PROGRAMMING

Subject Title: Programming in 'C'
Subject Code: CAM203-1C

Unit-3 Pointers

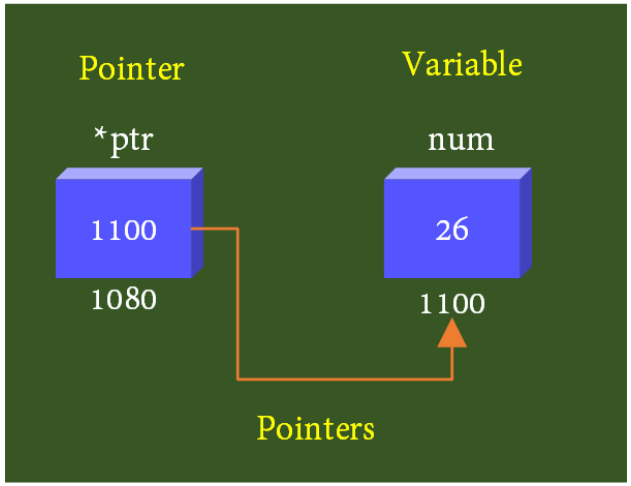
Prepared by Ankit Rami(AR)

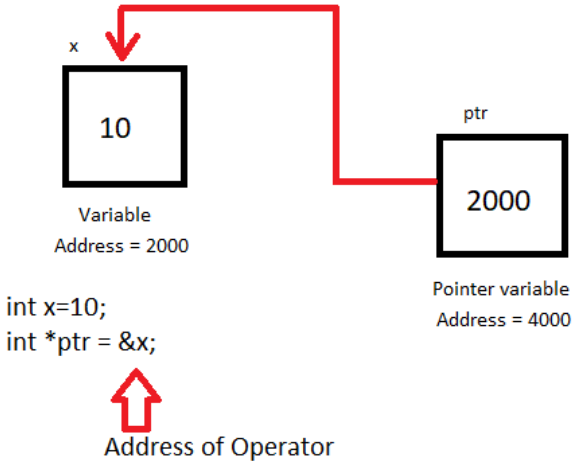
Any Query Contact – 8460467193

Email – ankitramiblog@gmail.com

Introduction for Pointers

- ✓ The pointer in C language is a variable which stores the address of another variable.
- ✓ This variable can be of type int, char, array, function, or any other pointer.
- ✓ A pointer variable points to a data type (like int) of the same type, and is created with the * operator
- ✓ **"A pointer is a variable that stores the memory address of another variable as its value."**





Concept and Application of Pointers

- ✓ For passing the argument by using references.
- ✓ For accessing the elements of an array.
- ✓ For dynamic memory allocation by using malloc() and calloc() functions.Malloc and Calloc functions are used for the allocation of memory during the runtime of a C program.
- ✓ Used in arrays, functions to improve the performance of code.
- ✓ For returning multiple values.
- ✓ Implementing a data structure.
- ✓ For doing system-level programming.

Features of Pointers

- ✓ It saves the space of memory.
- ✓ The execution time of code is faster when using a pointer since the data are manipulated by using an address, which is also used for direct access to the memory location.
- ✓ By using pointers, memory can be accessed efficiently because a pointer is used for assigning as well as releasing memory. Hence it can be concluded that the memory of the pointers is located dynamically
- ✓ It can be used with a data structure for representing two or multi-dimensional arrays.
- ✓ By using a pointer, an array of any type can be accessed.
- ✓ Has a feature for file handling.
- ✓ Can allocate memory dynamically.
- ✓ If any pointer is declared in the base class, it could access the object of the derived class, however, the opposite of that is not possible.

