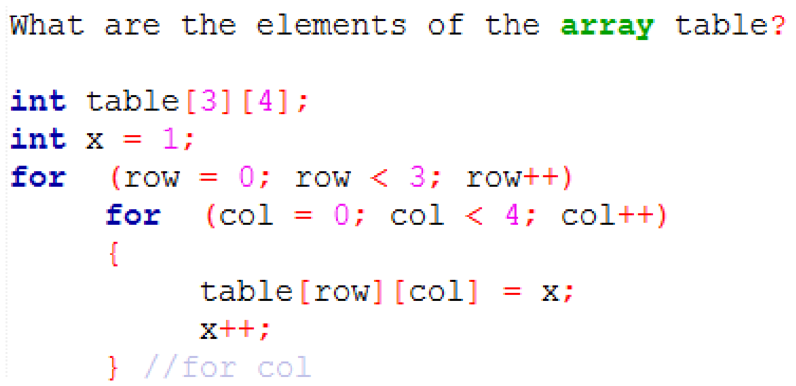
Representation of Arrays

score	hw 0	1	2	3	4	5
student 0	13	15	25	25	?	?
student 1	12	12	25	20	?	?
student 2	5	17	25	24	?	?
student 3	15	19	25	13	?	?
student 4	2	0	25	25	?	?
student 5	25	22	24	21	?	?
student 6	8	4	25	12	?	?
student 7	?	?	?	?	?	?

13	15	25	25	?	?	12	12	25	20	?	?	5	17	25	24	?	?	15	...

student 0 student 1 student 2

Example : Fill Array



```
What are the elements of the array table?

int table[3][4];
int x = 1;
for  (row = 0; row < 3; row++)
     for  (col = 0; col < 4; col++)
     {
          table[row][col] = x;
          x++;
     } //for col
```

O-30

Array Elements as Parameters

Individual array elements can be used as parameters, *just like other simple variables*.
Examples:

printf(“Last two are %f, %f”, rain[5], rain[6]) ;

draw_house(color[i], x[i], y[i], windows[i]) ;

scanf(“%lf”, &rain[0]) ;

swap(&rain[i], &rain[i+1]) ;

O-31

Whole Arrays as Parameters

Array parameters (entire arrays) work differently:

- An array is never copied (no call by value)
- The array name is *always treated as a pointer parameter*
- The & and * operators are not used

Programming issue: in C, arrays do not contain information about their size, so the size often needs to be passed as an additional parameter.

O-32

Array Parameter Example

```
#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 ) ;
```

O-33

General Vector Sum

Usually the size is omitted in an array parameter declaration.

```
/* sum the vectors of the given length */
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] ;
}

// in the main program:
-------------------------------
int x[3] = {1,2,3},  y[3] = {4,5,6},  z[3] ;
VectorSum( x , y , z , 3 );
```

Write a program that adds two 2x2 arrays and stores the sum in third array.

```
#include <stdio.h>
#include <stdlib.h>
#define rows 2
#define cols 2
void Fill(int [][cols]);
void Sum(int  [][cols],int [][cols],int [][cols]);
void Print(int [][cols]);
int main()
{
   int array_a[rows][cols];
   int array_b[rows][cols];
   int result[rows][cols];
   printf ("array 1: \n");
   Fill(array_a);
   printf ("array 2: \n");
   Fill(array_b);
   Sum(array_a,array_b,result);
   printf ("result: \n");
   Print(result);
   return 0;
}
```

```
void Fill(int array[][cols]) {
   int i,j;
   for (i=0;i<rows;i++)    {
      for (j=0;j<cols;j++)       {
        printf("array[%d][%d]:  ",i,j);
        scanf("%d",&array[i][j]);
      } }
   printf ("\n\n\n");
}
```

```
void Sum(int array_a[][cols],int array_b[][cols],int result[][cols])
{
   int i,j;
   for (i=0;i<rows;i++)    {
      for (j=0;j<cols;j++)       {
         result[i][j]=array_a[i][j]+array_b[i][j];
      }    }  }
```

```
void Print(int array[][cols]) {
   int i,j;
   for (i=0;i<rows;i++)    {
      printf ("\n");
      for (j=0;j<cols;j++)       {
        printf("%d   ",array[i][j]);
      }    }
   printf ("\n\n\n");
}
```

Fill and print array using function & reverse

```
#include <stdio.h>
#define size 5 //  array size= 5
void fillArray (int[],int);
void printArray (int[],int);
void printArrayInreverse(int[],int);
int main ()    {
   int n[ size ];
   printf("Fill Array\n-----------\n");
   fillArray(n,size);
   printf("Print Array\n-----------\n");
   printArray(n,size);
   printf("\nReverse Array\n------\n");
   printArrayInreverse(n,size);
   return 0;
}
void fillArray (int myArray[],int s) {
   int i;
   for (i=0;i<s;i++)
   {
      printf ("myArray[%d]= ",i);
      scanf("%d",&myArray[i]);
      printf("\n");
   }
```

