Lecture **9**

Arrays and Strings

Fundamentals of Computer and Programming

Instructor: Morteza Zakeri, Ph.D. (m-zakeri@live.com)

Spring 2024

Modified Slides from Dr. *Hossein Zeinali* and *Dr. Bahador Bakhshi*

Computer Engineering Department, Amirkabir University of Technology

What We Will Learn

- Introduction
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Array of Strings

What We Will Learn

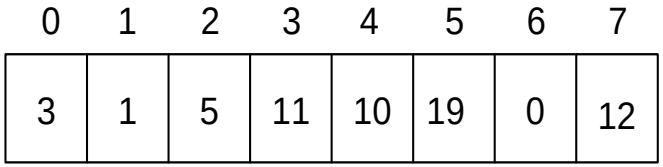
- Introduction
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Array of Strings

Introduction

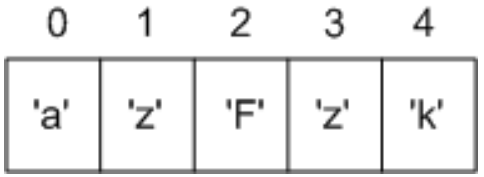
- Algorithms usually work on large data sets
  - Sort a set of numbers
  - Search a specific number in a set of numbers
- How to read and store a set of data?
- To read
  - Repeat the scanf statement
  - Use the loop statements
- To store the data
  - Save each data in a single variable??
    - 3000 int variables! ! ! !

Array

- A collection of same type variables
- A n*x*1 vector of
  - Integers, chars, floats, …
- Example
  - An array of 8 integer



  - An array of 5 chars



Arrays in C

- Array declaration in C

  <Elements' Type> <identifier> [<size>]

- <Elements' Type>: int, char, float, …
- <size>
  - Old compilers (standard): it should be constant
  - New compilers (standard): it can be variable
- Elements in array
  - From 0 to (size – 1)

Example

```
int num[20];
```

- **num** is array of 20 integers
- **num[0]** is the first integer variable
- **num[19]** is the last integer

- Indices are **always integer** while the elements' type can be any type

```
float farr[100];
```

- **farr** is array of 100 floats
- **farr[0]** is the first float
- **farr[49]** is the 50th float
- **farr[99]** is the last float

What We Will Learn

- Introduction
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Array of Strings

Arrays in Functions

```
int number[20];
```

- **`number[i]`** is an integer variable
- Array element can be used for **call by value** input
- Array element can be use for output

```
int f(int x);
void h(void){
    int arr[50];
    //Array element in call by value
    arr[30] = f(arr[5]);
}
```

Arrays in Functions (cont'd)

- Array **cannot** be used as output type of function

```
int [] f(int x, int y); // Compile error
```

- Arrays can be used in input list of functions
- Arrays are not passed by **Call By Value**
- Arrays are passed by **Call By Reference**
  - If we change array elements in a function
    - The element is changed in the caller function

Arrays in Functions (version 1)

- Function **arr_func** takes an array of integers

```
int arr_func(int num[90]){
}
// or
int arr_func(int num[]){
}
```

- Array **a** is passed to function **arr_func**

```
int a[90];
i = arr_func(a);
```

Arrays in Functions (version 1)

```
#include <stdio.h>

void init_array(int arr[10]){
   int i;
   for(i = 0; i < 10; i++)
      arr[i] = i;
}

void main(void){
   int a[10];
   init_array(a);

   int j;
   for(j = 0; j < 10; j++)
      printf("a[%d] = %d\n", j, a[j]);
}
```

تابعی که یک آرایه به طول ۱۰را می‌گیرد و اعضای آن را با اعداد ۰ تا ۹ مقداردهی می‌کند.

Array Size in Functions

- If array is an input parameter of a function
  - It cannot find out the size of the array
- Array size **should be passed** from caller function to called function
  - Using definitions

```
#define SIZE 20
void func(int a[]){ for(int i = 0; i < SIZE; i++)…}
```

  - Using input variable

```
void read(int a[], int size){ for(int i = 0; i < size; i++)
// or
void read(int size, int a[size]){
    for(int i = 0; i < size; i++) …}
```

Array Size in Functions (cont'd)

- If array is declared in a function
  - It knows the size of the array
  - It can find out the size of the array using **sizeof**

```
void func(void){
   int i, a[200];
   for(i = 0; i < 200; i++)
      a[i] = 0;
   // or
   for(i = 0; i < sizeof(a)/sizeof(a[0]); i++)
      a[i] = 0;
}
```

Out-of-range access

- C compiler does not check the range you access
  - `int x[10];  x[20] = 30;  y = x[100];`
  - No compiler error!
- What happen
  - Read: Logical error
    - `y = x[100];`
  - Write: May or may not logical or runtime error
    - `x[20] = 30;`

Out-of-range Example

```
#include <stdio.h>
void init_array(int size, int arr[]){
  int i;
  for(i = 0; i < size; i++)
     arr[i] = i;
}

void print_array(int size, int arr[]){
  int i;
  for(i = 0; i < size; i++)
     printf("arr[%d] = %d\n", i, arr[i]);
}
```

