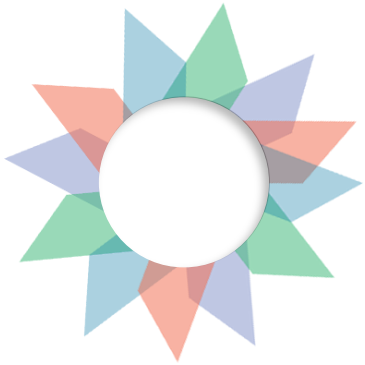
UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

ALGORITHMS & DATA STRUCTURES

01 - Array & Pointer

Farica Perdana Putri



Learning Outcomes

Students are able to understand the concepts and methods of applying arrays and pointers.

Outline

1. Concept of Data Structures
2. Array Declaration
3. Operations on Array
4. Types of Array

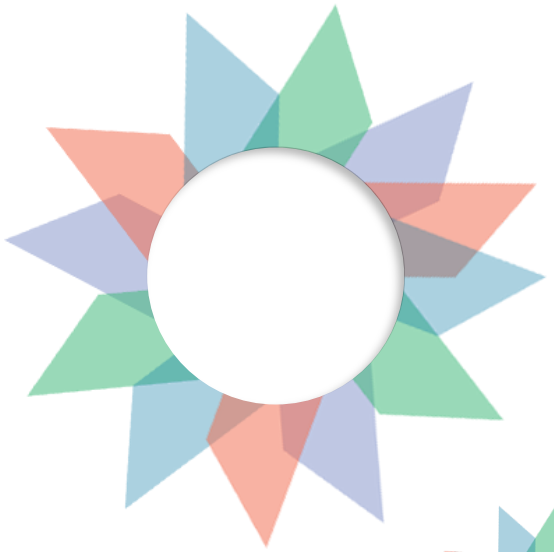
Outline

5 Pointer Initialization

6 Pointer Operator

7 Relationships between Array and Pointer

8 Strings and Characters



Concept of Data Structures

- Definition
- Classification
- Operations

Definition of Data Structures

A data structure is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.

Classification of Data Structures

01

Primitive Data Structures

The fundamental data types which a re supported by a programming lan guage. Some basic data types are in teger, real, character, and Boolean.

02

Non-Primitive Data Structures

Data structures which are created using p rimitive data structures. Examples of such data structures include linked lists, stacks, trees, and graphs. Non-primitives divided into 2 categories: linear and non-linear.

03

Linear Structures

A data structure are stored in a linea r or sequential order. Examples inclu de arrays, linked lists, stacks, and qu eues.

04

Non-Linear Structures

Examples include trees and graphs.

Operations on Data Structures

Traversing

Access each data item exactly once.

Searching

Find the location of one or more data items that satisfy the given constraint.

Merging

Lists of two sorted data items can be combined to form a si ngle list of sorted data items.

Inserting

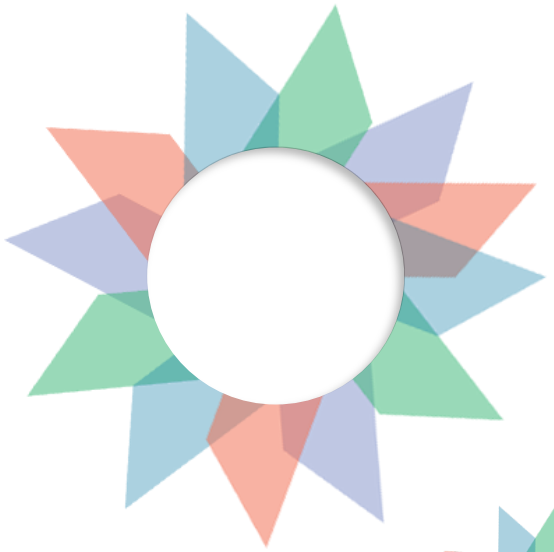
Add new data items to the given list of data items.

Sorting

Data items can be arranged in some order like ascending order or descending order.

Deleting

Remove (delete) a particu lar data item.



Array

- Definition
- Declaration
- Initialization
- Operations
- Types

Definition of Array

Arrays are data structures consisting of related

data items of the same type under the same name.

An array is a group of contiguous memory locations that all have the same type.

Definition of Array

- To refer to a particular location or element in the array, we specify the name of the array and the position number of the particular element in the array.

- The first element in every array is the zeroth (0^{th}) element.

- The position number contained within square brackets is more formally called a subscript or index (must be an integer or integer expression).

- The array name is a symbolic reference to the address of the first byte of the array.

