CPE102 Programming II

Week 5

Pointers

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

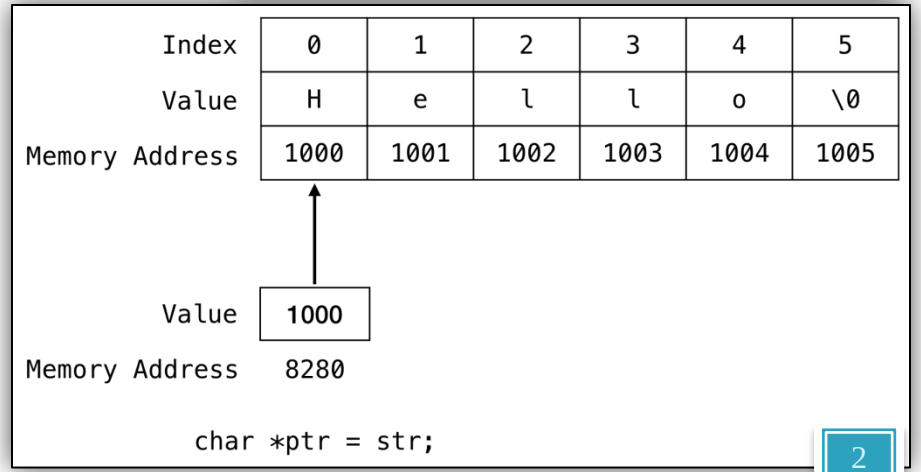
Relationship Between Pointers and Arrays

- An array name can be thought as a constant pointer. The array name str points at the first location of the allocated memory locations of the array.
- Arrays and Pointers are closely related.
- Pointers can also point arrays like they point variables.

```
char str[6] = "Hello";
char *ptr;
ptr= str; //Now ptr[0] and str[0] is same.
```

- To explicitly assign ptr to the address of first element of str use ptr= & str[0]

Variable	Index	Value	Address
str	0	H	1000
	1	e	1001
	2	l	1002
	3	l	1003
	4	o	1004
	5	\0	1005



Incrementing pointer variable

- We can use the pointers to access the elements of the array:

  *(ptr+ n) ;//n indicates the index number of the array element

  *(ptr+ 4); //gets the value of element str[4]

- Other alternatives for str[4]

  ptr[4]

  *(str + 4)

- we can increment the pointer variable to points to the next memory location based on the size of the data type.

  ptr++

  Ptr--

Pointers and Arrays, Example

ArrayPtr1.c

```c
#include <stdio.h>

int main(void) {
  // character array
  char str[6] = "Hello";
  // pointer ptr
  // pointing at the character array str
  char *ptr = str;

  // print the elements of the array str
  while (*ptr != '\0') {
    printf("%c\n", *ptr);
    // make the pointer ptr point at the
    // next memory location
    ptr++;
  }
  printf("End of code\n");
  return 0;
}
```

ArrayPtr2.c

```c
#include <stdio.h>

int main(void) {

  // character array
  char str[6] = "Hello";
  // pointer ptr
  // pointing at the character array str
  char *ptr = str;
  // print the elements of the array str
    for(int i=0;i<6;i++)
    printf("%c\n", *(ptr+i));

  printf("End of code\n");

  return 0;
}
```

You can change *(ptr+i) with: *(str+i) , ptr[i], or str[i]

