CS 332/532 Systems Programming

Lecture 5

- Pointers-

Professor : Mahmut Unan – UAB CS

Agenda

- Pointers
- Double Pointer
- Arrays & Pointers
- Passing Pointers to Functions
- Char
- Strings

Example - page 1/2

```c
#include <stdio.h>
int main()
{
    int *ptr, a;
    a = 25;
    /* without using a pointer */
    printf("Address of a: %p\n", &a);
    printf("Value of a: %d\n", a);

    /*let's use a pointer */
    ptr = &a;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);

    /* how about if we change the value of int */
    a = 125;
    printf("Address of the pointer : %p\n", ptr);
    printf("Value of the pointer : %d\n", *ptr);
```

Example - page 2/2

```c
20        /* let's change the value using pointer*/
21        *ptr = 250;
22        printf("Address of a: %p\n", &a);
23        printf("Value of a: %d\n", a);
24
25        /* we can reuse the pointer */
26        int b = 50;
27        ptr = &b;
28        printf("Address of the pointer : %p\n", ptr);
29        printf("Value of the pointer : %d\n", *ptr);
30        return 0;
31    }
```

Example - output

```
Address of a: 0x7ffeee56291c
Value of a: 25
Address of the pointer : 0x7ffeee56291c
Value of the pointer : 25
Address of the pointer : 0x7ffeee56291c
Value of the pointer : 125
Address of a: 0x7ffeee56291c
Value of a: 250
Address of the pointer : 0x7ffeee562918
Value of the pointer : 50
```
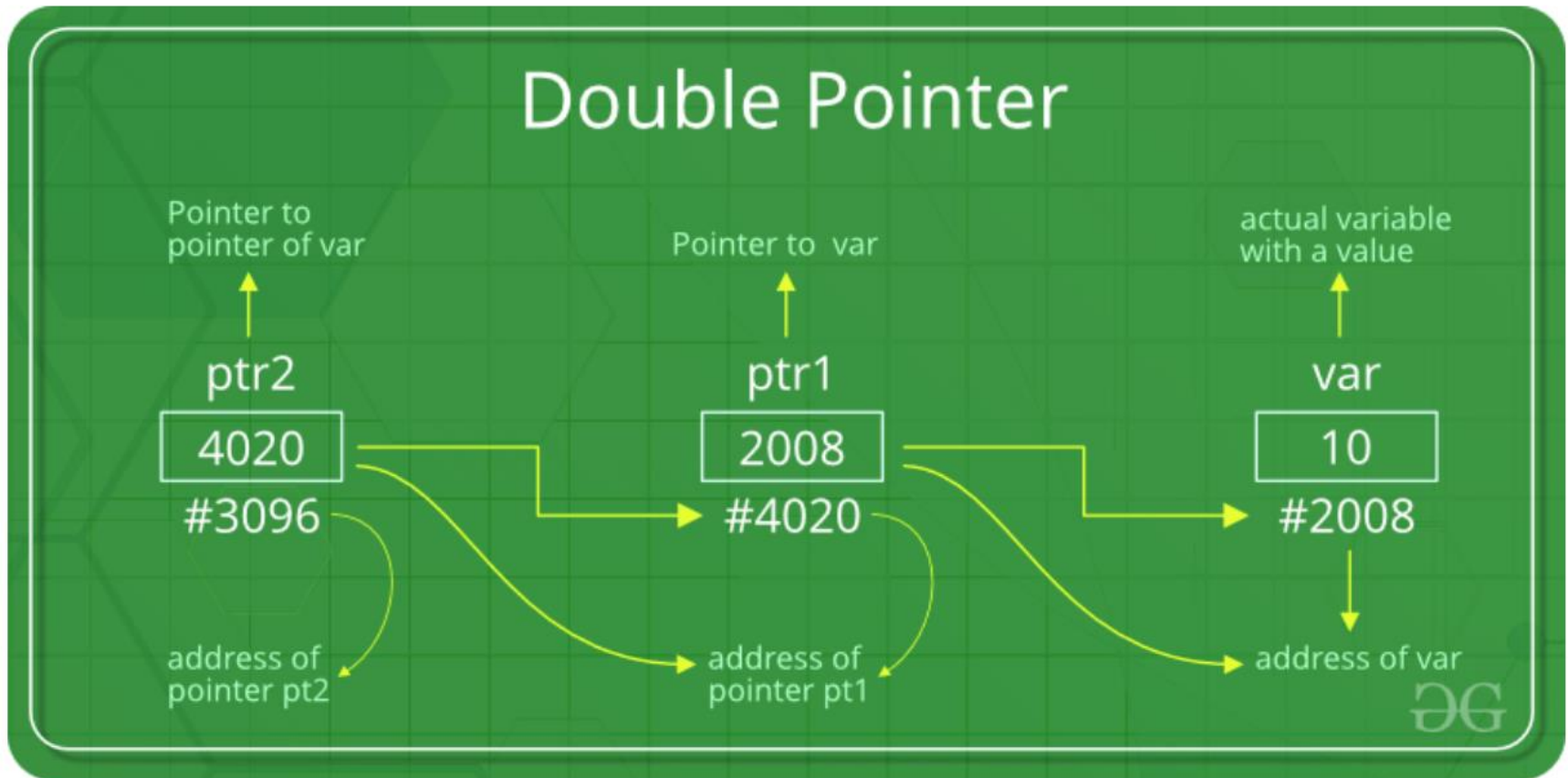
The * and & cancel each other when used together

```c
#include <stdio.h>

int main() {
    int *ptr, i=5;
    ptr = &i;
    printf("%p %p %p %d %p\n", &i, *&ptr, &*ptr, *ptr, &ptr);
    return 0;
}
```

```
0x7ffee636291c 0x7ffee636291c 0x7ffee636291c 5 0x7ffee6362920
```