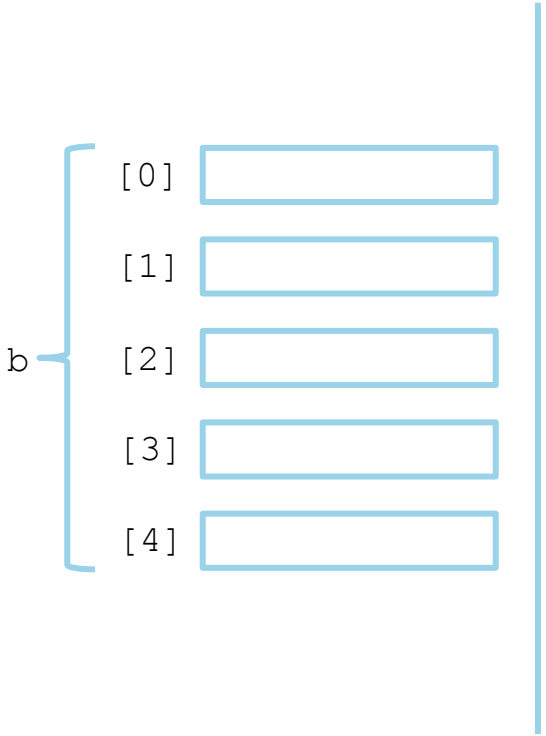
Array Declaration

- Syntax

```
element_data_type array_name[size];
```

- Example

```
int a[100];
int b[5];
```



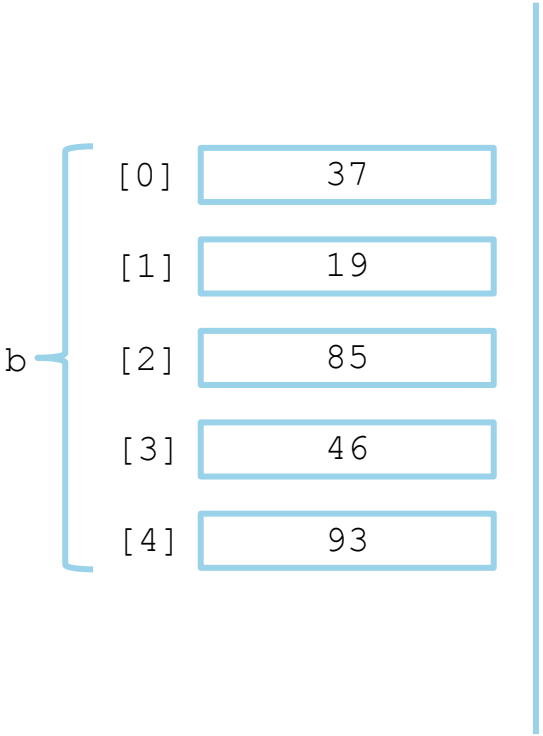
Array Initialization

- Using for statements

```
int a[100], i;

for(i = 0;i < 100;i++)
{
   a[i] = 0;
}
```

- Using an initializer list

```
int b[5] = {37,19,85,46,93};
```



Array Initialization

- If there are fewer initializers than elements in the array, the remaining elements are initialized to zero.

```
int a[100] = {0};
```

  - This explicitly initializes the first element to zero and initializes the remaining 99 elements to zero.

- If the array size is omitted from a definition with an initializer list, the number of elements in the array will be the number of elements in the initializer list.

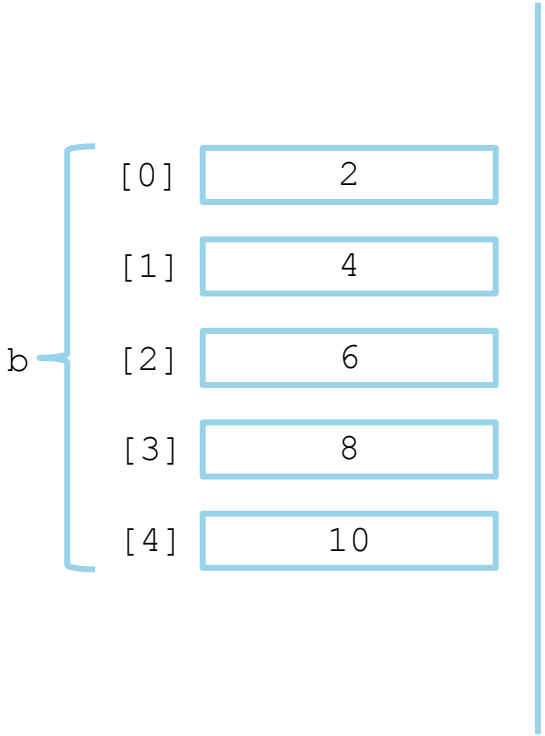
```
int b[] = {37,19,85,46,93};
```

  - This would create a five-element array.

Array Initialization

- Using symbolic constant and calculations

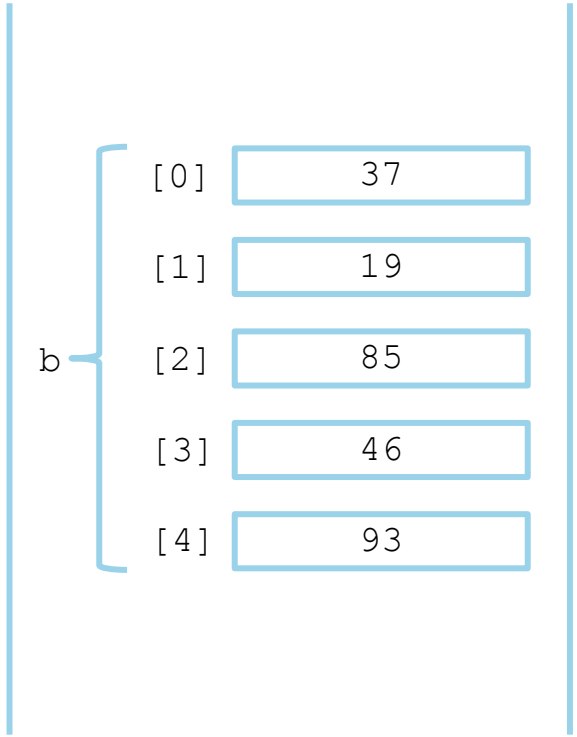
```
#define SIZE 5

int b[SIZE];
int i;

for(i = 0;i < SIZE;i++)
{
   b[i] = 2 + 2 * i;
}
```



Operations on Array

- Traversing: accessing each and every element of the array for a specific purpose.