Out-of-range Example (Cont'd)

```
void main(void){
   int x = 1;
   int y = 2;
   int a[10] = {0}; // all elements set to Zero

   init_array(10, a);          // OK
   print_array(10, a);         // OK

   print_array(30, a);         // Wrong output

   init_array(1000, a);        // May be Run-time error!!!

   init_array(20, a);          // May changes X, Y!!!
                               // Logical error
   printf("x = %d, y = %d\n" , x, y);
}
```

Find Array Max index

```
#include <stdio.h>

int max_index(int a[], int size){
    int i;
    int index = 0;
    for(i = 1; i < size; i++)
        if(a[i] > a[index])
            index = i;
    return index;
}

void main(void){
    int arr[] = {1, 4, 12, 93, 23};
    printf("max index = %d\n", max_index(arr, 5));
}
```

تابعی که یک آرایه را بگیرد و محل بزرگترین عضو آن‌را برگرداند.

Array elements swap

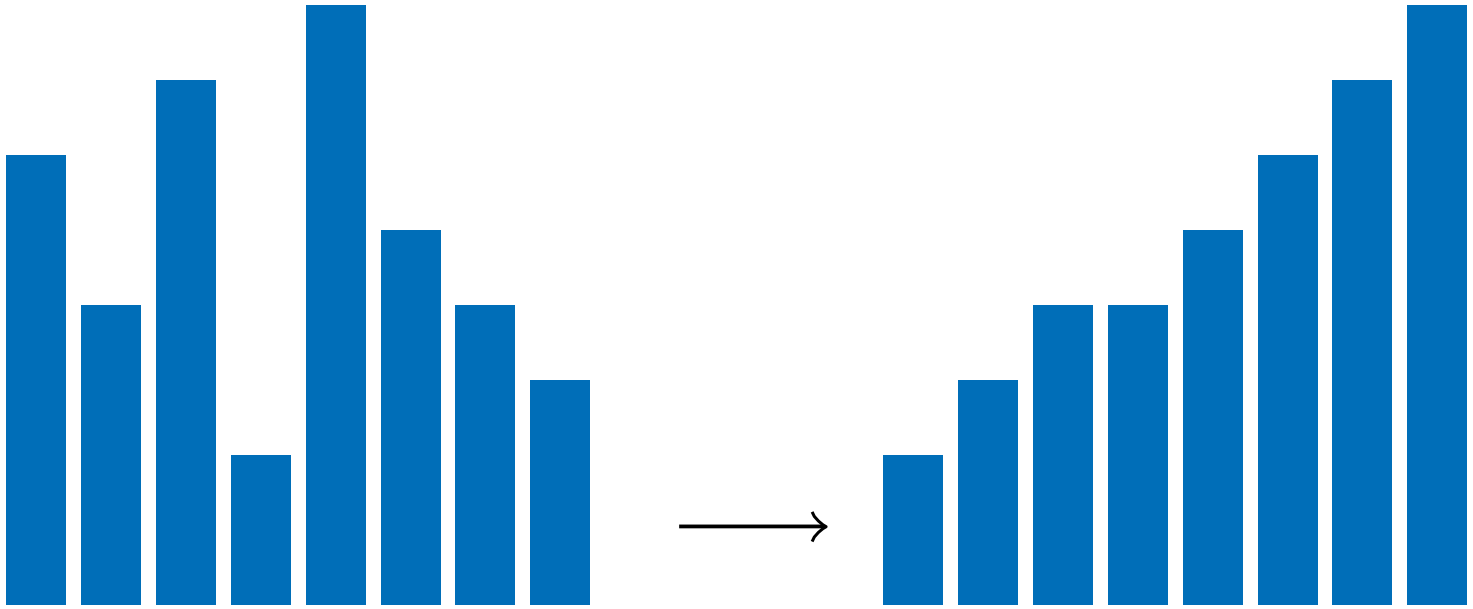
```
#include <stdio.h>

void array_swap(int a[], int i, int j){
   int tmp;
   tmp = a[i];
   a[i] = a[j];
   a[j] = tmp;
}

void main(void){
   int num[10] = {1, 2, 5, 6};
   int x = 2, y = 6;
   printf("num[%d] = %d, num[%d] = %d\n", x, num[x], y,
   num[y]);
   array_swap(num, x, y);
   printf("num[%d] = %d, num[%d] = %d\n", x, num[x], y,
   num[y]);
}
```

تابعی که یک آرایه و دو محل آن‌را بگیرد و آنها را باهم جابجا کند.

Sorting Problem



Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Selection Sort

Algorithm:

- Find the **smallest** element in the array
- Exchange it with the element in the first position
- Find the second smallest element and exchange it with the element in the second position
- Continue until the array is sorted.

الگوريتمي كه يك رشته عدد را كه محل عضو اول آن با start و محل عضو آخر آن با end مشخص شده است را به صورت صعودي مرتب كند.

