Pointers and Arrays

Structural Programming

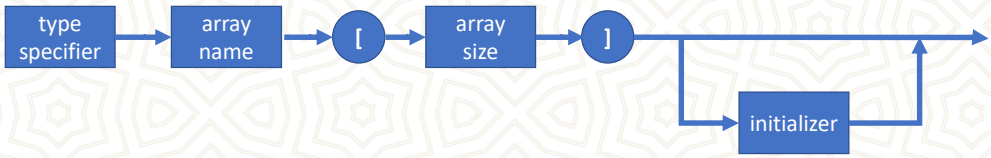
by

Z. Cihan TAYSI

Outline

- Declaration
- How arrays stored in memory
- Initializing arrays
- Accessing array elements through pointers
- Examples
- Strings
- Multi-dimensional arrays

Declaration



int dailyTemp[365];

dailyTemp[0] = 38;

dailyTemp[0] = 23;

- **subscripts begin at 0, not 1 !**

How Arrays Stored in Memory

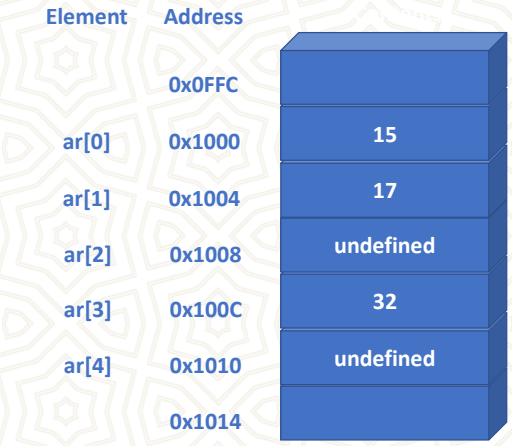
int ar[5]; /* declaration */

ar[0] = 15;

ar[1] = 17;

ar[3] = ar[0] + ar[1];

- **Note that ar[2] and ar[4] have undefined values!**
  - the contents of these memory locations are whatever left over from the previous program execution



Initializing Arrays

- It is incorrect to enter more initialization values than the number of elements in the array
- If you enter fewer initialization values than elements, the remaining elements initialized to zero.
- **Note that 3.5 is converted to the integer value 3!**
- When you enter initial values, you may omit the array size
  - the compiler automatically figures out how many elements are in the array…

```
int a_ar[5];
int b_ar[5] = {1, 2, 3.5, 4, 5};
int c_ar[5] = {1, 2, 3};

char d_ar[] = {'a', 'b', 'c', 'd'};
```

Accessing Array Elements Through Pointers

```
short ar[4];
short *p;

p = & ar[0]; // assigns the address of array element 0 to p.
```

- p = ar; **is same as above assignment!**
- *(p+3) refers to the same memory content as ar[3]

```
float ar[5], *p;
…
p = ar ;          // legal
ar = p;           // illegal
&p = ar;          // illegal
ar++;             // illegal
ar[1] = *(p+3);   // legal
p++;              // legal
```

examples

- Bubble sort

6 5 3 1 8 7 2 4

- Selection sort

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Strings

- A string is an array of characters terminated by a null character.
  - **null character is a character with a numeric value of 0**
  - it is represented in C by the escape sequence '\0'
- A string constant is any series of characters enclosed in double quotes
  - it has datatype of array of char and each character in the string takes up one byte!

- char str[] = "some text";
- char str[10] = "yes";
- **char str[3] = "four"**
- **char str[4] = "four"**

- char *ptr = "more text" ;

String Assignments

```
main () {
    char array[10];
    char *ptr="10 spaces";
    char *ptr2;
    array = "not OK";          // can NOT assign to an address !
    array[5] = 'A';            // OK
    ptr1[5] = 'B';             // OK
    ptr1="OK";                 // questionable due to the prior assignment
    ptr1[5]='C';               // type mismatch
    *ptr2 = "not OK";
    ptr2="OK";
}
```

Strings vs. Chars

Chars

```
char ch = 'a';      // one byte is allocated for 'a'
*p = 'a';           // OK
p = 'a';            // Illegal
```

Strings

```
char *p = "a";      // two bytes allocated for "a"
*p = "a";           // INCORRECT
p = "a";            // OK
```