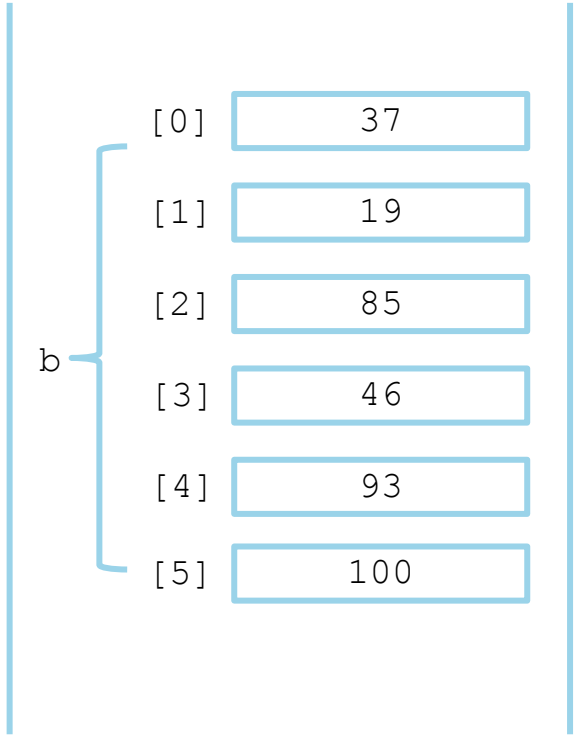
```
int i;

for(i = 0;i < 5;i++)
{
   printf("%d\n",b[i]);
}
```



Operations on Array

- Inserting

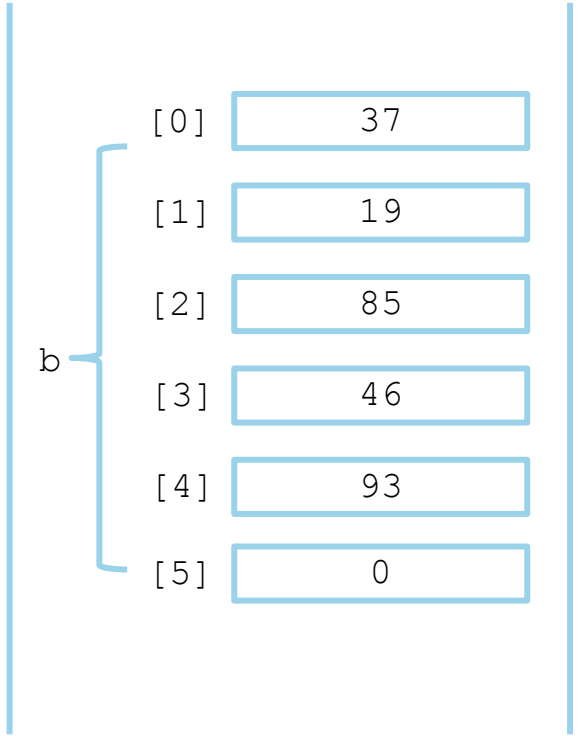
```
int i;
int b[6] = {37,19,85,46,93};

b[5] = 100;
```



Operations on Array

- Deleting

```
int i;
int b[6] = {37,19,85,46,93};

b[5] = 0;
```



Operations on Array

- Merging: merging two arrays in a third array

```
int i,j;
int a[5] = {37,19,85,46,93};
int b[5] = {7,1,8,6,9};
int c[10];

j = 0;
for(i=0; i<10; i++)
{
    if(i<5)
    {
        c[i] = a[j++];
    }
    else c[i] = b[i-j];
}
```

Operations on Array

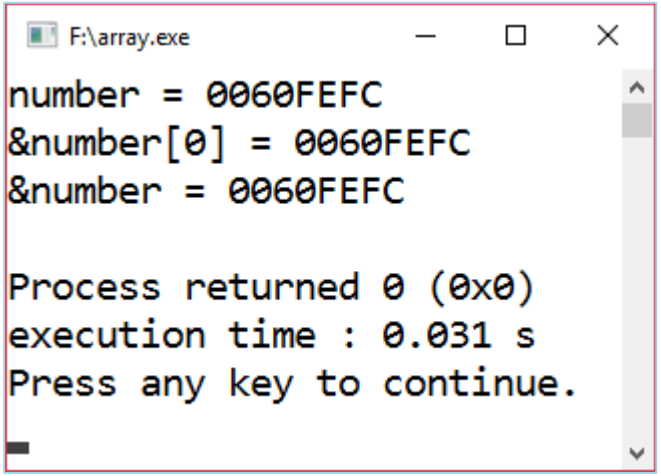
- Passing Arrays to Functions
  - C automatically passes arrays to functions by reference
    - The called functions can modify the element values in the callers' original arrays
  - The name of the array evaluates to the address of the first element of the array
    - **Array name is the same as the address of the array's first element**

```
array_name = &array_name[0]
```

Operations on Array

- Passing Arrays to Functions

```c
#include <stdio.h>

int main()
{
    int number[5];

    printf("number = %p\n",number);
    printf("&number[0] = %p\n",&number[0]);
    printf("&number = %p\n",&number);

    return 0;
}
```

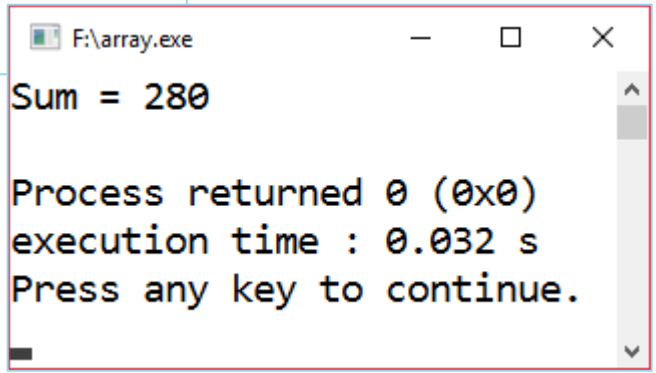


Operations on Array

- Passing Arrays to Functions

```c
int sum(int n[])
{
    int result = 0, i;

    for(i = 0;i < 5;i++)
    {
        result += n[i];
    }

    return result;
}
```

```c
int main()
{
    int total;
    int number[5] = {37, 19, 85, 46, 93};

    total = sum(number);
    printf("Sum = %d\n",total);

    return 0;
}
```



Types of Array

- 2-Dimensional
  - Two-dimensional array is used to represent tables of data (tables of values consisting of information arranged in rows and columns), matrices, and other two-dimensional objects.