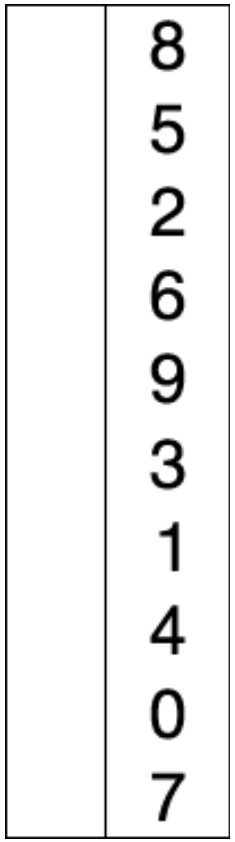
Algorithm: sort (**x, start, end**)

```
while (start != end)
    j ← find index of minimum element from start to end
    swap x[j] and x[start]
    start ← start + 1
```

==========

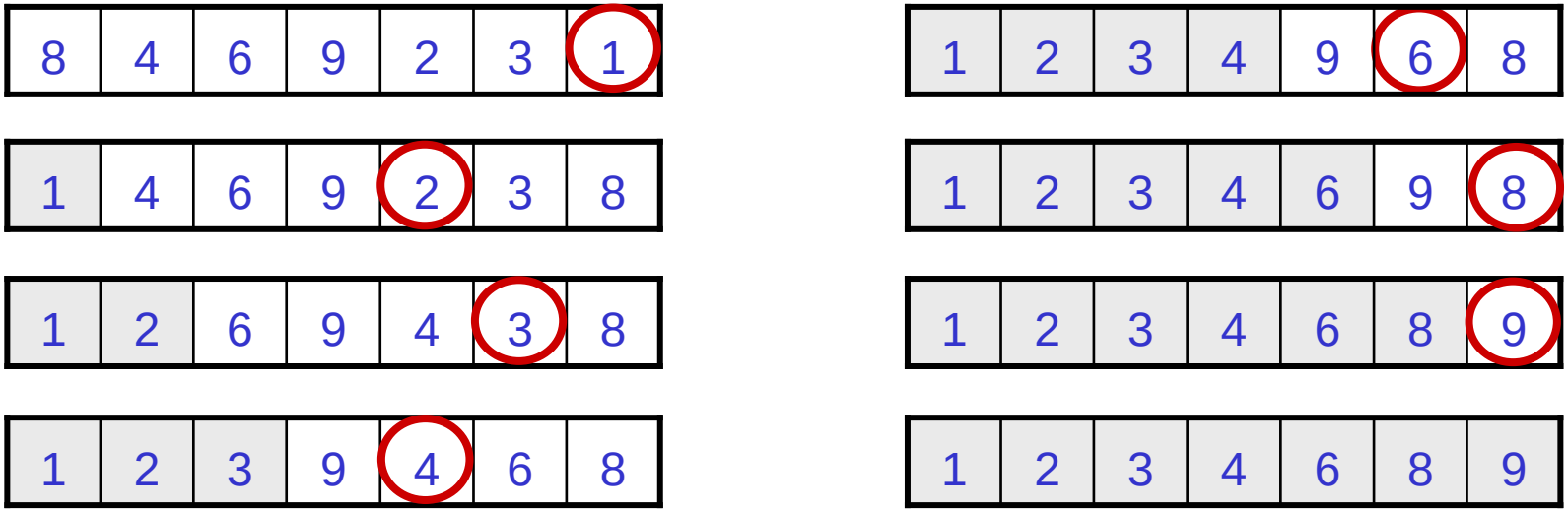
Algorithm find_min(**x, start, end**)

```
y ← start
i ← start + 1
while (i <= end)
    if(x[i] < x[y])
        y ← i
    i ← i + 1
return y
```