double pointer? even triple…

```c
#include <stdio.h>
int main() {

    int a = 25;
    int *ptr = &a;//the first pointer

    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);

    *ptr = 45;// change the value
    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);

    int **ptr2 = &ptr;// second pointer
    printf("Address - First pointer: %p\n", ptr);
    printf("Value -First Pointer: %i\n", *ptr);
    printf("Address - First pointer: %p\n", ptr2);
    printf("Value -First Pointer: %i\n", **ptr2);
    return 0;
}
```

```c
#include <stdio.h>
int main() {

    int a = 25;
    int *ptr = &a;//the first pointer

    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);


    *ptr = 45;// change the value
    printf("Address : %p\n", ptr);
    printf("Value : %i\n", *ptr);


    int **ptr2 = &ptr;// second pointer
    printf("Address - First pointer: %p\n", ptr);
    printf("Value -First Pointer: %i\n", *ptr);
    printf("Address - First pointer: %p\n", ptr2);
    printf("Value -First Pointer: %i\n", **ptr2);
    return 0;

}
```

```
Address : 0x7ffee76aa928
Value : 25
Address : 0x7ffee76aa928
Value : 45
Address - First pointer: 0x7ffee76aa928
Value -First Pointer: 45
Address - First pointer: 0x7ffee76aa920
Value -First Pointer: 45
```

Arrays & Pointers

```c
#include <stdio.h>
int main() {
    int i,arr[5];
    double arr2[5];

    for(i = 0; i <5; i++)
        printf("address of arr[%d] = %p\n", i, &arr[i]);

    for(i = 0; i <5; i++)
        printf("address of arr2[%d] = %p\n", i, &arr2[i]);
    return 0;
}
```

Arrays & Pointers

```
address of arr[0] = 0x7ffeeece0910
address of arr[1] = 0x7ffeeece0914
address of arr[2] = 0x7ffeeece0918
address of arr[3] = 0x7ffeeece091c
address of arr[4] = 0x7ffeeece0920
address of arr2[0] = 0x7ffeeece08e0
address of arr2[1] = 0x7ffeeece08e8
address of arr2[2] = 0x7ffeeece08f0
address of arr2[3] = 0x7ffeeece08f8
address of arr2[4] = 0x7ffeeece0900
```