Accessing the Address of a Variable

- ✓ C Memory Address: The location of a byte of data in memory
- ✓ Memory Address types in C: unsigned long int or uintptr_t
- ✓ As we know that a pointer is a special type of variable that is used to store the memory address of another variable.
- ✓ A normal variable contains the value of any type like **int**, **char**, **float** etc, while a pointer variable contains the memory address of another variable.
- ✓ **Steps:**
  1. Declare a normal variable, assign the value
  2. Declare a pointer variable with the same type as the normal variable
  3. Initialize the pointer variable with the address of normal variable
  4. Access the value of the variable by using asterisk (*) - it is known as **dereference operator**

Simple Example of Pointers

```c
#include <stdio.h>
int main()
{
   int a=11;
   int* p=&a;  // Pointer declaration
   // Output the memory address for pointer
   printf("Memory Address- %p\n", p);
   // Output the value of a
   printf("Value- %d\n", *p);
   return 0;
}
```

```
Memory Address- 000000000062FE14
Value- 11

--------------------------------
Process exited after 0.008473 seconds
with return value 0
Press any key to continue . . .
```

📣 Declaring and Initializing Pointers

- **Pointer Declaration**

✓ For pointer declaration in C, you must make sure that the data type you're using is a valid C data type and that the pointer and the variable to which the pointer variable points must have the same data type.

✓ For example, if you want a pointer to point to a variable of data type int, i.e. int var=5 then the pointer must also be of datatype 'int', i.e. int *ptr1=&var. The * symbol indicates that the variable is a pointer. To declare a variable as a pointer, you must prefix it with *.

✓ **Syntax:** datatype *pointer_variableName;

✓ **Example:** int *ptr1;

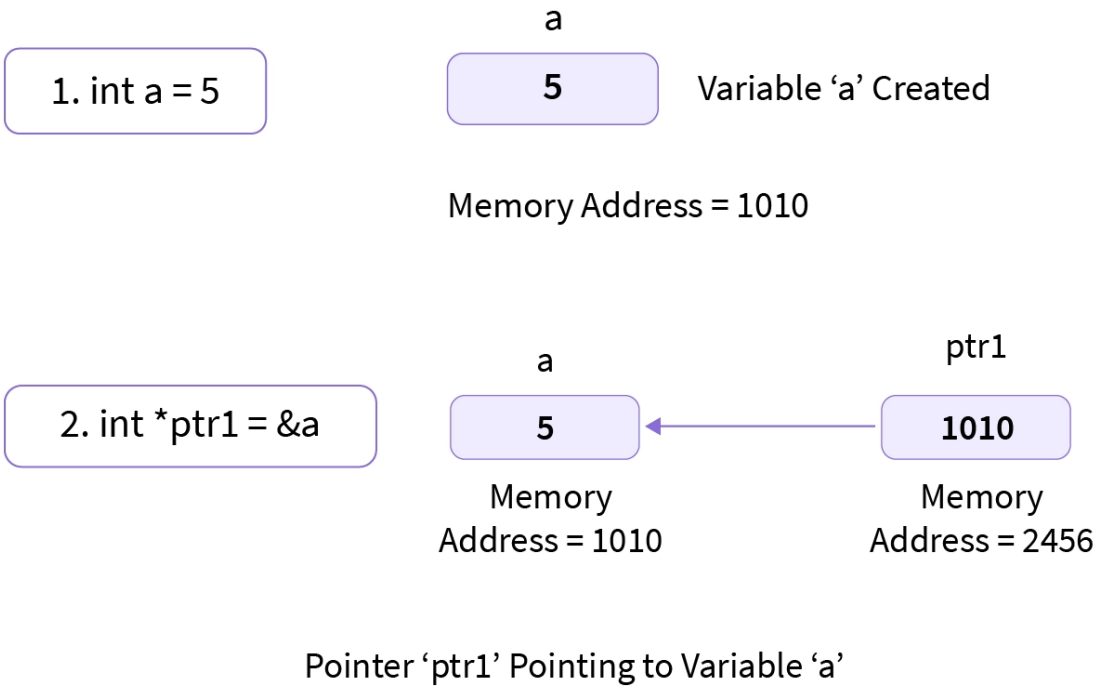
Declaring and Initializing Pointers

- **Pointer Initialize**

✓ 2 ways of initializing a pointer in C

✓ **Method 1**

✓ We make use of the reference operator, i.e. '&' to get the memory address of a variable. It is also known as the address-of-operator.