	Column 0	Column 1	Column 2	Column 3
Row 0	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]
Row 1	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]
Row 2	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]

Types of Array

- Multi Dimensional
  - Declaration

```
element_data_type array_name[size1][size2]...[sizen];
```

  - As a function parameter

```
element_data_type array_name[size1][size2]...[sizen];
```

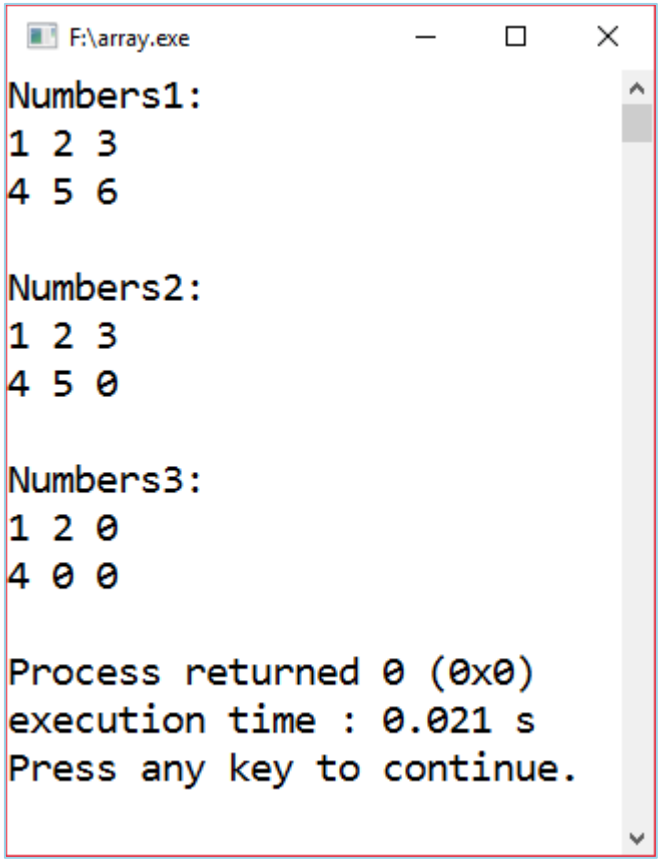
```
element_data_type array_name[][size2]...[sizen];
```

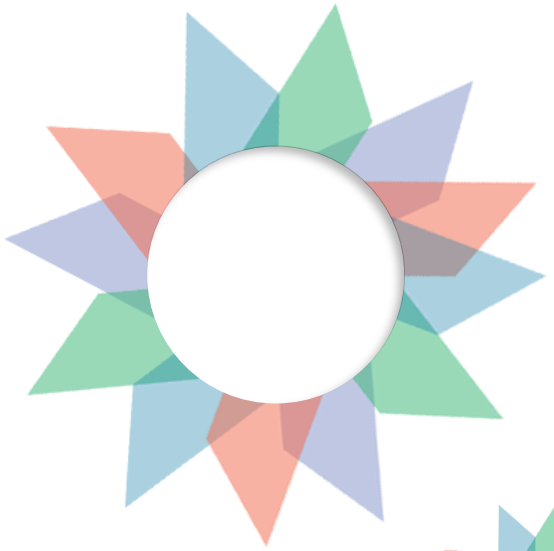
Types of Array

```c
void printArray(int arr[][3])
{
    int iRow, iCol;

    for(iRow = 0;iRow <= 1;iRow++)
    {
        for(iCol = 0;iCol <= 2;iCol++)
        {
            printf("%d ",arr[iRow][iCol]);
        }
        printf("\n");
    }
}
```

```c
int main()
{
    int numbers1[2][3] = {{1, 2, 3}, {4, 5, 6}};
    int numbers2[2][3] = {1, 2, 3, 4, 5};
    int numbers3[2][3] = {{1, 2}, {4}};

    printf("Numbers1:\n");
    printArray(numbers1);

    printf("\nNumbers2:\n");
    printArray(numbers2);

    printf("\nNumbers3:\n");
    printArray(numbers3);

    return 0;
}
```

Types of Array





Pointer

- Definition
- Initialization
- Operators
- Relationship between array and pointer

Definition of Pointer

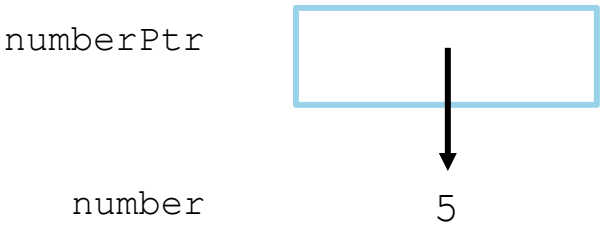
Pointers are variables whose values are memory addresses

Definition of Pointer

- A pointer contains an address of a variable that contains a specific value.
- A variable name directly references a value, but a pointer indirectly references a value.

Pointer **numberPtr** indirectly references the value 5

Variable **number** directly contains the value 5



Pointer Initialization

- Pointers must be defined before they can be used

```
data_type *pointer_name;
```

- Example

```
int *iPtr;
```

  - What is the data type of **iPtr** ?

```
int *iPtr;
```

  - What is the data type of ***iPtr** ?