Verify the Algorithm

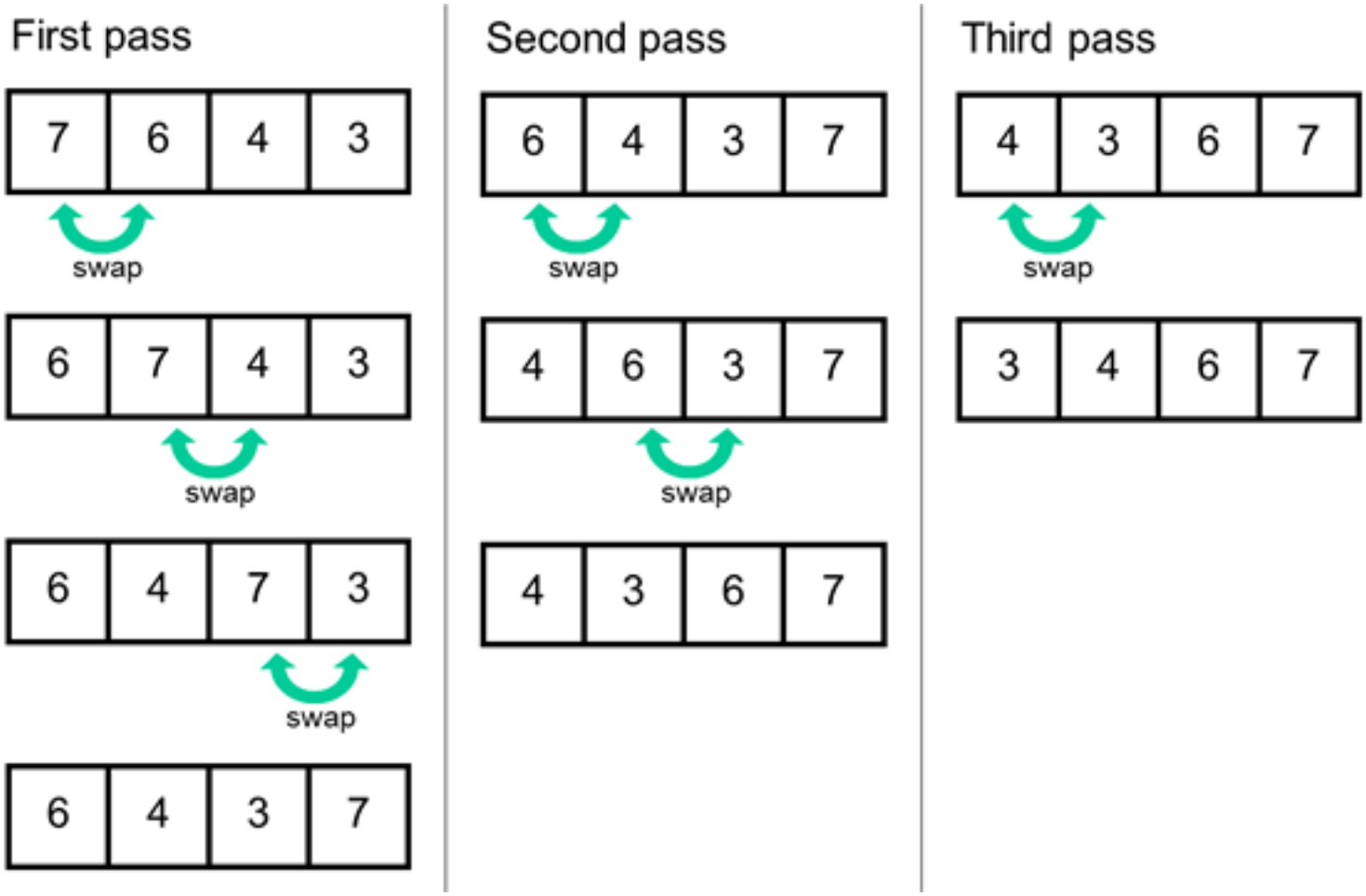
Selection Sort: Example



Selection Sort

```
void selection_sort(int arr[], int n) {
    int i, j, min_idx;
    // One by one move boundary of unsorted subarray
    for (i = 0; i < n - 1; i++) {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }
        // Swap the found minimum element with the first element
        if (min_idx != i)
            swap(arr[min_idx], arr[i]);
    }
}
```

Bubble sort


First pass
7 6 4 3
swap
6 7 4 3
swap
6 4 7 3
swap
6 4 3 7
Second pass
6 4 3 7
swap
4 6 3 7
swap
4 3 6 7
Third pass
4 3 6 7
swap
3 4 6 7

Sorting an array: Bubble sort

```
#include <stdio.h>
void array_swap(int a[], int i, int j){
    int tmp;
    tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}
```

