C Built-in Arrays

Arrays - Introduction

- **The definition given in your textbook:**
  - An array contains objects (instances) of a given type, stored consecutively in a continuous memory block.

- An array definition has the following syntax:

  type_name   identifier   [number_of_elements];

- Examples of declaration of arrays with fixed length:

  int a[3] = {44, 66, 85};   // declares an array of three integers
  char c[3] = {'A', 'M', 'D'};   // declares an array of three characters
  double z[3] = {2.3, 5.5, 7.8};   // declares an array of three doubles

a[0]	44
a[1]	66
a[2]	85

c[0]	'A'
c[1]	'M'
c[2]	'D'

z[0]	2.3
z[1]	5.5
z[2]	7.8

Arrays – Declaration

- Arrays can have any storage classes:
  - Can be declared outside all function.
  - Can be declared as local variable.
  - Can be declared within a block.
  - Can be declared as other storage classes such as local or global **static** (we will discuss about the keyword **static** later in the course).
  - The only restriction is that no function parameter can be an array. We will see later that an array argument passed to a function is always converted into a pointer to the first array element.

- C99 also allows declaration of arrays with variable-length at the **runtime**.
  - Examples:

    double x [5 * 2];

    duble y [200 + 2];

    int x = 20;

    double z [x * 5];

Arrays – Access to Elements of the Array

- Accessing array elements:
  - Puts 99 into the third element of z:

    z[2] = 99;

- Displaying the element's value:
  - Displays the third element of array z

    printf("%d", z[2]);

  - How to display all element of z?

    - By using a for loop

- Reading and storing value into the array element:
  - Reads three integer numbers and stores into the the first three elements of the array z:

    for(int I = 0; I < 3; i++) {
        printf("\nEnter a number: ");
        scanf("%d", &z[i]) ;
    }

Arrays - Initialization

- Initializing Arrays:
  - To initialize an array explicitly when you define it, you must use an initialization list:

    int a[5] = { 10, 23, 33, 44, 66};

  - If you do not explicitly initialize an array variable, the usual rules apply:
    - If array has automatic storage, its elements have undefined values.
    - For other storage classes, all elements are initialized to the value 0.
  - You cannot include an initialization in the definition of a variable-length array.
  - You may omit the length of the array, if you supply an initialization list:

    int a[ ] = { 10, 23, 33, 44, 66};

  - If array declaration contains both length and initialization list, then the length is indicated by the length between square brackets.
    - Any element with no initializer is initialized to zero.
    - If the list has more initializers than the array length, the superfluous initializers are ignored.

Arrays – Copying Issues

- Unlike data structures in some other language, arrays in C can not be copies.

  int a [2];

  int b [2];

  a[0] = 55;

  a[1] = 60;

  b = a;   // ERROR -- NOT allowed

- Unlike arrays in Processing, C arrays don't do index range checking.

  - Let's see an example:

Array of Characters and C-Strings

What is Character, Character Constant, and String Constant in C

- Character in C is one byte memory space (or in fact a one byte integer).
- You can either assign an integer or a character constant in a character data type:

  char c;

  c = 65;

  c = 'A';

- A character constant is confined between **single quotation** marks. A character constant can be a letter, a digit, or a non-printing character such as '\n'.

  c = '3'

