18ES611 Embedded System Programming

Sarath tv

Array and pointers

- **int my_array[] = {1,23,17,4,-5,100};**

- An array containing 6 integers.
- Each of these integers by means of a using **my_array[0]** through **my_array[5]**.
- Alternatively access them via a pointer as follows.

- **Int *ptr;**

- **ptr = &my_array[0];** /* point our pointer at the first integer in our array */

- Initialize a int array
- Create a pointer
- Main
    - Initialize pointer to the starting address of the array.
    - Inside a loop print content of the array by index method and pointer method.

```c
#include <stdio.h>

int my_array[] = {1,23,17,4,-5,100};
int *ptr;

int main(void)
{
    int i;
    ptr = &my_array[0];     /* point our pointer to the first
                                    element of the array */
    printf("\n\n");
    for (i = 0; i < 6; i++)
    {
      printf("my_array[%d] = %d   ",i,my_array[i]);   /*<-- A */
      printf("ptr + %d = %d\n",i, *(ptr + i));        /*<-- B */
    }
    return 0;
}
```

- we might use **&var_name[0]** we can replace that with **var_name**, thus in our code where we wrote:
- ptr = &my_array[0];
- we can write:
- **ptr = my_array;** to achieve the same result.
- This means ***"the name of the array is the address of first element in the array".***

we cannot write **my_array = ptr;** 

The reason is that while **ptr is a variable**, **my_array is a constant**.

That is, the ***location*** at which the ***first element*** of my_array will be ***stored cannot*** be ***changed once my_array***[] has been ***declared***.

Array and String

- Strings are arrays of characters
- In C a string is an array of characters terminated with a binary zero
- character (written as **'\0'**)
- **terminated with a nul character**.

Copy a string using pointer

- *Create two strings and initialize one of the string with some random sentence.*
- *Create two char pointers – one for source and one for destination*
- *Initialize the pointers to the array address.*
- *Till the first string reaches nul*
    - *Copy char from first string to second –using dereferencing operator.*
- *After this terminate the second string using nul*
- *Print second string*

```
/***custom strcpy***//
#include <stdio.h>

char strA[80] = "A string to be used for demonstration purposes";
char strB[80];

int main(void)
{

    char *pA;      /* a pointer to type character */
    char *pB;      /* another pointer to type character */
    puts(strA);    /* show string A */
    pA = strA;     /* point pA at string A */
    puts(pA);      /* show what pA is pointing to */
    pB = strB;     /* point pB at string B */
    putchar('\n');         /* move down one line on the screen */
    while(*pA != '\0')
    {
        *pB++ = *pA++;
    }
    *pB = '\0';
    puts(strB);            /* show strB on screen */
    return 0;
```

when we write **puts(strA);** as we have seen, we are passing the address of **strA[0]**.

Function development

```
void my_strcpy(char *destination, char *source)
  {
     char *p = destination;
     while (*source != '\0')
     {
        *p++ = *source++;
     }
     *p = '\0';
}
```

Slight modification

- my_strcpy(char *destination, const char *source);
- Here the "const" modifier is used to **assure the user that the function will not modify the contents pointed to by the source pointer**.
- modifying the function above, and its prototype, to include the "const" modifier
- Then, within the function *source ='X';
  - Try this!!!

Write your own versions of following standard functions using pointers

- **strlen();**-The function takes a single argument, i.e, the string variable whose length is to be found, and returns the length of the string passed.
- **strcat();**-It takes two arguments, i.e, two strings or character arrays, and stores the resultant concatenated string in the first string specified in the argument.
- **strchr();** Searches for the first occurrence of the character **c** (an unsigned char) in the string pointed to by the argument **str**.
- syntax-char **strchr(const char *str, int c)*

Slight change

```
void  my_strcpy(char dest[], char source[])
{
int i = 0;
while (source[i] != '\0')
{
dest[i] = source[i];

i++;
}
dest[i] = '\0';

}
```

We have chosen to use array notation instead of pointer notation to do the actual copying.

The results are the same, i.e. the string gets copied using this notation just as accurately as it did before.

Parameters are passed by value, in both the **passing of a character pointer or the name of the array** as above, **what actually gets passed is the address of the first element of each array.** Thus, the numerical value of the parameter passed is the same whether we use a character pointer or an array name as a parameter

This would imply that somehow **source[i]** is the same as ***(p+i)**.

- Wherever one writes **a[i]** it can be replaced with ***(a + i)** without any problems. In fact, the compiler will create the same code in either case.
- Either syntax produces the same result.
  - **NOT saying that pointers and arrays are the same thing, they are not.**
- We are only saying that **to identify a given element of an array** we have the choice of **two syntaxes**, **one using array indexing** and the **other using pointer arithmetic**, which yield identical results.

- The expression **(a + i)**, is a simple addition using the + operator and the rules of C state that such an expression is **commutative**. That is **(a + i) is identical to (i + a)**.
- Thus we could write ***(i + a)** just as easily as ***(a + i)**.

- **But *(i + a) could have come from i[a] !????**

char a[20];
int i;
writing
a[3] = 'x';
is the same as writing
3[a] = 'x';

Try it!

- char my_name[] = "Ted";
- char *my_name = "Ted";
- **Is there a difference between these?**

- Using the array notation **4 bytes** of **storage in the static memory** block are taken up, one for each character and one for the terminating nul character.
- But, in the pointer notation the same **4 bytes required**, **plus N** bytes to store the pointer variable **my_name** (where N depends on the system but is usually a minimum of 2 bytes and can be 4 or more).
- In the **array notation**, **my_name is short for &myname[0]** which is *the address of the first element of the array*. Since **the location of the array is fixed during run time, this is a constant** (not a variable). In the **pointer notation my_name is a variable.**

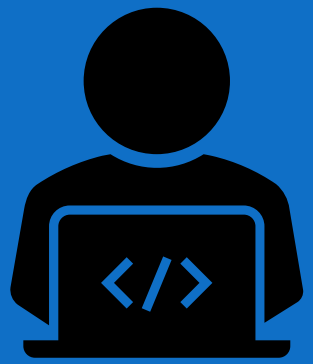
```
void my_function_A(char *ptr)
{
char a[] = "ABCDE"
.
.
}

void my_function_B(char *ptr)
{
char *cp = "FGHIJ"
.
.
}
```

In the case of **my_function_A**, the content, or value(s), of the array **a[]** is considered to be the data. The array is said to be initialized to the values ABCDE.

In the case of **my_function_B**, the value of the pointer **cp** is considered to be the data. The pointer has been *initialized to point to the string* **FGHIJ**.

In both **my_function_A** and **my_function_B** the definitions are local variables and thus the string **ABCDE** is stored on the stack, as is the value of the pointer **cp**. The string **FGHIJ** can be stored anywhere.

- Sometimes code getting the crash due to improper use of pointers. If you do not use the pointers in a proper way, the pointer can become the curse and it can create a very crucial issue (segmentation fault).



THANK YOU!!!!!