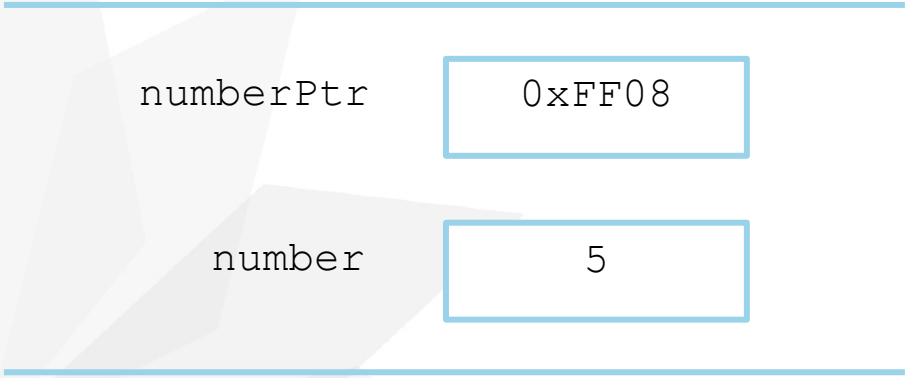
```
int *iPtr;
```

Pointer Initialization

- Pointers should be initialized either when they are defined or in an assignment statement
- A pointer may be initialized to NULL or an address
- A pointer with the value NULL points to nothing

Pointer Operators

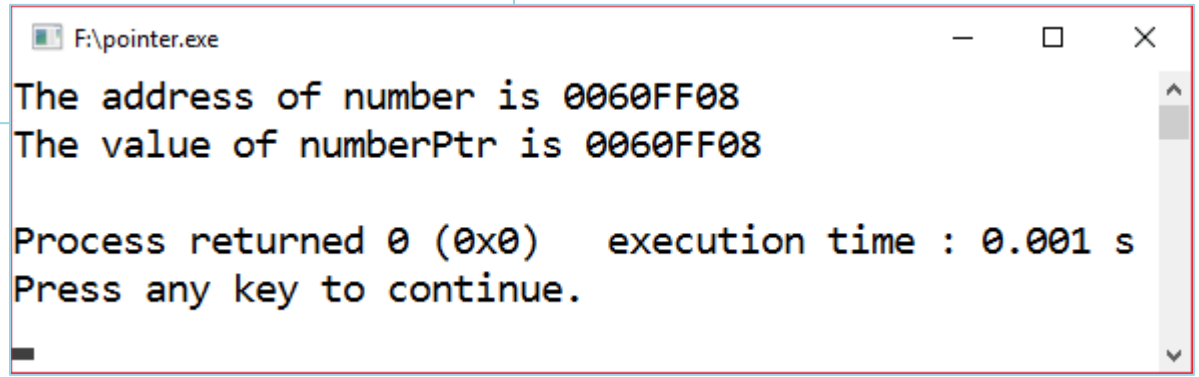
```
int number = 5;
int *numberPtr;

numberPtr = &number;

printf("%d",*numberPtr);
```



- The address operator (**&**) returns the address of its operand (variable).
- The indirection operator / dereferencing op erator (*****) returns the value of the object to w hich its operand (pointer) points.

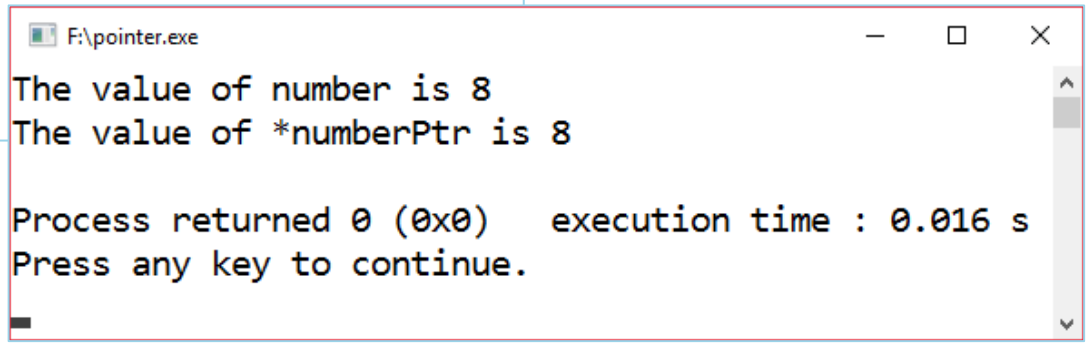
Pointer Operators

```c
#include <stdio.h>

int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    printf("The address of number is %p\n",&number);
    printf("The value of numberPtr is %p\n",numberPtr);

    return 0;
}
```



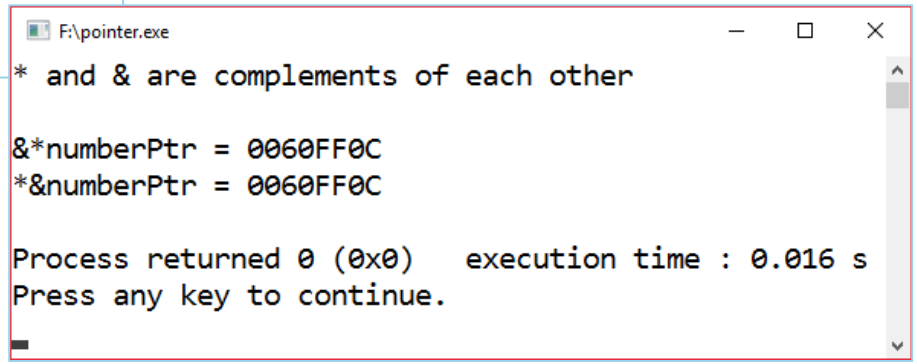
Pointer Operators

```c
#include <stdio.h>

int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    printf("The value of number is %d\n",number);
    printf("The value of *numberPtr is %d\n",*numberPtr);

    return 0;
}
```



