Pointer in C

Lecture-21

Dr. Asif Uddin Khan

C Pointers

- Pointers (pointer variables) are special variables that are used to store addresses of other variables.

How to find address of a variable?

& operator is used with the variable to find the address of a variable.

Example: &var

Example to print address of a variable

==========

```
#include <stdio.h>
int main()
{
  int var = 5;
  printf("var: %d\n", var);

  // Notice the use of & before var
  printf("address of var: %p", &var);
  return 0;
}
```

Output

```
var: 5
address of var: 2686778
```

C Pointers

- The pointer is a variable which stores the address of another variable.
- This variable can be of type int, char, array, function, or any other pointer.
- **Pointer Syntax**

  int* p;

  or

  int *p;

Example:

```
int n = 10;
int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.
```

Declaring a pointer

- The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

Example

- **int*** a;//pointer to int
- **char** *c;//pointer to char

Example of declaring pointers.

- int *p1, p2;
- Here, we have declared a pointer p1 and a normal variable p2.

Assigning addresses to Pointers

Example

```
int *pc, c;
c = 5;
pc = &c;
```

- Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

Get Value of the variable Pointed by Pointers

- To get the value of the thing pointed by the pointers, we use the * operator.

Example:

```
int *pc, c;
c = 5;
pc = &c;
printf("%d", *pc); // Output: 5
```

Note: In the above example, pc is a pointer, not *pc. You cannot and should not do something like *pc = &c;

Changing Value Pointed by Pointers

```
int* pc, c; c = 5;
pc = &c;
c = 1;
printf("%d", c); // Output: 1
printf("%d", *pc); // Ouptut: 1
```

We have assigned the address of c to the pc pointer. Then, we changed the value of c to 1. Since pc and the address of c is the same, *pc gives us 1.

Example-2

```
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc); // Ouptut: 1
printf("%d", c); // Output: 1
```

We have assigned the address of c to the pc pointer. Then, we changed *pc to 1 using *pc = 1;. Since pc and the address of c is the same, c will be equal to 1.

Example-3

- int* pc, c, d;
- c = 5; d = -15;
- pc = &c;
- printf("%d", *pc); // Output: 5
- pc = &d; printf("%d", *pc); // Ouptut: -15

Initially, the address of c is assigned to the pc pointer using pc = &c;. Since c is 5, *pc gives us 5. Then, the address of d is assigned to the pc pointer using pc = &d;. Since d is -15, *pc gives us -15.

Working of Pointers

```c
#include <stdio.h>
int main()
{
    int* pc, c;

    c = 22;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);  // 22

    pc = &c;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 22

    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 11

    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c); // 2
    return 0;
}
```

Output

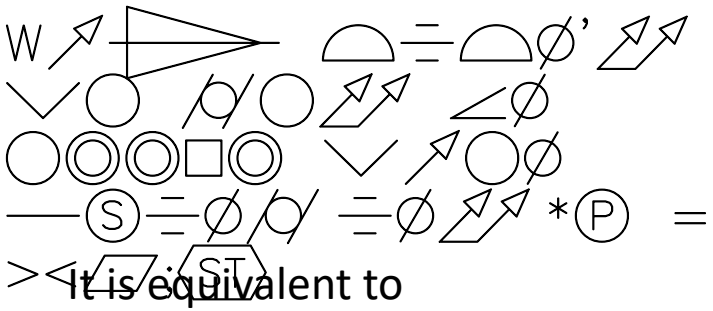
```
Address of c: 2686784
Value of c: 22

Address of pointer pc: 2686784
Content of pointer pc: 22

Address of pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2
```

Common mistakes when working with pointers

- int c, *pc; // pc is address but c is not
- pc = c; // Error // &c is address but *pc is not
- *pc = &c; // Error // both &c and pc are addresses
- pc = &c; // both c and *pc values *pc = c;

Here's an example of pointer syntax beginners often find confusing.

```
#include <stdio.h>
int main() {
int c = 5;
int *p = &c;
printf("%d", *p); //5
return 0;
}
```



In both cases, we are creating a pointer p (not *p) and assigning &c to it. To avoid this confusion, we can use the statement like following:
int* p = &c;

How to Use Pointers?