تابع مرتب‌سازی مجموعه اعداد صحیح

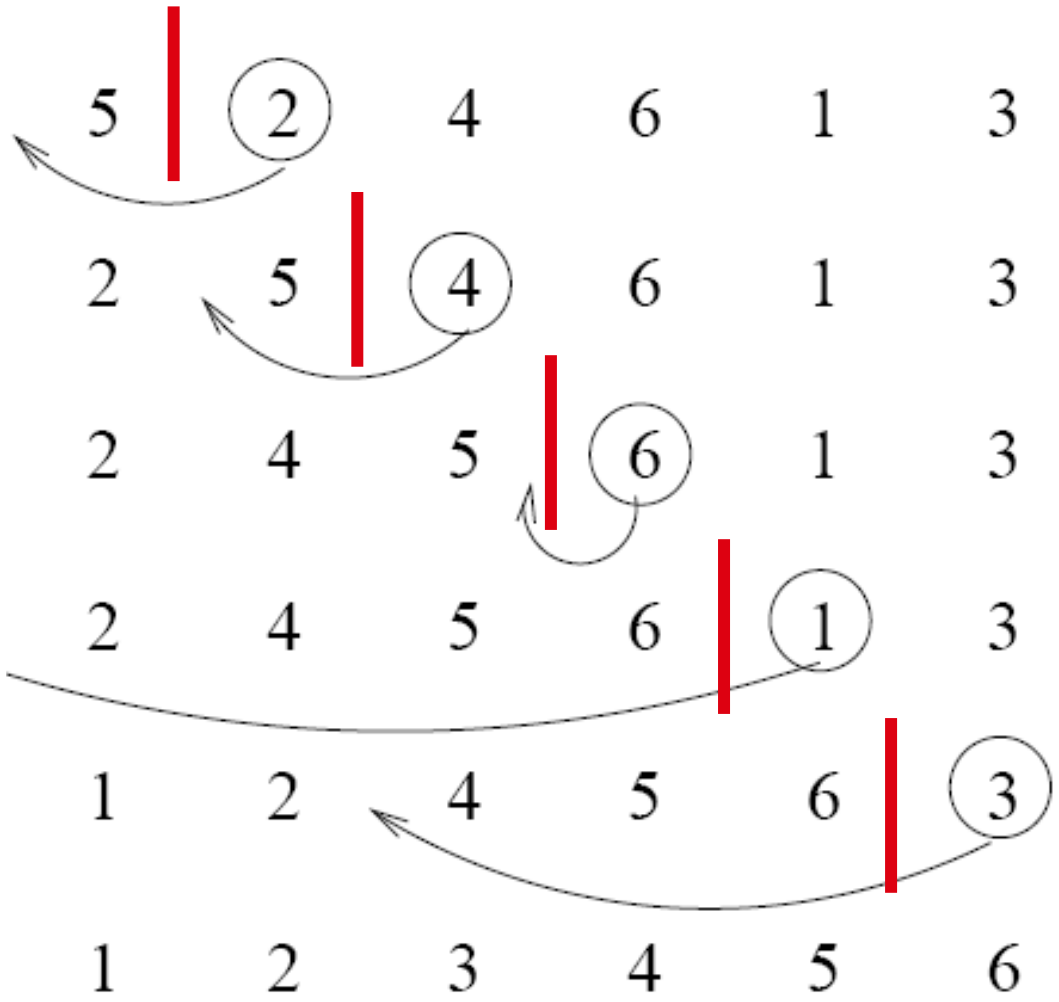
```
void bubble_sort(int a[], int size){
  int i, j;
  for(i = 0; i < size - 1; i++)
   for(j = i + 1; j < size; j++)
      if(a[i] < a[j])
         array_swap(a, i, j);
}
```

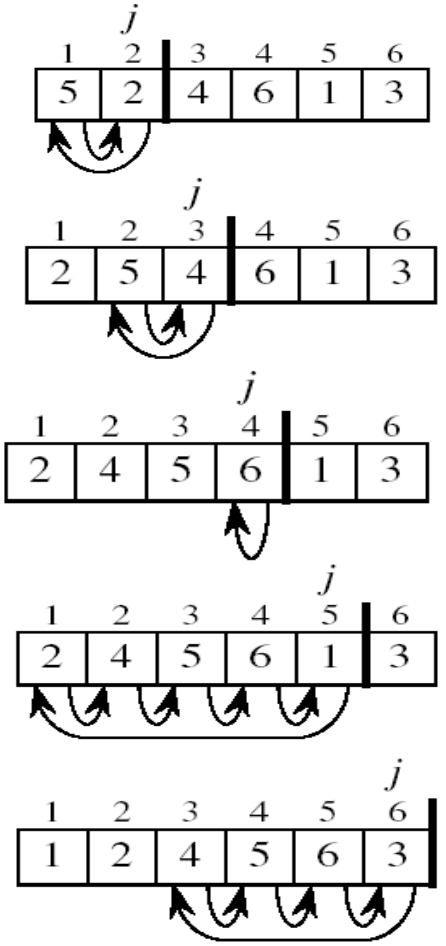
Sorting an array: Bubble sort (Cont'd)

```
void print(int a[], int size){
     int i;
     for(i = 0; i < size; i++)
          printf("%d ", a[i]);
     printf("\n");
}

int main(void){
    int arr[] = {1, 7, 3, 7, 11, 0};
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Before sort: ");
    print(arr, size);
    bubble_sort(arr, size);   // Call print in bubble_sort to show
    printf("After sort: ");   //progress
    print(arr, size);
     return 0;
}
```


Exercise: Insertion Sort





Quiz: Insertion Sort

Write the code of insertion sort as a C function.

Quiz: Insertion Sort (Answer)

```
void insertion_sort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
   /* Move elements of arr[0..i-1], that are greater than key, to one
position ahead of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Binary Search

```
int bsearch(int start, int end, int a[], int value){
  int mid = (start + end) / 2;
  if(a[mid] == value)
     return mid;
  else if(start >= end)
     return -1;
  else if(value > a[mid])
     return bsearch(mid + 1, end, a, value);
  else
     return bsearch(start, mid - 1 , a, value);
}
```

What We Will Learn

- Introduction
- Initializing arrays
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Array of Strings

Multidimensional Arrays

- If element of an array is array itself, it will be **Multidimensional array**
- n*x*n matrix, m*x*n*x*n*x*m matrix

```
int t[10][20];
```

- 10x20 matrix of integers

```
t[1][1];   // t[1,1] → compile error
```

- Integer variable in location (1,1)

Initializing Multidimensional Arrays

```
int num[2][3] = {1, 2, 0, 3, 4 , 7};
int num[2][3] = {{1, 2, 0},{3, 4, 7}};
```

- num[0][2] is 0, num[1][0] is 3