Pointers and Two Dimensional Array

col →

row ↓	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
int num[3][4] = {
  {1, 2,  3,  4},
  {5, 6,  7,  8},
  {9, 10, 11, 12}
};
```

row-wise memory allocation

	← row 0 →				← row 1 →				← row 2 →			
value	1	2	3	4	5	6	7	8	9	10	11	12
address	1000	1002	1004	1006	1008	1010	1012	1014	1016	1018	1020	1022

↑ first element of the array num

- The compiler will allocate the memory for the above two dimensional array row-wise meaning the first element of the second row will be placed after the last element of the first row.

Pointers and Two Dimensional Array

If we want to get the value at any given row, column of the array then we can use the following formula.

arr[i][j] = *(ptr + (i x no_of_cols + j))

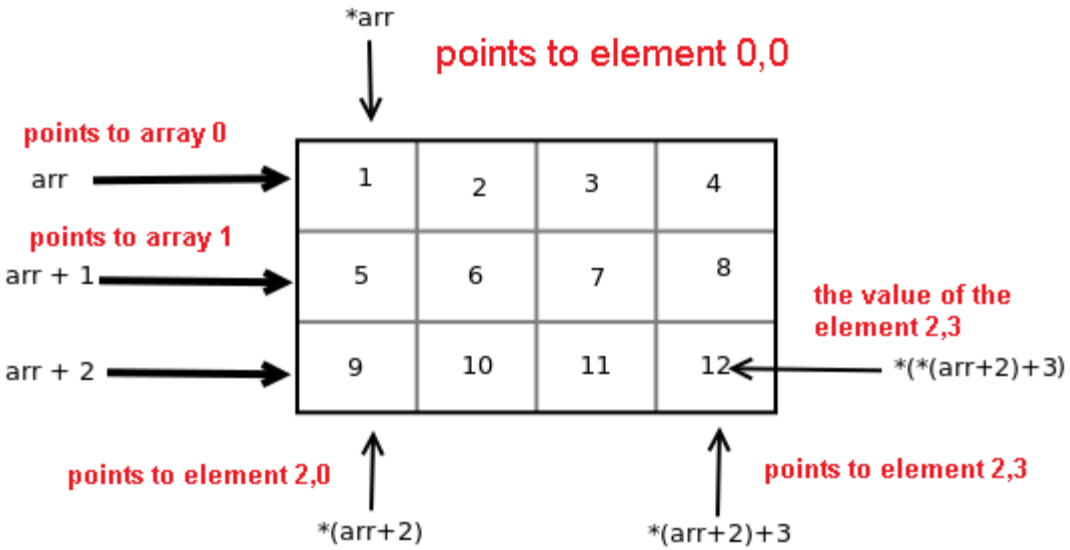
2_DimArrayPtr.c

```c
#include <stdio.h>

int main(void) {
  // 2d array
  int num[3][4] = {
    {1, 2,  3,  4},
    {5, 6,  7,  8},
    {9, 10, 11, 12}
  };
  // pointer ptr pointing at array num
  int *ptr = &num[0][0];
  // other variables
  int
    ROWS = 3,
    COLS = 4,
    TOTAL_CELLS = ROWS * COLS,
    i;
  // print the elements of the array num via pointer ptr
  for (i = 0; i < TOTAL_CELLS; i++)
    printf("%d ", *(ptr + i));