  c = '\n'   // non-printing – end of line

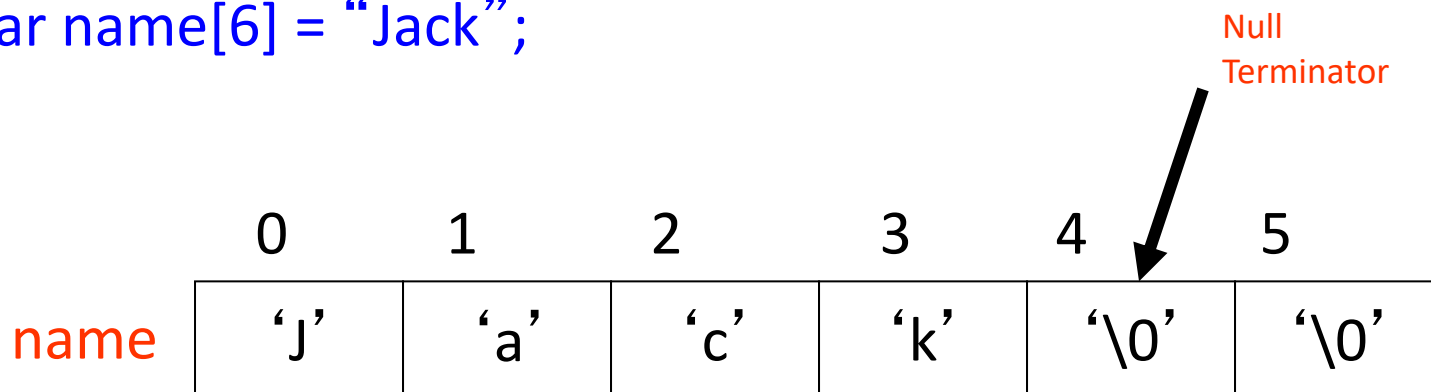- String constants are confined between double quotation marks:
  - "ABACDD"
  - "Judy"
  - "ENCM   339"
  - "34890'

C-Strings

- Basically, a string is represented by an array of characters which is terminated by a NULL ('\0') character.

- Null termination is only used for strings.

- Example:

char name[6] = "Jack";

Null Terminator

	0	1	2	3	4	5
name	'J'	'a'	'c'	'k'	'\0'	'\0'

Printing Strings and Characters

- Initializing array of characters and strings in C :

```
char str1[5] = "Apple"; // OK, as an array of chars, but not C-string
char str2[]  = "XYZ" ; // OK, as array of characters and C-string
char str3[3] = "AB";    // OK, as array of characters and C-string
char charArray [3] = {'X','Y','Z'};// OK as array of characters – not a C-string
```

- Access to the data in an element of a string:

```
printf("%c", str1[3]);              //displays 'l' (ell)
printf("%c", str2[2]);              //displays 'Z'
```

- Displaying strings:

```
printf("%s", str2);    //displays: XYZ
```

  – printf displays all characters starting from element zero, that up to the last element before the element that contains '\0'.

Question: Draw AR diagrams at point 1, 2, and 3? What is the program output?

```
int main(){
      char str[10];
      str[0] = 'A' ;
      str[1] = 'B' ;
      str[2] = 'C' ;
      // point 1

      str[3] = '\0'
      str[4] = '9' ;
      // point 2

      printf ("%s", str);
      str[5] = 'X' ;
      str[6] = '\0' ;
      // point 3

      printf ("%s", str);
      return 0;
}
```
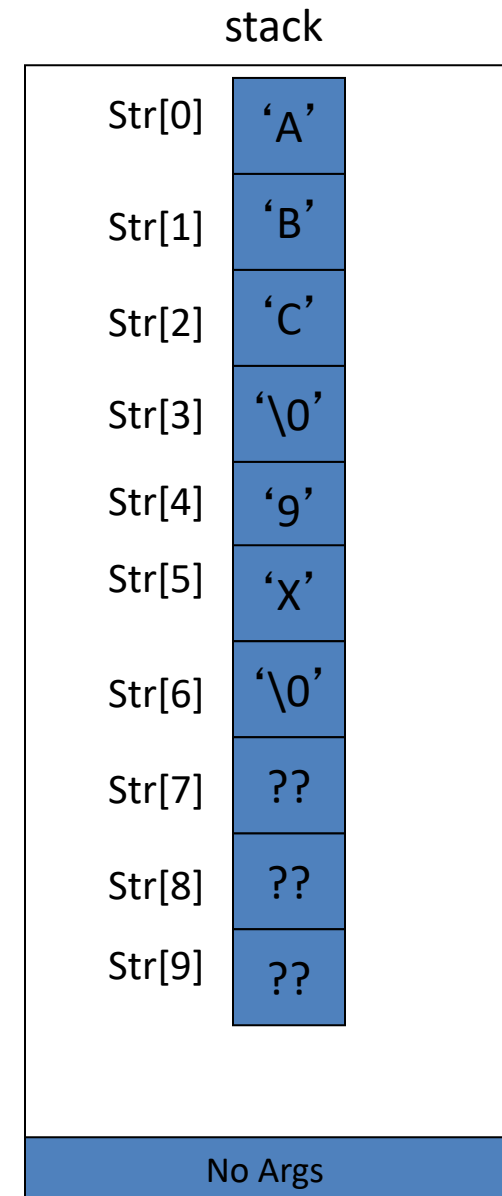
stack