```
int num[5][3] = {{1, 2, 0},{3, 4, 7}};
```

- num[2][2] is 0, num[1][2] is 7

```
int num[2][3][2] = {{{1,2},{3,4},{5,6}},
  {{1},{2},{3}}};
```

- num[0][2][1] is 6, num[1][0][1] is 0

```
int num[][2] = {{1,1},{2,2},{3,3}};
```

- num[1][1] is 2, num[2][0] is 3

Multidimensional Arrays in Functions

- Can be used as input of functions
  - All dimensions except the **first** one must be given

```
void func(int a[10][20][5]);
```

  - Input is a 10x20x5 integer matrix

```
void func(int a[][20][30], int size);
```

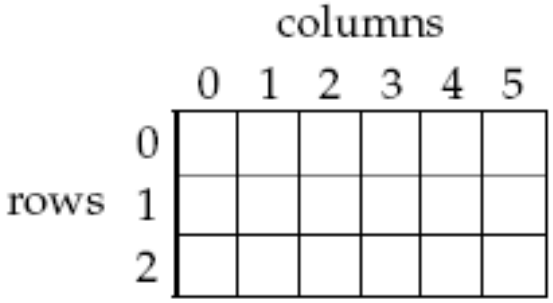
```
void func(int size1, int size2, int
  a[size1][size2]);
```

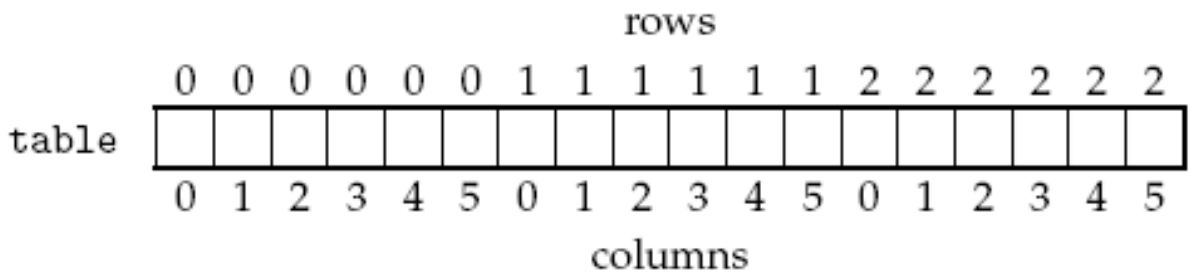
  - Input is a matrix of integers that both rows and columns are variable

Multidimensional arrays memory layout

```
#define ROWS 3
#define COLS 6

int table[ROWS][COLS];
```





$$\&\texttt{table[i][j]} = \&\texttt{table[0][0]} + \texttt{sizeof(int)}(\texttt{i} \times \texttt{COLS} + \texttt{j})$$

Print a matrix

تابعی برای چاپ یک ماتریس که ابعاد آن ثابت است.

```
#include <stdio.h>
void displayMatrix (int nRows, int nCols, int
   matrix[nRows][nCols]){
    int row, column;
    for ( row = 0; row < nRows; ++row) {
      for ( column = 0; column < nCols; ++column )
        printf ("%5i", matrix[row][column]);
      printf ("\n");
    }
}

int main (void){
  int sampleMatrix[3][5] = {{ 7, 16, 55, 13, 12 }, { 12, 10,
   52, 0, 7 }, { -2, 1, 2, 4, 9 }};
   printf ("Original matrix:\n");
   displayMatrix (3, 5, sampleMatrix);
}
```

Transpose matrix (A^T)

محاسبه ترانهاده ماتریس

```
#define SIZE 5

void swap(int a[SIZE][SIZE], int i, int j){
   int tmp;
   tmp = a[i][j];
   a[i][j] = a[j][i];
   a[j][i] = tmp;
}

void transpose(int a[][SIZE]){
   int i, j;
   for(i = 0; i < SIZE; i++)
      for(j = i; j < SIZE; j++)
          swap(a, i, j);
}
```

Matrix multiplication

ضرب ماتریس‌ها

Math recap:

Matrix 1

1	1	1
2	2	2
3	3	3

Matrix 2

1	1	1
2	2	2
3	3	3

Matrix 1 * Matrix 2

1*1+1*2+1*3	1*1+1*2+1*3	1*1+1*2+1*3
2*1+2*2+2*3	2*1+2*2+2*3	2*1+2*2+2*3
3*1+3*2+3*3	3*1+3*2+3*3	3*1+3*2+3*3

Matrix 1 * Matrix 2

6	6	6
12	12	12
18	18	18

Matrix multiplication

ضرب ماتریس‌ها