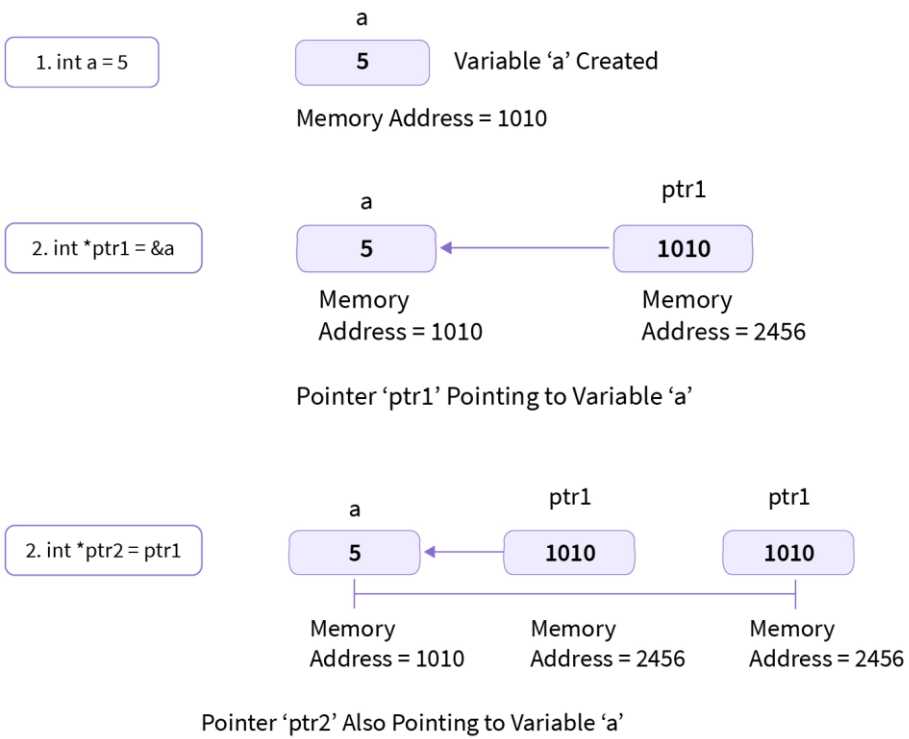


Declaring and Initializing Pointers

- **Pointer Initialize**

✓ **Method 2**

✓ Let us consider the case when we want another pointer to point to the same variable, then, in that case, we can make use of this method to do the same instead of doing method 1 all over again i.e. we simply assign the old pointer to the new pointer.



📣 Declaring and Initializing Pointers

- **Access Pointer**

✓ You can access both the address in memory where the pointer points to and the value it points to as well. To do this, let us first understand what a dereference operator is i.e. '*'.

✓ The process of getting a value from a memory address pointed by a pointer is known as dereferencing. To get the value pointed by a memory address, we utilize the unary operator, *.

✓ **Example:**

```
int a=5;
int *ptr1=&a; //Declaring and Initializing the pointer
printf("%p\n",ptr1); //Prints the memory address that the pointer points to
printf("%d",*ptr1); //Prints the value the pointer points to
```