Steps

1. Declare a pointer variable
2. Assign the address of a variable to a pointer
3. Finally access the value at the address available in the pointer variable.

```c
//How to use pointer
#include <stdio.h>
int main () {
   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */
   ip = &var;  /* store address of var in pointer variable*/
   printf("Address of var variable: %x\n", &var  );
   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );
   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );
   return 0;
}
```

NULL Pointers

- A pointer that is assigned NULL is called a **null** pointer.
- The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program

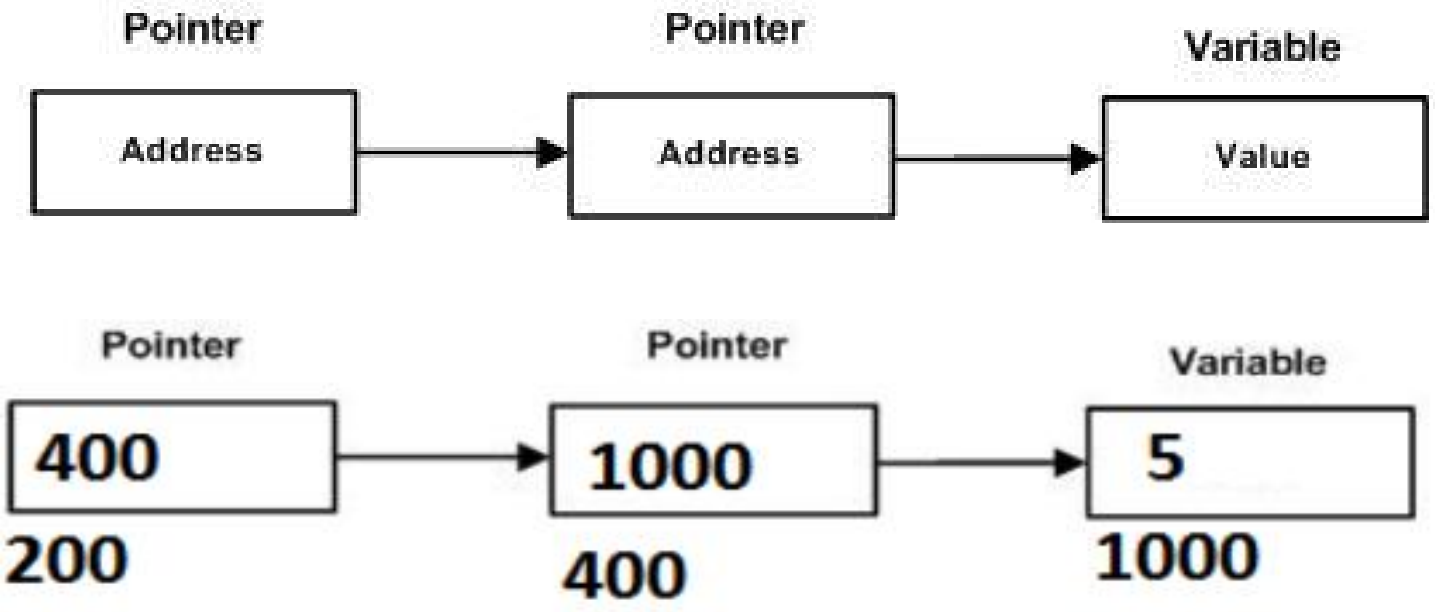
```
#include <stdio.h>
int main () {
   int  *ptr = NULL;
   printf("The value of ptr is : %x\n", ptr  );
   return 0;
}
```

Output:

The value of ptr is 0

Pointer to Pointer

- The pointer to pointer is a variable which stores the address of another pointer.
- The first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



Declaration of a pointer to a pointer of integer type

- int **var;

```c
#include <stdio.h>

int main () {

   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   return 0;
}
```

Output

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Pointers and Functions

Pass Addresses

- In C programming, it is also possible to pass addresses as arguments to functions.
- To accept these addresses in the function definition, we can use pointers. It's because pointers are used to store addresses.