```
#include <stdio.h>
#include <stdlib.h>
// matrix dimensions so that we dont have to pass them as
// parametersmat1[R1][C1] and mat2[R2][C2]
#define R1 2 // number of rows in Matrix-1
#define C1 2 // number of columns in Matrix-1
#define R2 2 // number of rows in Matrix-2
#define C2 3 // number of columns in Matrix-2
void multiplyMatrix(int m1[][C1], int m2[][C2]){
    int result[R1][C2];
    printf("Resultant Matrix is:\n");
    for (int i = 0; i < R1; i++) {
        for (int j = 0; j < C2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < R2; k++) {
                result[i][j] += m1[i][k] * m2[k][j];
            }
            printf("%d\t", result[i][j]);
        }
        printf("\n");   }
}
```

Exercise: Matrix calculation

Write programs for other operations on matrix:

- Add,
- Sub,
- Determinant,
- Eigenvalues,
- etc.

What We Will Learn

- Introduction
- Initializing arrays
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Bugs and avoiding them

Introduction

- Until now
  - We have seen strings in **`printf()`**
  - Our old definition: string is a set of chars between “ ”

```
printf("This is a string\n");

printf("This is %s\n", "a string\n");
```

- Strings:
  - An array of chars
  - Terminated by the null char `'\0'`

Strings in C

- Since strings are array

```
char str3[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};

char str1[8] = "program";

char str2[] = "program";

char *str3  = "program"; //we will see later
```

'p'	'r'	'o'	'g'	'r'	'a'	'm'	'\0'

Reading and Writing Strings

- **`printf`** can be used to print strings

```
printf("program");

printf("%s", "program");
```

- **`scanf`** can be used to read strings

```
char str[200];

scanf("%s", str);
```

  - Initial white spaces are ignored
  - Read until space or '\n' (which is replaced by \0)

- We must allocate sufficient size.

Reading and Writing Strings (cont'd)

- **`puts(str)`** is very simple version of **`printf()`**
  - Can only be used to print strings
  - Adds '\n' to end of string

- **`gets(char str[])`** can be used to read strings
- **`gets`** does not ignore the white spaces
  - Read until **`\n`**
- String should be large enough

What We Will Learn

- Introduction
- Initializing arrays
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Bugs and avoiding them

String Library

- Access to string library by

```
#include <string.h>
```

- Many functions to work with strings
  - Find the length of string
  - Compare strings
  - Copy strings
  - Search in strings
  - …

Length of String

- **`strlen(str)`**: Length of string

- From start to first occurrence of the null char

```
char str[] = "This is test";
char str1[10]={'a', 'b', '\0', 'c', '\0'};
strlen(str)  → 12
strlen(str1) → 2
```

Compare Strings

- str1 and str2 are compared as follows
  - Compare char by char from left to right until str1 and str2 has same chars.
  - In the first different char
    - If(char of str1 < char of str2) → str1 < str2
  - If (both string finish) → str1 = str2
- **`strcmp(str1, str2)`**: compare str1 and str2
  - If(str1 == str2) → return 0
  - If(str1 < str2) → return -1
  - If(str1 > str2) → return 1

Compare Strings: Examples

```
char s1[] = "abc";
char s2[] = "abc";
i = strcmp(s1, s2);   //i  = 0

char s3[] = "abc";
char s4[] = "abx";
i = strcmp(s3, s4);   //i  = -1

char s5[] = "axc";
char s6[] = "abc";
i = strcmp(s5, s6);   //i  = 1

char s7[] = "ab";
char s8[] = "abc";
i = strcmp(s7, s8);    //i  = -1

char s9[]  = "abc";
char s10[] = "aBc";
i = strcmp(s9, s10);   //i  = 1
```

Compare Strings

- **`strcmpi(str1, str2)`**
- Compares str1 and str2 similar to strcmp
- But ignores uppercase/lowercase difference

```
char str1[]="ABC", str2[]="abC";

strcmpi(str1, str2) → 0
```

Copy Strings

- Strings should be copied char by char
- **`strcpy(dst_str, src_str)`**: copy the **`src_str`** to the **`dst_str`**
- **`src_str`** is a constant string
- **`dst_str`** should have sufficient size

Copy Strings: Example

```
char str1[] = "Test String";

char str2[20];

strcpy(str2, str1);

printf("%s\n", str2);

printf("%s\n", str1);
```

Test String

Test String

Concatenate Strings

- **`strcat(dst, src)`**: Append the **`src`** string to the end of **`dst`**
- **`src`** is constant string
- **`dst`** should have sufficient space

Concatenate Strings: Example

```
char str1[]  = "String";

char str2[20]= "Test ";


strcat(str2, str1);

printf("%s\n", str2);   // Test String
```


Sized Version of the Functions

- **`strncpy(dst, src, n)`**:
  - copys **`n`** chars from **`src`** to **`dst`**
  - **`If(strlen(src) > n)`**
    - Copies **`n`** chars to **`dst`**
    - Does not add '\0' to end of **`dst`**
  - **`If(strlen(src) < n)`**
    - Copy **`src`** to **`dst`**
    - Add **`n - strlen(src) - 1`** '\0' to end of **`dst`**
      - **`dst`** must be large enough
      - n < size of **`dst`**

Sized Version of Functions

- **`strncmp(str1, str2, n)`**:
  - Compares the first **`x`** chars
  - **`x`** = min{n, strlen(str1)+1, strlen(str2)+1 }

- **`strncat(dst, src, n)`**:
  - Appends the **`x`** chars from **`src`** to **`dst`**
  - **`x`** = min{**`n`**, **`strlen(src)`**}
  - Adds **`'\0'`** to end of **`dst`**
  - **`dst`** must be large enough

Numbers and Strings: number → string

- To convert a number to string