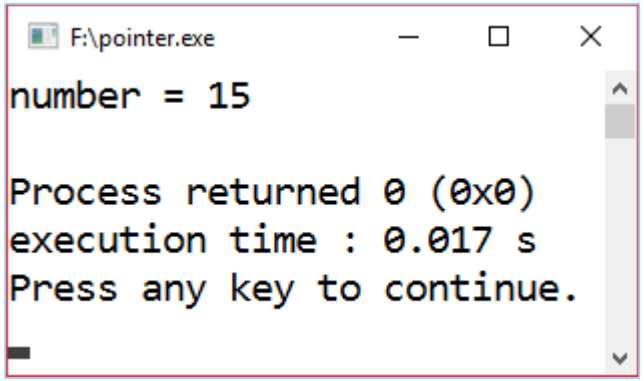
Pointer Operators

```c
#include <stdio.h>

int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    printf("* and & are complements of each other\n\n");
    printf("&*numberPtr = %p\n",&*numberPtr);
    printf("*&numberPtr = %p\n",*&numberPtr);

    return 0;
}
```



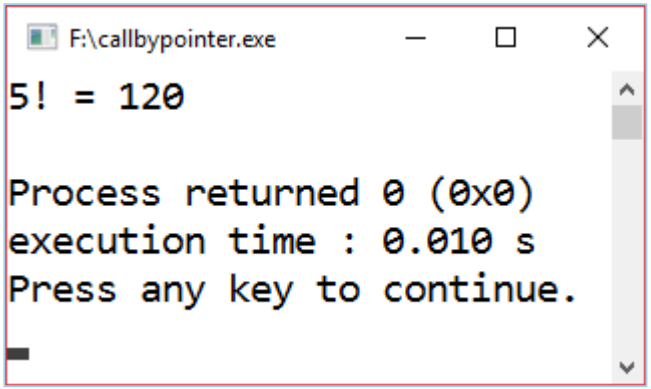
Pointer Operators

```c
#include <stdio.h>

int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    *numberPtr = number + 7;
    printf("number = %d\n",number);

    return 0;
}
```



Pointer Operators

```c
#include <stdio.h>

void factorial(int *n)
{
    int i;

    for(i = *n - 1;i > 1;i--)
    {
        *n *= i;
    }
}

int main()
{
    int number = 5;

    factorial(&number);
    printf("5! = %d\n",number);

    return 0;
}
```



Relationship Between Array and Pointer

```
int arr[5];
int *arrPtr;
```

- Since the array name (without a subscript) is a pointer to the first element of the array, we ca n set **arrPtr** equal to the address of the first element in array **arr**

```
arrPtr = arr;
```
=
```
arrPtr = &arr[0];
```

- The address **&arr[n]** can be written with the pointer expression

```
arrPtr + n
```

Relationship Between Array and Pointer

```
int arr[5];
int *arrPtr;
```

- Array element **arr[n]** can alternatively be referenced with the pointer expression

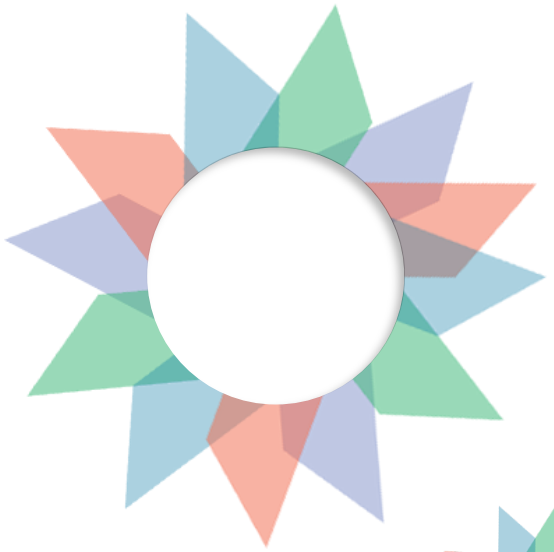
```
*(arrPtr + n)
```

- The array itself can be treated as a pointer

```
*(arr + n)
```

- Pointers can be subscripted exactly as arrays can

```
arrPtr[n]
```



Strings and Characters

- Definition
- Operations
- Array of Strings
- Pointer to Strings

Definition of String

A string is a series of characters treated as a single unit

Definition of String

- A string may include letters, digits, and various special characters
- String literals (string constants) in C are written in **double quotation marks**
- A string in C is an **array of characters ending in the null character ('\0')**
- A string is accessed via a pointer to the first character in the string
- The value of a string is the address of its first character

Operations on String

```
char color[] = “blue”;
char color[] = {‘b’, ‘l’, ‘u’, ‘e’};
```

- Create a 5-element array color containing the characters ‘b’, ‘l’, ‘u’, ‘e’, and ‘\0’

color[0]	color[1]	color[2]	color[3]	color[4]
b	l	u	e	\0

```
char color[7] = “blue”;
```

color[0]	color[1]	color[2]	color[3]	color[4]	color[5]	color[6]
b	l	u	e	\0		

Operations on String

```
char *color = “blue”;
```

- Creates pointer variable color that points to the string “blue” somewhere in memory