Str[0]	'A'
Str[1]	'B'
Str[2]	'C'
Str[3]	'\0'
Str[4]	'9'
Str[5]	'X'
Str[6]	'\0'
Str[7]	??
Str[8]	??
Str[9]	??

AR main
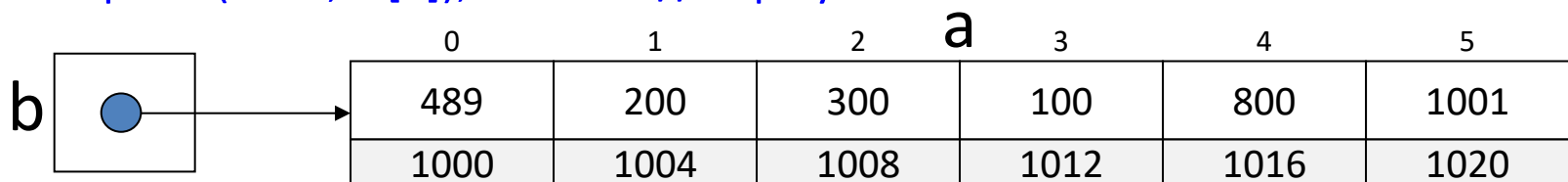
Point 3

No Args

Arrays and Pointers

Arrays and Pointers

- The name of built-in arrays in C and C++ is treaded as a pointer
  - The name of an array holds the address of the first byte of the first element.
  - In other words the name of an array can be treated like a fixed-pointer that points always to the first element of the array (it can not hold another address).

    ```
    int a [6] = {4, 2, 3, 1, 8, 11};
    printf("%d",  *a);              // Using Pointer Notation – prints 4
    printf("%d",  a[0]);            // Using Array Notation– prints 4
    printf("%d",  a[5]);            // prints 11
    ```

- Another Examples:
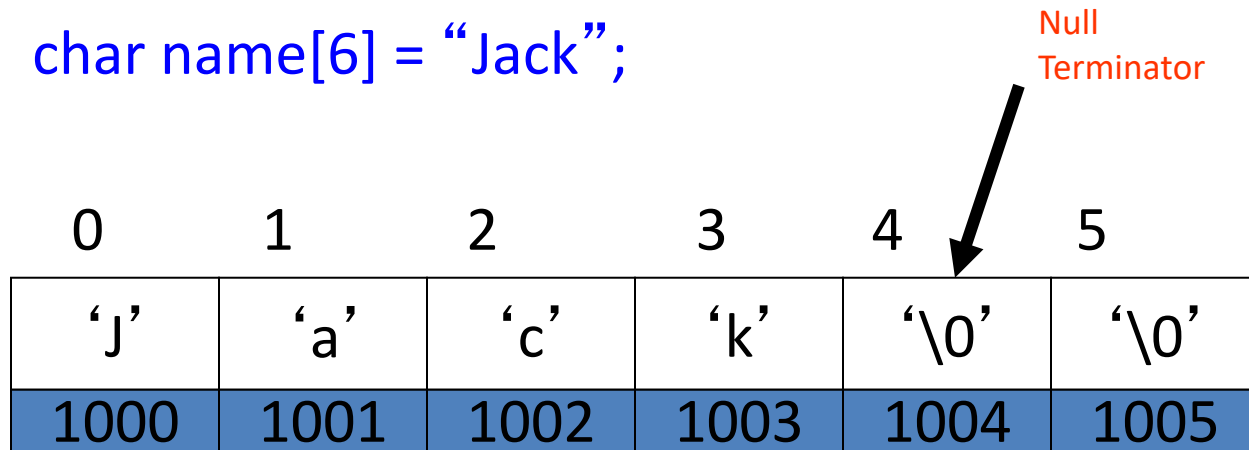
    ```
    int a[6] = {489, 200, 300, 100, 800, 1001};
    int *b;
    b = a;
    printf("%d",  *b);          //displays 489
    printf("%d",  *a);          //displays 489
    printf("%d",  b[4]);        //displays 800
    printf("%d",  a[4]);        //displays 800
    ```