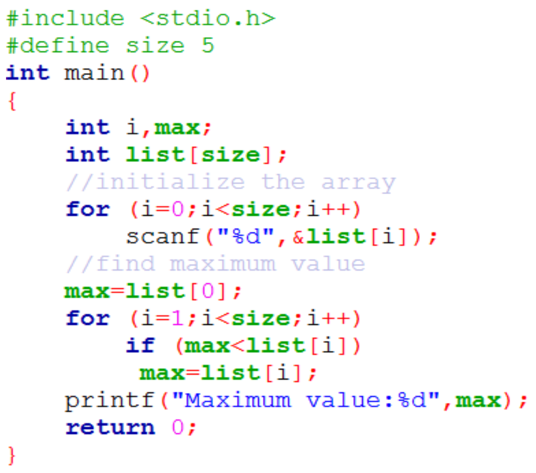
```
void printArray (int myArray[],int s)
{
    int i;
    for (i=0;i<s;i++){
        printf ("myArray[%d]= ",i);
        printf("%d",myArray[i]);
        printf("\n");

    }

}
```

```
void printArrayInreverse(int myArray[],int s)
{  int i;
   for (i=s-1;i>=0;i--){
      printf ("myArray[%d]= ",i);
      printf("%d",myArray[i]);
      printf("\n");
   }
}
```

Example: Finding the Maximum



```
#include <stdio.h>
#define size 5
int main()
{
    int i,max;
    int list[size];
    //initialize the array
    for (i=0;i<size;i++)
        scanf("%d",&list[i]);
    //find maximum value
    max=list[0];
    for (i=1;i<size;i++)
        if (max<list[i])
         max=list[i];
    printf("Maximum value:%d",max);
    return 0;
}
```

Strings

A **string** is a sequence of characters treated as a group

We have already used some string literals:

- “filename”
- “output string”

Strings are important in many programming contexts:

- names
- other objects (numbers, identifiers, etc.)

Strings in C

No explicit type, instead strings are maintained as arrays of characters

Representing strings in C

- stored in arrays of characters
- array can be of any length
- end of string is indicated by a *delimiter*, the zero character ‘\0’

"A String"

A		S	t	r	i	n	g	\0

String Literals

String literal values are represented by *sequences of characters between double quotes* (“)

Examples

“” - empty string

“hello”

“a” versus ‘a’

‘a’ is a single character value (stored in 1 byte) as the ASCII value for a

“a” is an array with two characters, the first is a, the second is the character value \0

Referring to String Literals

String literal is an array, can refer to a single character from the literal as a character

Example:

printf(“%c”,”hello”[1]); <u>outputs</u> the character ‘e’

During compilation, C creates space for each string literal (# of characters in the literal + 1)

Duplicate String Literals

Each string literal in a C program is stored at a *different location*
So even if the string literals contain the same string, they are not equal (in the == sense)
Example:

```
char string1[6] = “hello”;
char string2[6] = “hello”;
```

but *string1 does not equal string2* (they are stored at different locations)

String Variables – Declaration

Allocate an array of a size large enough to hold the string (*plus 1 extra value for the delimiter*)
Examples (with initialization):

```
char str1[6] = “Hello”;
char str2[] = “Hello”;
char *str3 = “Hello”;
char str4[6] = {‘H’,’e’,’l’,’l’,’o’,’\0’};
```

Note, each variable is considered a constant in that the space it is connected to cannot be changed

```
str1 = str2; /* not allowable, but we can copy the contents
                of str2 to str1 (more later) */
```

Changing String Variables

Cannot change **space** string variables connected to, but *can use pointer variables that can be changed*

Example:

```
char *str1 = “hello”;   /* str1 unchangeable */
char *str2 = “goodbye”; /* str2 unchangeable */
char *str3; /* Not tied to space */
str3 = str1; /* str3 points to same space s1 connected to */
str3 = str2;
```

Changing String Variables (cont)

Can change parts of a string variable

```
char str1[6] = “hello”;
str1[0] = ‘y’;
/* str1 is now “yello” */
str1[4] = ‘\0’;
/* str1 is now “yell” */
```

Important: to retain delimiter (replacing str1[5] in the original string with something other than ‘\0’ *makes a string that does not end*)

Have to stay within limits of array

Linear and Binary Search

Linear Search

Problem

Given a list of N values, determine whether a given value X occurs in the list.