color[0]	color[1]	color[2]	color[3]	color[4]
b	l	u	e	\0

Operations on String

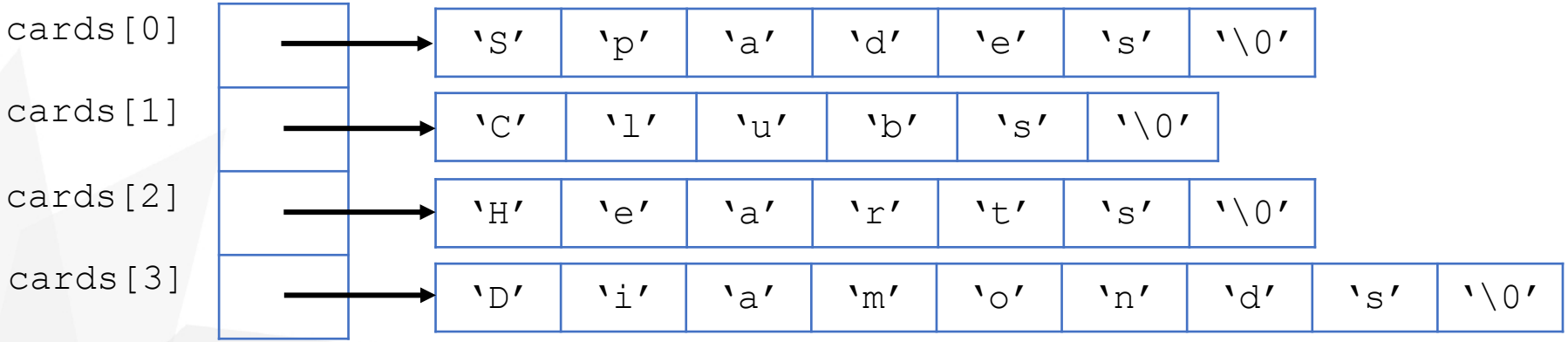
Function Prototype	Function Description
`char *strcpy(char *s1, const char *s2);`	**Copies string s2 into array s1** The value of s1 is returned
`char *strncpy(char *s1, const char *s2, size_t n);`	**Copies at most n characters of string s2 into array s1** The value of s1 is returned
`char *strcat(char *s1, const char *s2);`	**Appends string s2 to array s1** The first character of s2 overwrites the terminating null character of s1 The value of s1 is returned
`size_t strlen(const char *s);`	**Determines the length of string s** The number of characters preceding the terminating null character is returned

Operations on String

Function Prototype	Function Description
`int strcmp(const char *s1, const char *s2);`	**Compares the string s1 with the string s2** The function returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, respectively
`int strncmp(const char *s1, const char *s2, size_t n);`	**Compares up to n characters of the string s1 with the string s2** The function returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, respectively

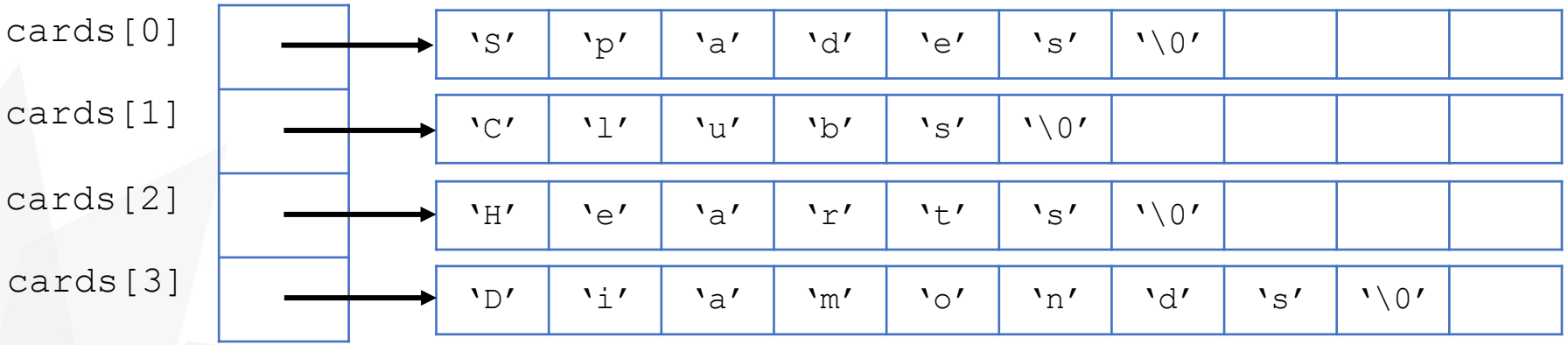
Array of Strings

```
char *cards[4] = {“Spades”, “Clubs”, “Hearts”, “Diamonds”};
```



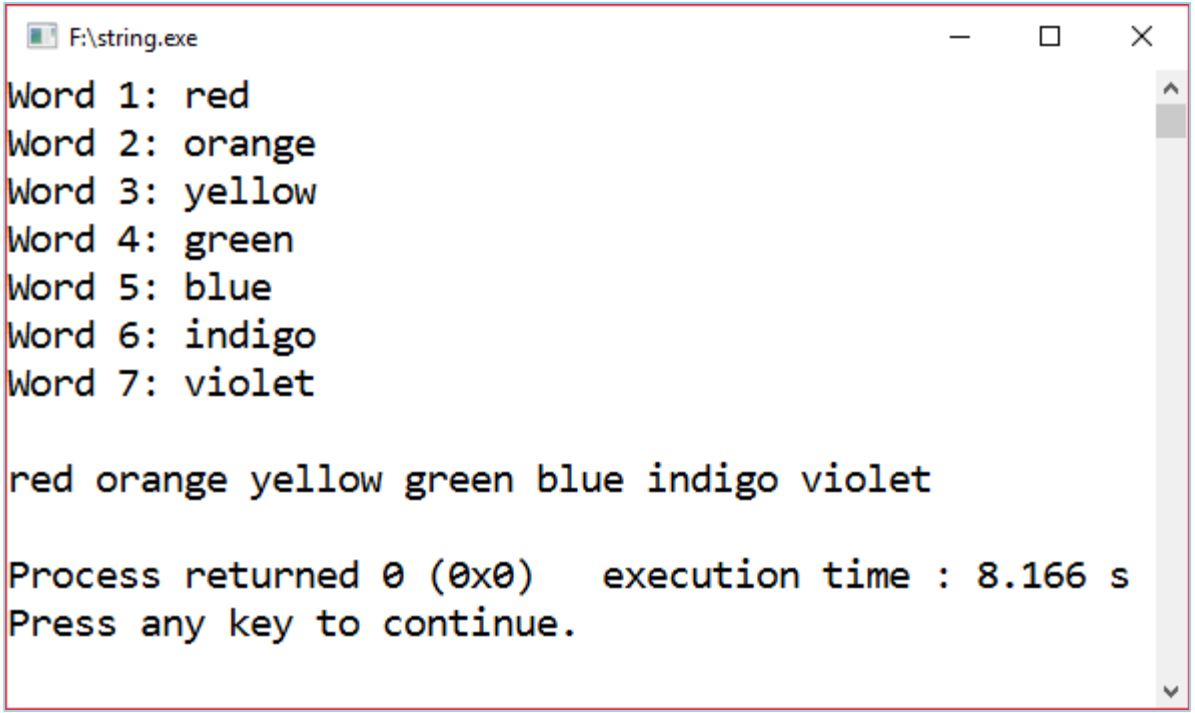
Array of Strings

```
char *cards[4][10] = {“Spades”, “Clubs”, “Hearts”, “Diamonds”};
```