Array Manipulation

```c
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr;

    ptr = &arr[3]; // address of the fourth element

    printf("\n Pointer value : %d", *ptr);
    printf(" \n Next Value : %d", *(ptr+1));
    printf("\n Previous Value : %d", *(ptr-1));

    printf("\n Address of the Pointer : %p", &(*(ptr)));
    printf("\n Address of the Next Value : %p", &(*(ptr+1)));
    printf("\n Address of the Previous Value : %p", &(*(ptr-1)));
    return 0;
}
```

Array Manipulation

```
Pointer value : 40
Next Value : 50
Previous Value : 30
Address of the Pointer : 0x7ffeeb08e91c
Address of the Next Value : 0x7ffeeb08e920
Address of the Previous Value : 0x7ffeeb08e918
```

```c
    printf("\n Pointer value : %d", *ptr);
    printf(" \n Next Value : %d", *(ptr+1));
    printf("\n Previous Value : %d", *(ptr-1));

    printf("\n Address of the Pointer : %p", &(*(ptr)));
    printf("\n Address of the Next Value : %p", &(*(ptr+1)));
    printf("\n Address of the Previous Value : %p", &(*(ptr-1)));
    return 0;
}
```

What is the Result ?

```c
#include <stdio.h>
int main(void)
{
    int *ptr, totalSum, arr[5] = {10, 20, 30, 40, 50};
    totalSum = 0;
    for(ptr = arr; ptr < arr+5; ptr++)
    {
        --*ptr;
        totalSum += *ptr;
    }
    printf("Sum = %d\n", totalSum);
    return 0;
}
```

What is the Result ?

```c
#include <stdio.h>
int main(void)
{
    int *ptr, totalSum, arr[5] = {10, 20, 30, 40, 50};
    totalSum = 0;
    for(ptr = arr; ptr < arr+5; ptr++)
    {
        --*ptr;
        totalSum += *ptr;
    }
    printf("Sum = %d\n", totalSum);
    return 0;
}
```

145

Passing Pointers to functions

```c
#include <stdio.h>
void test(int a);
int main(void)
{
    void (*ptr)(int a);
    ptr = test;
    (*ptr)(10);
    return 0;
}
void test(int a)
{
    printf("%d\n", 2*a);
}
```

```
20
```

```c
#include <stdio.h>
int addTwoNumbers(int a, int b);
int subtractTwoNumbers(int a, int b);

int main(void)
{
    int (*ptr[2])(int a, int b);
    int i, j, result;
    ptr[0] = addTwoNumbers;
    ptr[1] = subtractTwoNumbers;

    printf("Enter two integer numbers: ");
    scanf("%d %d", &i, &j);
    if(i > 0 && i < 25)
        result = ptr[0](i, j);
    else
        result = ptr[1](i, j);
    printf("Result  : %d\n", result);
    return 0;
}
int addTwoNumbers(int a, int b)
{   return a+b;  }
int subtractTwoNumbers(int a, int b)
{   return a-b;   }
```

17

```c
#include <stdio.h>
int addTwoNumbers(int a, int b);
int subtractTwoNumbers(int a, int b);

int main(void)
{
    int (*ptr[2])(int a, int b);
    int i, j, result;
    ptr[0] = addTwoNumbers;
    ptr[1] = subtractTwoNumbers;

    printf("Enter two integer numbers: ");
    scanf("%d %d", &i, &j);
    if(i > 0 && i < 25)
        result = ptr[0](i, j);
    else
        result = ptr
    printf("Result
    return 0;
}
int addTwoNumbers(int
{   return a+b;   }
int subtractTwoNumber
{   return a-b;    }
```

```
Enter two integer numbers: 20 30
Result  : 50
```

```
Enter two integer numbers: 45 20
Result  : 25
```

The `char` Type

- Since a character in the ASCII set is represented by an integer between 0 and 255, we can use the **char** type to store its value.