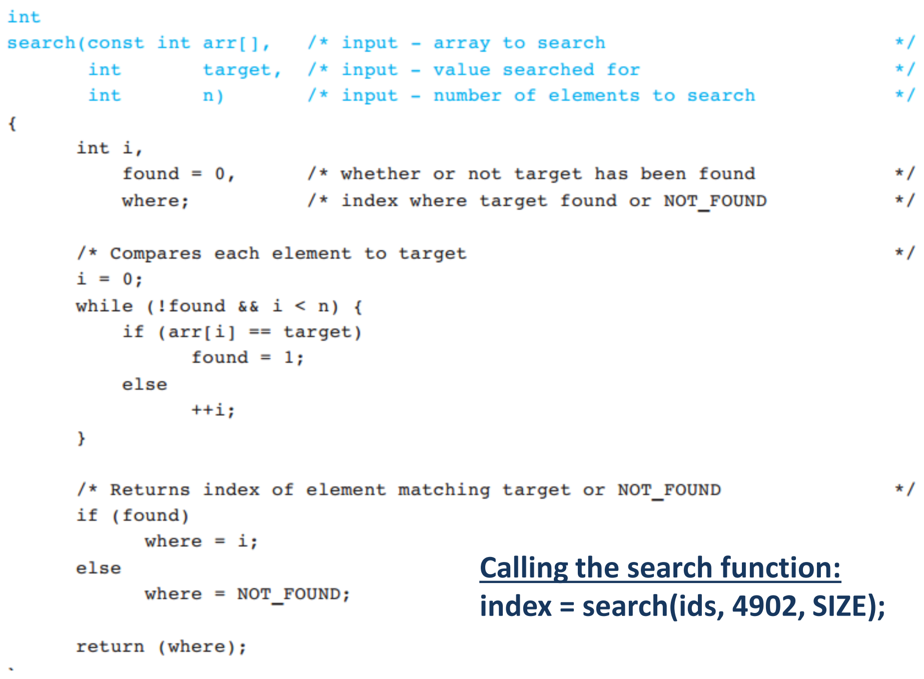
Example: consider the problem of determining whether the value 55 occurs in:

1	2	3	4	5	6	7	8
17	31	9	73	55	12	19	7

ALGORITHM

```
1. Assume the target has not been found.
2. Start with the initial array element.
3. repeat while the target is not found and there are more array elements
       4. if the current element matches the target
              5. Set a flag to indicate that the target has been found.
          else
              6. Advance to the next array element.
7. if the target was found
       8. Return the target index as the search result.
   else
       9. Return -1 as the search result.
```

Code:



```
int
search(const int arr[],   /* input - array to search             */
       int       target,  /* input - value searched for          */
       int       n)       /* input - number of elements to search */
{
      int i,
          found = 0,      /* whether or not target has been found */
          where;          /* index where target found or NOT_FOUND */

      /* Compares each element to target                          */
      i = 0;
      while (!found && i < n) {
          if (arr[i] == target)
                found = 1;
          else
                ++i;
      }

      /* Returns index of element matching target or NOT_FOUND     */
      if (found)
            where = i;
      else
            where = NOT_FOUND;

      return (where);
```

Calling the search function:

index = search(ids, 4902, SIZE);

Linear Search – Example 2

```
#include <stdio.h>
#define size 7
int main() {
 int myArray[size]={209,99,887,01234,987,54,66};
   int target;      // input - value searched for
   int location;    //  index of the target
   int found = 0;
   int i=0;
   printf("please enter a target: ");
   scanf("%d",&target);
    while (i<size)   {
          if (target==myArray[i])  {
              location=i;  //update location
              found=1;    //Matching target
              break;
          }   // end if
          i++;
    }   // end while
```

```
 if (found==1)
       printf("Matching target,
location is %d\n",location);
    else
        printf("Not found\n");
   return 0;
 } // main
```

Sorting: Selection Sort

Example: Sorting in descending order

```
#include <stdio.h>
#include <stdlib.h>
#define Size 3
void Sort (int []);

int main()
{
   int i;
   int array[Size];
   printf("Enter array size %d\n",Size);
   for(i=0;i<Size;i++)
          scanf("%d",&array[i]);
   Sort (array);
   printf("array after sorted :");
   for(i=0;i<Size;i++)
         printf("%d ",array[i]);
   printf("\n");
   return 0;
}
```

```
void Sort(int array[])
{
   int i,j;
   int temp;
   for(i=0;i<Size-1;i++)
   {
      for (j=i+1;j<Size;j++) {
        if (array[i]<array[j]) {
           temp=array[j];
           array[j]=array[i];
           array[i]=temp;
        } // if statement
      }   // inner loop
   }      // outer loop
}         // Sort function
```

```
Enter array of integers
with size 3  3 4 5
array after sorted :5 4 3
```