Example: Pass Addresses to Functions

(Swapping using call by reference)

```c
// program for swapping
#include <stdio.h>
void swap(int *n1, int *n2);
int main()
{
    int num1 = 5, num2 = 10;
    // address of num1 and num2 is passed
    swap( &num1, &num2);
    printf("num1 = %d\n", num1);
    printf("num2 = %d", num2);
    return 0;
}
void swap(int* n1, int* n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Output:

num1 = 10

num2 = 5

Example 2: Passing Pointers to Functions

```c
// passing pointer as function arguments
#include <stdio.h>
void addOne(int* ptr) {
   (*ptr)++; // adding 1 to *ptr
}
int main()
{
   int* p, i = 10;
   p = &i;
   addOne(p);
   printf("%d", *p); // 11
   return 0;
}
```

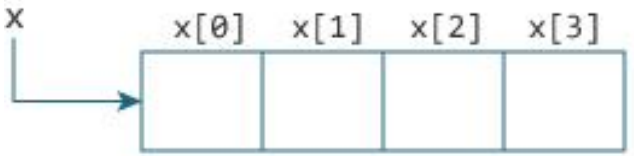
Relationship Between Arrays and Pointers

- An array is a block of sequential data. Let's write a program to print addresses of array elements.

```c
#include <stdio.h>
int main() {
    int x[4];
    int i;
    for(i = 0; i < 4; ++i) {
        printf("&x[%d] = %p\n", i, &x[i]);
    }
    printf("Address of array x: %p", x);
    return 0;
}
```

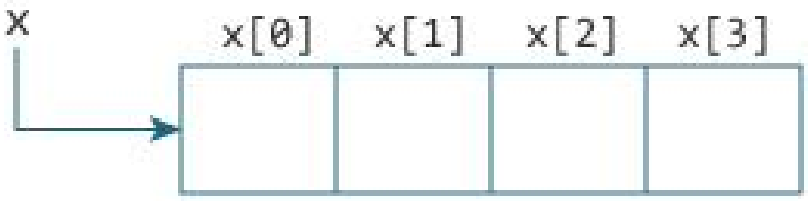
Output

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```



There is a difference of 4 bytes between two consecutive addresses. It is because the size of int is 4 bytes (on my compiler).
Notice that, the address of &x[0] and x is the same. It's because the array name x points to the first element of the array.

Relationship Between Arrays and Pointers



- From the above, it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.
- &x[0] is equivalent to x+0 and x[0] is equivalent to *(x+0)

Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example 1: Pointers and Arrays

```c
// Array and pointer
#include <stdio.h>
int main() {
   int i, x[6], sum = 0;
   printf("Enter 6 numbers: ");
   for(i = 0; i < 6; ++i) {
   // Equivalent to scanf("%d", &x[i]);
        scanf("%d", x+i);

   // Equivalent to sum += x[i]
        sum += *(x+i);
   }
   printf("Sum = %d", sum);
   return 0;
}
```

Output

```
Enter 6 numbers:  2
 3
 4
 4
 12
 4
Sum = 29
```

Example 2: Arrays and Pointers

```
#include <stdio.h>
int main() {
   int x[5] = {1, 2, 3, 4, 5};
   int* ptr;
   // ptr is assigned the address of the third element
   ptr = &x[2];
   printf("*ptr = %d \n", *ptr);   // 3
   printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
   printf("*(ptr-1) = %d", *(ptr-1));  // 2
   return 0;
}
Output
=======
*ptr = 3
*(ptr+1) = 4
*(ptr-1) = 2
```

Sort an array using pointer

```
for(i=0;i<n;i++)
   {
    for(j=i+1;j<n;j++)
    {
       if( *(a+i) > *(a+j))
       {
      tmp = *(a+i);
      *(a+i) = *(a+j);
      *(a+j) = tmp;
       }
    }
   }
```

How to return a Pointer from a Function in C

```c
// Function returning pointer
int* fun()
{
    int A = 10;
    return (&A);
}
//The below program will give segmentation
//fault since 'A' was local to the function

// Driver Code
int main()
{
    // Declare a pointer
    int* p;
    // Function call
    p = fun();
    printf("%p\n", p);
    printf("%d\n", *p);
    return 0;
}
```

Explanation: The main reason behind this scenario is that compiler always make a stack for a function call. As soon as the function exits the function stack also gets removed which causes the local variables of functions goes out of scope.

Output