printf("\n");
  return 0;
}
```

2_DimArrayPtr2.c

```c
#include <stdio.h>
int main(void) {
  // 2d array
  int num[3][4] = {
    {1, 2,  3,  4},
    {5, 6,  7,  8},
    {9, 10, 11, 12}
  };
  int
    ROWS = 3,
    COLS = 4,
    i, j;
  // pointer
  int *ptr = &num[0][0];
  // print the element of the array via pointer ptr
  for (i = 0; i < ROWS; i++) {
    for (j = 0; j < COLS; j++) {
       printf("%3d ", *(ptr + (i * COLS + j)));
      // printf("%3d ", *(*(num + i) + j));
    }
    printf("\n");
  }
  return 0;}
```

Pointers and Two Dimensional Array

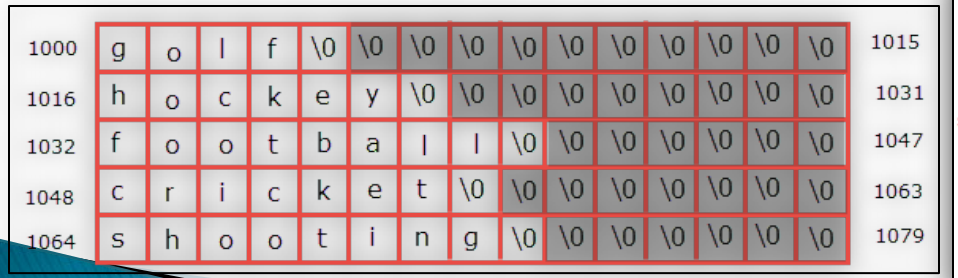


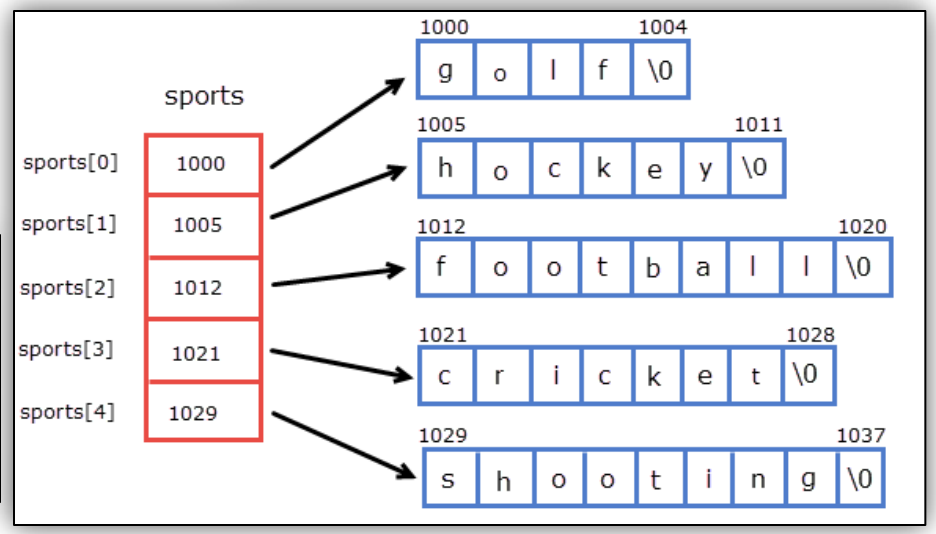
Array of pointers

- Arrays can contain pointer.
- Can access multiple arrays with arrays of pointers.

```
char sports[5][15] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                      };
```

```
char *sports[5] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                      };
//you can even remove the number of rows
```

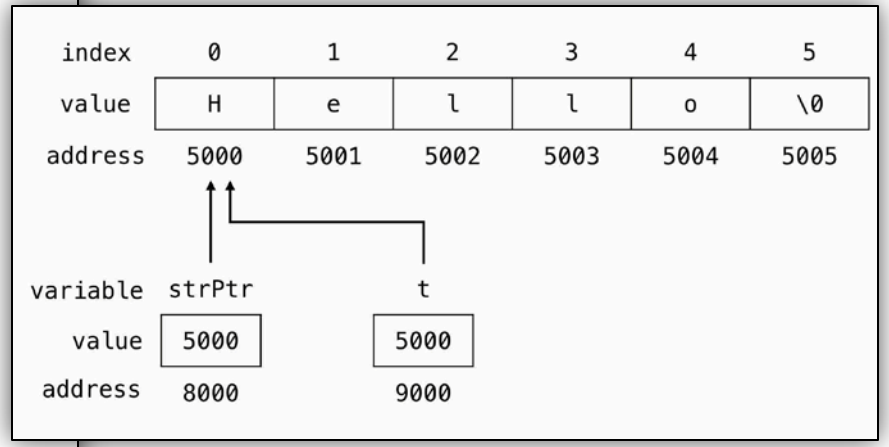




Pointers and Strings

CopyArrayPtr.c