Reading & Writing Strings

```c
#include <stdio.h>

#define MAX_CHAR 80

int main ( void )  {

        char str[MAX_CHAR];
        int i;

        printf("Enter a string :");
        scanf("%s", str);

        for(i=0;i<10;i++)  {
                printf("%s\n", str);
        }
        return 0;
}
```

- You can read strings with *scanf()* function.
  - the data argument should be a pointer to an array of characters ***that is long enough to store*** the input string.
  - after reading input characters scanf() automatically appends a null character to make it a proper string
- You can write strings with *printf()* function.
  - the data argument should be a pointer to a null terminated array of characters

String Length Function

- We test each element of array, one by one, until we reach the null character.
  - it has a value of zero, making the while condition **false**
  - any other value of str[i] makes the while condition **true**
  - once the null character is reached, we exit the while loop and return *i*, which is the last subscript value

```c
int strlen ( char *str)  {

        int i=0;

        while(str[i])  {
                i++;
        }

        return i;
}
```

String Copy Function

```
void strcpy ( char s1[], char s2[])  {
        int i;
        for(i=0; s1[i]; ++i)
                s2[i] = s[i];
        s2[++i] = '\0';
}
```

```
void strcpy ( char *s1, char *s2)  {
        int i;
        for(i=0; *(s+i); ++i)
                *(s2+i) = *(s1+i);
        s2[++i]= '\0';
}

void strcpy ( char *s1, char *s2)  {
        while(*s++ = *s1++);
}
```

Other String Functions

- strcpy()
- strncpy()
- strcat()
- strncat()
- strcmp()
- strncmp()
- strchr()
- strcspn()
- strpbrk()
- strrchr()
- strspn()
- strstr()
- strtok()
- strerror()
- strlen()
- ...

Pattern Matching Example

- Write a program that
  - gets two strings from the user
  - search the first string for an occurrence of the second string
  - if it is successful
    - return byte position of the occurence
  - otherwise
    - return -1

Multi-Dimensional Arrays

- In the following, ar is a 3-element array of 5-element arrays

  int ar[3][5];

- In the following, x is a 3-element array of 4-elemet arrays of 5-element arrays

  char x[3][4][5];

- the array reference

  ar[1][2]

- is interpreted as

  *(ar[1]+2)

- which is further expanded to

  ((ar+1)+2)

Initialization of Multi-Dimensional Arrays

```
int exap[5][3] = { {  1, 2, 3 },
                   { 4 },
                   { 5, 6, 7 } };
```

1	2	3
4	0	0
5	6	7
0	0	0
0	0	0

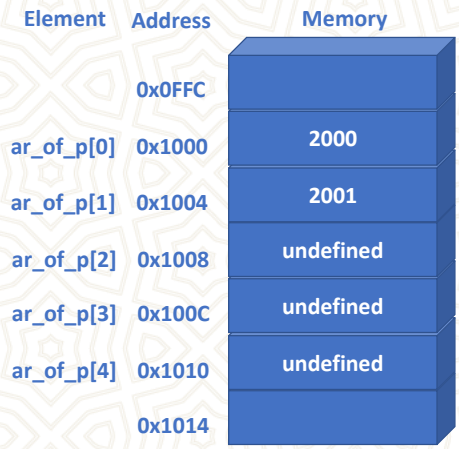
```
int exap[5][3] = {  1, 2, 3,
                    4,
                    5, 6, 7 };
```

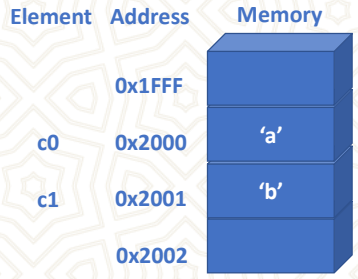
1	2	3
4	5	6
7	0	0
0	0	0
0	0	0

array of pointers

```
char *ar_of_p[5];
char c0 = 'a';
char c1 = 'b';

ar_of_p[0] = &c0;
ar_of_p[1] = &c1;
```





Pointers to Pointers

int r = 5;	declares ***r*** to be an int
int *q = &r;	declares ***q*** to be a pointer to an int
int **p = &q;	declares ***p*** to be a pointer to a pointer to an int
r = 10;	Direct assignment
*q = 10;	Assignment with one indirection
**p = 10;	Assignment with two indirections