```
asif@asif-VirtualBox:~/sit$ g++ ptrfunr.c
ptrfunr.c: In function 'int* fun()':
ptrfunr.c:4:5: warning: address of local variabl
e 'A' returned [enabled by default]
asif@asif-VirtualBox:~/sit$ ./a.out
p=0xbfd79d44
*p=-1217226472
```

How to return a Pointer from a Function in C

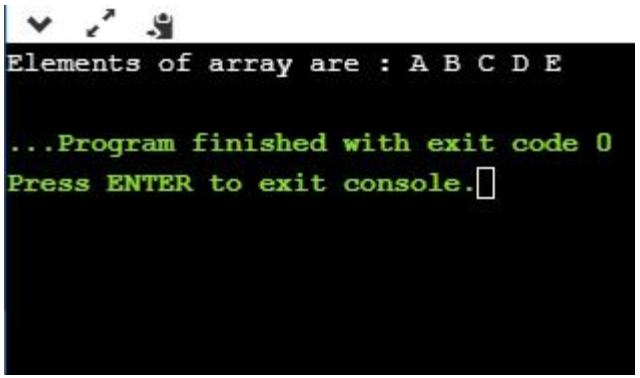
```c
// C program to illustrate the concept of
// returning pointer from a function
#include <stdio.h>
// Function that returns pointer
int* fun()
{
    // Declare a static integer
    static int A = 10;
    return (&A);
}
// Driver Code
int main()
{
    // Declare a pointer
    int* p;
    // Function call
    p = fun();
    // Print Address
    printf("%p\n", p);
    // Print value at the above address
    printf("%d\n", *p);
    return 0;
}
```

Output

```
asif@asif-VirtualBox:~/sit$ g++ ptrfunr.c
asif@asif-VirtualBox:~/sit$ ./a.out
p=0x804a014
*p=10
```

Passing array to a function as a pointer

```c
#include <stdio.h>
void printarray(char *arr)
{
   printf("Elements of array are : ");
   for(int i=0;i<5;i++)
   {
      printf("%c ", arr[i]);
   }
}
int main()
{
  char arr[5]={'A','B','C','D','E'};
  printarray(arr);
  return 0;
}
```



How to return an array from a function

```c
#include <stdio.h>
int *getarray()
{
    int arr[5];
    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &arr[i]);
    }
    return arr;
}
int main()
{
  int *n;
  n=getarray();
  printf("\nElements of array are :");
  for(int i=0;i<5;i++)
    {
        printf("%d", n[i]);
    }
    return 0;
}
```

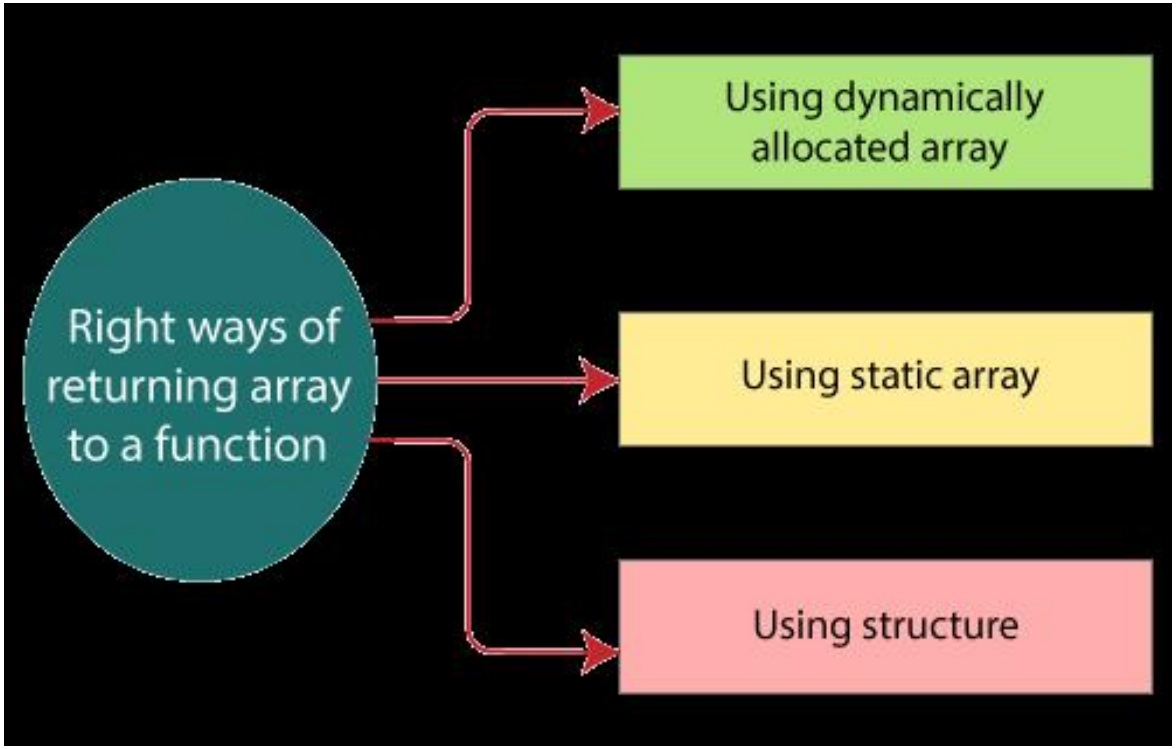
- A function can return an array by returning pointer pointing to the array as follows