```
char str1[100];

int i = 100;

sprintf(str1, "%d", i);        // str1 = “100”

float f = 10.11;

sprintf(str1, "%0.2f", f);  // str1 = “10.11”
```

- String **str1** should have sufficient size

Numbers and Strings: string → number

- To convert from strings to numbers

```
#include <stdlib.h>

char str1[] = "10";
int i;
i = atoi(str1);              // i = 10
sscanf(str1, "%d", &i);      // i = 10
char str2[] = "20.44";
double f;
f = atof(str2);              // f = 20.44
sscanf(str2, "%lf", &f);     // f = 20.44
```

String as Array

- Strings are **array of chars**
- We work on arrays element by element
- We can work on strings char by char

```
char str1[] = "100000";
str1[2] = '2';
```

- We can pass strings to functions

String as Array: Example

```
#include <stdio.h>
#include <string.h>

void str_n_m_cat(char s1[], char s2[], int n, int m, char
   res[]){
   strncpy(res, s1, n);
   res[n] = '\0';
   strncat(res, s2, m);
  res[n+m] = '\0';
}

void main(void){
   char s1[] = "abcdefgh",  s2[] = "abcdefgh";
   char result[50];

   str_n_m_cat(s1, s2, 6, 6, result);
   str_n_m_cat(s1, s2, 600, 6, result);//Runtime error
}
```

تابعی که دو رشته s1 و s2 و دو عدد n و m را بگیرد و یک رشته تولید کند که شامل n عضو اول s1 و m عضو s2 است.

Array of Strings

- **2** dimensional array, each row is a string

```
char numeri[][8] = {"zero","uno","due","tre","quattro"};
```

numeri

'z'	'e'	'r'	'o'	'\0'	'\0'	'\0'	'\0'
'u'	'n'	'o'	'\0'	'\0'	'\0'	'\0'	'\0'
'd'	'u'	'e'	'\0'	'\0'	'\0'	'\0'	'\0'
't'	'r'	'e'	'\0'	'\0'	'\0'	'\0'	'\0'
'q'	'u'	'a'	't'	't'	'r'	'o'	'\0'

Array of Strings: Example

```
#include <stdio.h>
#include <string.h>
#define MAX_NAME_SIZE 100
void read_data(char names[][MAX_NAME_SIZE], double grades[],
   int size){
      int i;
      for(i = 0; i < size; i++){
          printf("Enter name: ");
          scanf("%s", names[i]);
          printf("Enter grade: ");
          scanf("%lf", &(grades[i]));
      }
}
double get_average(double grades[], int size){
      int i;
      double res = 0;
      for(i = 0; i < size; i++)
           res += grades[i];
      return (res / size);
}
```

برنامه‌ای که تعداد دانشجویان را بگیرد، سپس اسم هر دانشجو و نمره را بگیرد.

اسم دانشجویانی که نمره آنها بیشتر از میانگین است، را چاپ کند.

Array of Strings: Example (Cont'd)

```
void  print_names(char  names[][MAX_NAME_SIZE],  double
  grades[], int size, double average){
     int i;
     printf("Average = %lf \n", average);
     printf("List of students whose grade is above the
  average: \n");

     for(i = 0; i < size; i++)
         if(grades[i] > average)
             printf("%s\n", names[i]);
}
```

Array of Strings: Example (Cont'd)

```
int main(void){
    int num;
    printf("Enter number of students: ");
    scanf("%d", &num);
    double grades[num];
    char names[num][MAX_NAME_SIZE];
    read_data(names, grades, num);
    double average = get_average(grades, num);
    print_names(names, grades, num, average);

    return 0;
}
```

ctype.h

- Many function to work on chars
  - Check digit
  - Check alphabetic
  - Check lower or upper case
  - Convert from/to upper/lower case

ctype.h

- **`int isdigit(ch)`**
  - Check ch is digital char or not
- **`int isalpha(ch)`**
  - Check ch is alphabetic or not
- **`int islower(ch)`**
  - Check ch is lowercase alphabetic or not
- **`int isupper(ch)`**
  - Check ch is uppercase alphabetic or not
- **`char tolower(ch)`**
  - Convert ch to lowercase and return it
- **`char toupper(ch)`**
  - Convert ch to upper case and return it

What We Will Learn

- Introduction
- Initializing arrays
- Arrays in functions
- Multidimensional arrays
- String
- String functions
- Bugs and avoiding them

Common Bugs and Avoiding them

- Strings which are used as destination
  - scanf, sprintf, strcpy, ….
  - Must be large enough

Take care about the '\0'

- You should never destroy it, some library functions do!
- **Out of range array index!!!** (read/write, wrong size in function, multidimensional array memory)
- You cannot assign a value to array

```
int a[4], b[4];  a = b; // Error
```

- To debug
  - Print the array index and corresponding value

Reference

- Reading Assignment: Chapters 6 and 8 of "C How to Program"