Array of Strings

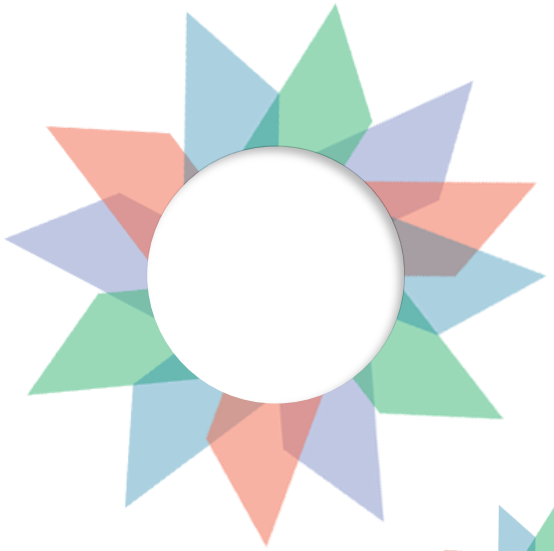
```c
#include <stdio.h>

int main()
{
    int i;
    char words[7][10];

    for(i = 0;i < 7;i++)
    {
        printf("Word %d: ",i+1);
        scanf("%s",words[i]);
    }
    printf("\n");

    for(i = 0;i < 7;i++)
    {
        printf("%s ",words[i]);
    }
    printf("\n");

    return 0;
}
```



Pointer to Strings

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

int main()
{
    char word[15];
    char *wordPtr;

    strcpy(word,"Algorithm");
    wordPtr = word;

    printf("[1] word\t\t: %s\n",word);
    printf("[2] wordPtr\t\t: %s\n",wordPtr);
    printf("[3] *wordPtr\t\t: %c\n",*wordPtr);
    printf("[4] *(wordPtr + 2)\t: %c\n",*(wordPtr + 2));

    strcpy(word,"Programming");
    printf("[5] wordPtr\t\t: %s\n",wordPtr);

    wordPtr = "Computer";
    printf("[6] word\t\t: %s\n",word);
    printf("[7] wordPtr\t\t: %s\n",wordPtr);

    return 0;
}
```

```
F:\string.exe
[1] word                : Algorithm
[2] wordPtr             : Algorithm
[3] *wordPtr            : A
[4] *(wordPtr + 2)      : g
[5] wordPtr             : Programming
[6] word                : Programming
[7] wordPtr             : Computer

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```



Practice

Practice 1

- Write a for loop that sums the odd values from the 10-element array n. For example, the sum for this array would be 45 + 97 + 29 + 7 + 21 = 199.

n[0]	n[1]	n[2]	n[3]	n[4]	n[5]	n[6]	n[7]	n[8]	n[9]
45	97	29	42	50	7	12	62	21	10

Practice 2

- Write a for loop that sums the even-numbered elements from array n. For example, the sum for this array would be 45 + 29 + 50 + 12 + 21 = 157.

n[0]	n[1]	n[2]	n[3]	n[4]	n[5]	n[6]	n[7]	n[8]	n[9]
45	97	29	42	50	7	12	62	21	10

Practice 3

- Write a program to store an input list of five integers in an array, then display each data value and what percentage each value is of the total of all five values. The screen dialog ue should appear as follows:

```
 48                24.00
 62                31.00
 37                18.50
  3                 1.50
 50                25.00
```

```
0 <= input <= 99
```

Practice 4

- Write a program to read and display a 3 x 3 matrix.

Practice 5

- Given the string name (value is “Adams, John Quincy”) and the 40-character temporary variables tmp1 and tmp2, what string is displayed by the following code fragment?

```
strncpy(tmp1, &name[7], 4);
tmp1[4] = '\0';
strcat(tmp1, " ");
strncpy(tmp2, name, 5);
tmp2[5] = '\0';
printf("%s\n", strcat(tmp1, tmp2));
```

Practice 6

- Store in s3 matching initial portions of s1 and s2. For example, if s1 is “placozoa” and s2 is “placement”, s3 becomes “plac”. If s1 is “joy” and s2 is “sorrow”, s3 becomes th e empty string.

References

1. Paul Deitel and Harvey Deitel. 2016. *C How to Program: with an introd uction to C++*, 8th Edition, Global Edition. Great Britain: Pearson Educa tion.
2. Reema Thareja, 2014. Data Structures Using C, 2nd Edition. India: Oxfor d University Press.

Next Outline

- Concept of Structures
- Structures Utilization
- Concept of Unions
- Unions Utilization
- Concept of Enumerations
- Enumerations Utilization


Thank you