- Once a character is stored into a variable, it is the character's ASCII value that is actually stored.

char `ch;`

`ch = 'c';`

  - the value of `ch` becomes equal to the ASCII value of the character 'c'.
  - Therefore,
    - the statements `ch = 'c';` and `ch = 99;` are equivalent.
    - Of course, 'c' is preferable than 99; not only it is easier to read, but also your program won't depend on the character set as well.

```c
#include <stdio.h>
int main(void)
{
    char ch;
    ch = 'a';
    printf("Char = %c and its ASCII code is %d\n", ch, ch);
    return 0;
}
```

```
Char = a and its ASCII code is 97
```

- Since C treats the characters as integers, we can use them in numerical expressions. For example:

```
char ch = 'c';
int i; ch++; /* ch becomes 'd'. */
ch = 68; /* ch becomes 'D'. */
i = ch-3; /* i becomes 'A',that is 65 */
```

Example

```c
C upperLower.c > ...
1    #include<stdio.h>
2    #include<string.h>
3    int main()
4    {
5      char s[100];
6      printf("Enter a  smaple string:    ");
7      scanf("%[^\n]",s);
8
9      printf("Output in uppercase:\n");
10     puts(strupr(s));
11
12     return 0;
13   }
14
15
```

Example

```c
C upperLower.c > ...
1    #include<stdio.h>
2    #include<string.h>
3    int main()
4    {
5        char s[100];
6        printf("Enter a  smaple string:   ");
7        scanf("%[^\n]",s);
8
9        printf("Output in uppercase:\n");
10       puts(strupr(s));
```

```
PS C:\Users\unan\Desktop\c_coding> gcc upperLower.c -o upperLower
PS C:\Users\unan\Desktop\c_coding> .\upperLower.exe
Enter a  smaple string:   mahmut unan
Output in uppercase:
MAHMUT UNAN
```

Without using string library

```c
C upperLower2.c > ⬡ main()
1    #include<stdio.h>
2     int main()
3     {
4         char s[100];
5         int i = 0;
6
7        printf("Enter a  smaple string:    ");
8         scanf("%[^\n]",s);
9
10        while( s[i] != '\0' )
11        {
12
13            if( s[i] >= 97 && s[i] <= 122 )
14            // or  if( s[i] >= "a" && s[i] <= "z" )
15            {
16                s[i] = s[i] - 32;
17            }
18            i++;
19        }
20
21        printf("Output in uppercase:\n");
22        puts(s);
23
24        return 0;
25     }
```

Without using string library

```c
C upperLower2.c > ⊕ main()
1    #include<stdio.h>
2    int main()
3    {
4        char s[100];
5        int i = 0;
6
7        printf("Enter a  smaple string:   ");
8        scanf("%[^\n]",s);
9
10       while( s[i] != '\0' )
11       {
12
13           if( s[i] >= 97 && s[i] <= 122 )
14           // or  if( s[i] >= "a" && s[i] <= "z" )
15           {
16               s[i] = s[i] - 32;
17           }
18           i++;
```

```
PS C:\Users\unan\Desktop\c_coding> gcc upperLower2.c -o upperLower2
PS C:\Users\unan\Desktop\c_coding> .\upperLower2.exe
Enter a  smaple string:   mahmut unan
Output in uppercase:
MAHMUT UNAN
```

```
25    }
```

`getchar()` and `putchar()`

- The `getchar()` function is used to read a character from stdin.
- The `putchar()` function writes a character in stdout, for example, `putchar('a')`

Strings

- A string literal is a sequence of characters enclosed in double quotes.

- C treats it as a nameless character array.

- To store a string in a variable, we use an array of characters.

- Because of the C convention that a string ends with the null character, to store a string of **N** characters, the size of the array should be **N+1** at least.

```
char str[8];
```

An array can be initialized with a string, when it is declared. For example, with the declaration:

```
char str[8] = "message";
```

the compiler copies the characters of the "message" into the str array and adds the null character. In particular, str[0] becomes 'm', str[1] becomes 'e', and the value of the last element str[7] becomes '\0'. In fact, this declaration is equivalent to:

```
char str[8] = {'m', 'e', 's',
's', 'a', 'g', 'e', '\0'};
```

puts()

```c
#include <stdio.h>
int main(void)
{
    char str[] = "UAB CS 330 Course";
    puts(str);
    puts(str);
    str[4] = '\0';
    printf("%s\n", str);
    return 0;
}
```

```
UAB CS 330 Course
UAB CS 330 Course
UAB
```