```
input
main.c:27:12: warning: function returns address of local variable [-Wreturn-local-addr]
Array inside function: 1
2
3
4
5
Array outside function:
Segmentation fault (core dumped)
```

How to return an array from a function

There are three right ways of returning an array to a function:

1. Using dynamically allocated array
2. Using static array
3. Using structure



Returning a pointer to array by passing an array

```c
#include <stdio.h>
int *getarray(int *a)
{

    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &a[i]);
    }
    return a;
}
int main()
{
  int *n;
  int a[5];
  n=getarray(a);
  printf("\nElements of array are :");
  for(int i=0;i<5;i++)
    {
        printf("%d", n[i]);
    }
    return 0;
}
```

Strings and Pointers

- string is an array of characters, the pointers can be used in the same way they were used with array.
- There are various advantages of using pointers to point strings.
- Let us consider the following example to access the string via the pointer.

```
#include<stdio.h>
void main ()
{
    char s[5] = "asif";
    char *p = s; // pointer p is pointing to string s.
    printf("%s",p); // the string asif is printed if we print p.
}

Output
======
asif
```

Use of pointers to copy the content of a string into another

```c
#include<stdio.h>
int main ()
{
    char *p = "asif";
    printf("String p: %s\n",p);
    char *q;
    printf("copying the content of p into q...\n");
    q = p;
    printf("String q: %s\n",q);
   return 0;
}
```

Program to print string using pointer

```c
/*  C program to Print string using pointers  */
#include <stdio.h>
int main()
{
    char str[100];
    char *ptr;
    printf("Enter any string :: ");
    scanf("%s",str);
    //assign address of str to ptr
    ptr=str;
    printf("\nThe entered string is :: ");

    while(*ptr!='\0')
        printf("%c",*ptr++);
    return 0;
}
```

Example

```
#include<stdio.h>
int main(){
char str[100];
char* pp;
char *pt=(char *)"asif uddin khan";
pp=pt;
char *ptr;
printf("Enter a string:");
scanf("%s",str);
ptr=str;
printf("The string ptr is:");
while(*ptr!='\0')
printf("%c",*ptr++);
printf("\n");
printf("The string pt is:");
while(*pt!='\0')
printf("%c",*pt++);
printf("\n");
printf("after copy pp is :%s",pp);
printf("\n");
return 0;
}
```

Advantage of pointer

- 1) Pointer **reduces the code** and **improves the performance**, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.
- 2) We can **return multiple values from a function** using the pointer.
- 3) It makes you able to **access any memory location** in the computer's memory.

References

1. C programming by E Balaguruswami
2. Programming C by Y. kanitkar
3. Programming C by Denis Ritchie
4. NPTEL Lecture note of Dr. Partha Pratim Das, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur.
5. https://docs.oracle.com/cd/E18752_01/html/817-6223/chp-typeopexpr-2.html
6. https://data-flair.training/blogs/escape-sequence-in-c/
7. Internet source

References

1. NPTEL Lecture note of Dr. Partha Pratim Das, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur.
2. https://docs.oracle.com/cd/E18752_01/html/817-6223/chp-typeopexpr-2.html
3. https://data-flair.training/blogs/escape-sequence-in-c/
4. Internet source