a

	0	1	2	3	4	5
b →	489	200	300	100	800	1001
	1000	1004	1008	1012	1016	1020

Strings and Pointers

- A string name is also a pointer to the first element of an array of characters.

  char name[6] = "Jack";

Null Terminator

0	1	2	3	4	5
'J'	'a'	'c'	'k'	'\0'	'\0'
1000	1001	1002	1003	1004	1005

- The following statement is true:

  name == &name[0]

Address of name [0]

- Displaying strings:

  printf("%s", name);   //displays jack

  – printf displays all characters from name[0] to name[3].

Arrays and Functions

Passing Arrays to Functions

- You can pass an array to a function by passing the name of the array as an argument.
- You may also need to pass the number of elements of the array to the function.
- Since the name of the array is the address of the first element of the array, a function argument must be a pointer to hold this value.
- Example:

```c
#include <stdio.h>

int largest (const int *arr, int n);

int main()
{
    int x[5] = {90, 3, 4, 5, 1};
    int result;
    result = largest (x, 5); //only the name of the array is passed
    printf("The largest value is %d.", result);
    return 0;
}
```

Arrays as Function Arguments

As a function argument you can either use a pointer notation (int*), or array notation ([]).

- Argument arr in both cases in the following examples is a pointer, pointing to the first element of the array x in main.

- Within function largest, the value of any element of array x in main can be accessed via pointer arr.

```
int largest(const int *arr, int n)
 {

    int j;
    int max = arr[0];

    for (j = 1; j< n; j++)
         if (arr[j] > max)
           max = arr[j];

    return max;
}
```

```
int largest(const int arr[], int n)
 {

    int j;
    int max = arr[0];

    for (j = 1; j< n; j++)
         if (arr[ j ] > max)
            max = arr[j];

    return max;
}
```

Using `scanf` to Read Strings and Characters

Reading Strings and Characters using scanf

- scanf uses %s as a type identifier to read a string (up to a whitespace) from keyboard .

  char lastName[25];

  printf ("Enter your last name: ");

  scanf ("%s", lastName);

  – Three character: spacebar, tab and return are considered as whitespace characters.

- scanf doesn't need an address operator to read a string. Why?

- scanf uses %c as a type identifier to read a character.

  scanf ("%c", &lastName[0]);

  – Needs address operator to read a character.

- scanf is not the only library function to read strings and characters. There are other library functions such as:

  gets, fgets, getc, fgetc.

Using const keyword as a Function Argument

- When pointers are supposed to be used as read-only pointer you should use the const keyword.

- This style programming protects you data from malicious and unwanted changes.

```
void doSothing( const double *p)
{
    *p = 100.5;        // illegal

    // more code..
}
```

Pointers to Constant Characters

Using const char* to Declare a C-string

- Another way to define a c-string in C/C++ is to use a char*, pointing to a string constant on the static memory segment.

  char * p = "KLH" ;   // bad style. Why?

  const char * ptc = "CBC";   // better style – ptc is a pointer to a const

- How do we show this type of declaration in an AR diagram:

  - Assume the following declarations are in a main function;

  char str[ ] = "Jack";

  const char *ptr = "Joe";

Str[0] 'J' 2000
Str[1] 'a' 2001 2003
Str[2] 'c' 2002
Str[3] 'k'
Str[4] '\0' 2004

Stack

String Constant Area

(Static)

1000 'J'
1001 'o'
1002 'e'
1003 '\0'

AR main

ptr

No Args