```
1
2   #include <stdio.h>
3
4   int main(void) {
5
6     // pointer variable to store string
7     char *strPtr = "Hello";
8     char x[6]; // empty array
9     // temporary pointer variable
10    char *t,*t2;
11
12    printf("%s\n", strPtr);
13    t = strPtr;
14    t2= x;
15
16    // print the string
17    while(*t != '\0') {
18      *t2=*t; // using the pointers to copy strPtr to x
19      // move the t pointer to the next memory location
20      t++;
21      t2++;
22    }
23      printf("%s\n", x);
24    return 0;
25  }
```



Pointers and multidimensional array

MultiDimArrayPtr.c

```c
#include <stdio.h>
int main(void) {
   int i,j;
   char * SpringSeason[]={"March", "April", "May"};
   char * SummerSeason[]={"June", "July", "August"};
   char * FallSeason[]={"September", "October", "November"};
   char * WinterSeason[]={"December", "January", "February"};

   char ** Sasons[4];// 2D array of strings (3D)
   Sasons[0]=SpringSeason;
   Sasons[1]=SummerSeason;
   Sasons[2]=FallSeason;
   Sasons[3]=WinterSeason;

   for (i = 0; i < 4; i++)
     { for (j = 0; j < 3; j++)
          printf("%15s", Sasons[i][j]);
          //printf("%s \n", *(*(Sasons+i)+j));
          printf("\n");
      }
   return 0;
}
```

Pointers and Functions – Call by Value and Call by Reference

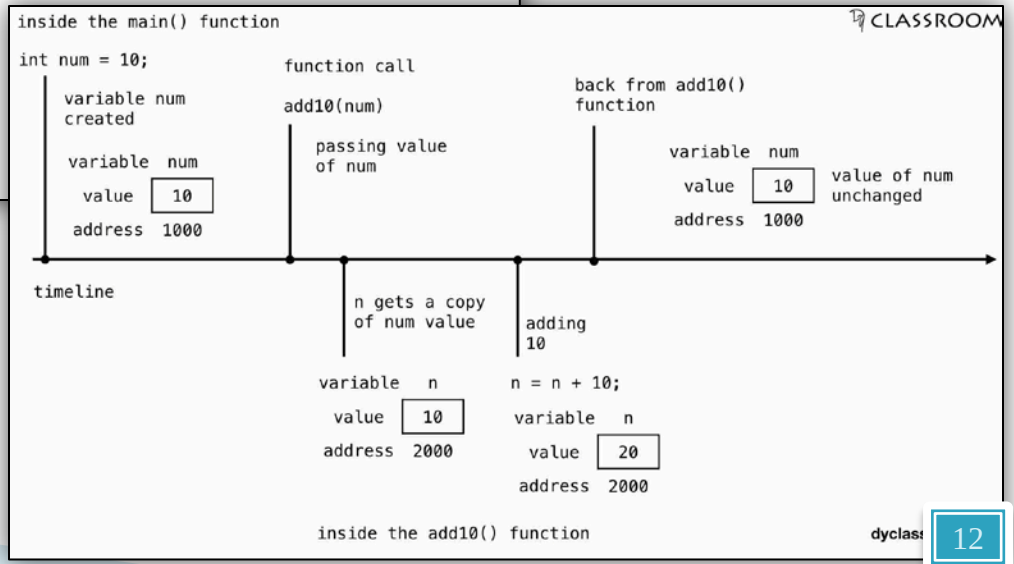
- There are two ways we can pass parameters(arguments) to a function:
  - Call by value – i.e., passing a copy of the variable.
  - Call by reference – this involves pointers, passing the memory address of the variable.
- Call by value (pass by value) will not change the original variable but its copy.
- Using call by reference (pass by reference) the arguments are not passed with their values, but with their addresses. Thus, all modifications on arguments effect the original variable.

Call by value – example

CallByValue.c

```
1   #include <stdio.h>
2   void add10(int);// function declaration
3   int main(void) {
4       int num = 10;// integer variable
5     printf("Value of num before function call: %d\n", num); // print value of num
6
7     add10(num);// pass by value
8     printf("Value of num after function call: %d\n", num);// print value of num
9     return 0;
10  }
11  // function definition
12  void add10(int n) {
13    n = n + 10;
14    printf("Inside add10(): Value %d\n", n);
15  }
```

Output

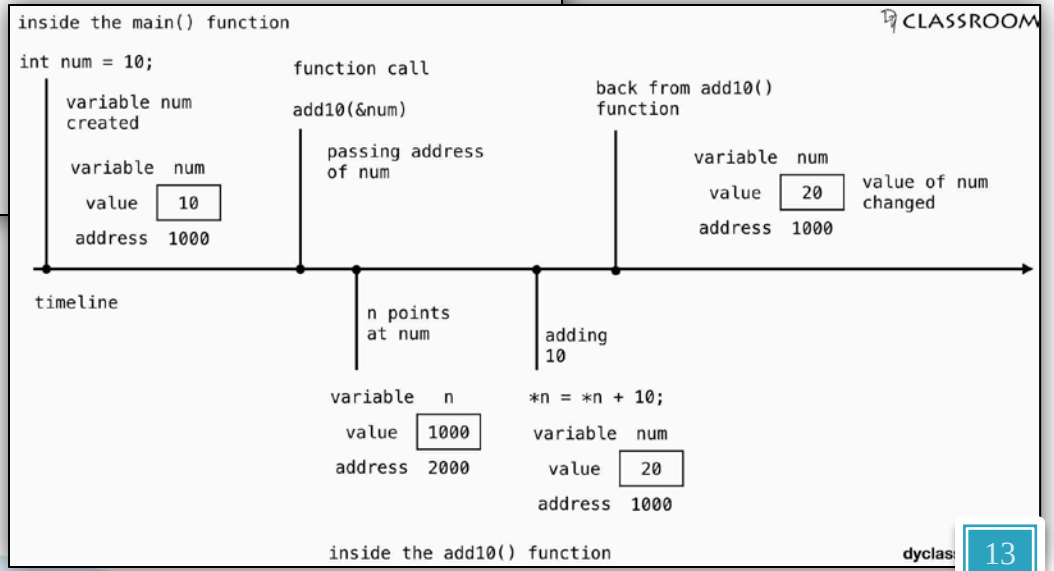


Call by reference – example

CallByReference.c

```c
#include <stdio.h>
void add10(int *);// function declaration
int main(void) {
    int num = 10;// integer variable
  printf("Value of num before function call: %d\n", num); // print value of num

  add10(&num);// pass by reference
  printf("Value of num after function call: %d\n", num);// print value of num
  return 0;
}
// function definition
void add10(int *n) {
  *n = *n + 10;
  printf("Inside add10(): Value %d\n", *n);
}
```

Output



Call by Reference

- If your function has to return more than one value, pass by reference usage is necessary.
- Because return keyword can only return one value from function.
- For example, we want to write a division function that gives division result and remainder.
- In this case, return keyword can only return one value, second value could be returned by reference method.(or you can return both values by reference)

Call by Reference – example 2

CallByReferenceDiv.c

```c
#include<stdio.h>
void div(int , int , int *, int *);

main() {
    int a = 76, b = 10;
    int q, r;
    div(a, b, &q, &r);
    printf("Quotient is: %d\nRemainder is: %d\n", q, r);
}

void div(int a, int b, int *quotient, int *remainder) {
    *quotient = a / b;
    *remainder = a % b;
}
```

CallByReferenceDiv2.c

```c
#include<stdio.h>
int div(int , int , int *);

main() {
    int a = 76, b = 10;
    int q, r;
    q= div(a, b, &r);
    printf("Quotient is: %d\nRemainder is: %d\n", q, r);
}

int div(int a, int b, int *remainder) {

    *remainder = a % b;
    return a / b;
}
```

Thanks ☺