✓ **Output:**

```
1010
5
```

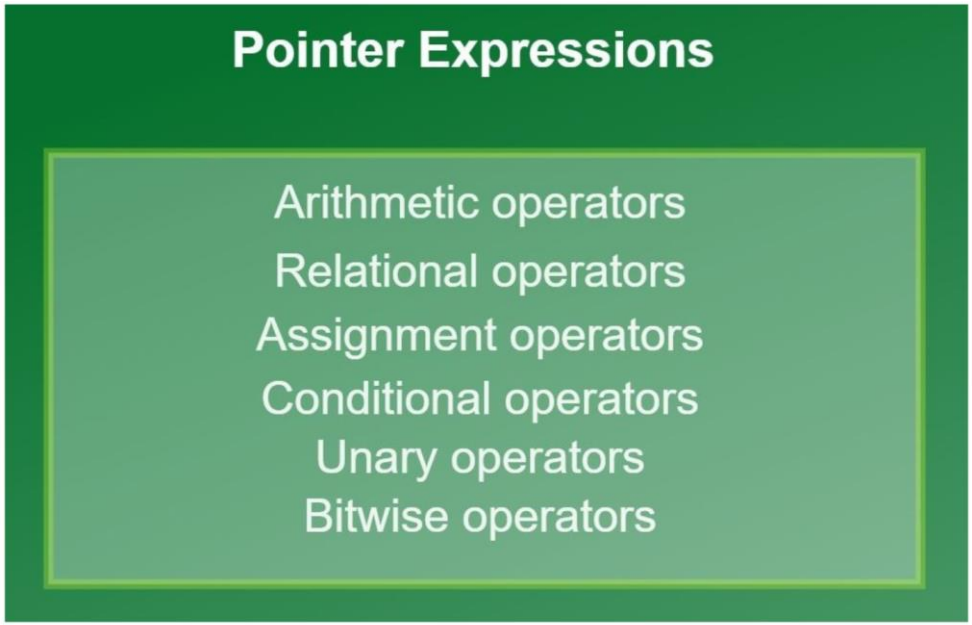
✓ **Reference Link - https://www.scaler.com/topics/c/pointer-declaration-in-c/**

📣 Pointer Expressions in C

- **Access Pointer**

✓ Pointers in C Pointers are used to point to address the location of a variable. A pointer is declared by preceding the name of the pointer by an **asterisk(*)**

✓ The **ampersand (&)** is used to get the address of a variable. We can directly find the location of any identifier by just preceding it with an ampersand(&) sign.



Pointer Expressions in C

✓ **Reference Link -** https://www.geeksforgeeks.org/pointer-expressions-in-c-with-examples/

Declaring and Initializing Pointers

- **Arithmetic Operators**

✓ We can perform arithmetic operations to pointer variables using arithmetic operators. We can add an integer or subtract an integer using a pointer pointing to that integer variable.

Arithmetic Operators

Operator	Context
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Division

Declaring and Initializing Pointers

- **Example of Arithmetic Operators**

```c
#include <stdio.h>
int main()
{
    int a = 20, b = 10;
    int add, sub, div, mul, mod;
    // Pointer variables for variables
    int *ptr_a, *ptr_b;
    // Initialization of pointers
    ptr_a = &a;
    ptr_b = &b;
    // Performing arithmetic Operations on pointers
    add = *ptr_a + *ptr_b;
    sub = *ptr_a - *ptr_b;
    mul = *ptr_a * *ptr_b;
    div = *ptr_a / *ptr_b;
    mod = *ptr_a % *ptr_b;
    // Printing values
    printf("Addition = %d\n", add);
    printf("Subtraction = %d\n", sub);
    printf("Multiplication = %d\n", mul);
    printf("Division = %d\n", div);
    printf("Modulo = %d\n", mod);
    return 0;
}
```

```
Addition = 30
Subtraction = 10
Multiplication = 200
Division = 2
Modulo = 0

--------------------------------
Process exited after 0.007427 seconds with return value 0
Press any key to continue . . .
```

Declaring and Initializing Pointers

- **Assignment Operators**

✓ Assignment operators are used to assign values to the identifiers. There are multiple shorthand operations available. A table is given below showing the actual assignment statement with its shorthand statement.

Assignment Operators

Assignment Operator	Shorthand operation
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b

Declaring and Initializing Pointers

- **Example of Assignment Operators**

D:\Programming-C AR\Unit-3\Assignment Operator in C.cpp - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

Project | Classes | Debug | arithmetic operations using pointer in c.cpp | Relational Operators using Pointer in c.cpp | Assignment Operator in C.cpp

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

    int a = 30;
    int* ptr_a;
    ptr_a = &a;
    // Changing the variable's value using assignment operator
    *ptr_a += 50;
    // Printing value of 'a' after updating its value
    printf("Value of variable a = %d", *ptr_a);
    return 0;
}
```

D:\Programming-C AR\Unit-3\Assignment Operator in C.exe

```
Value of variable a = 80
--------------------------------
Process exited after 0.007531 seconds with re
turn value 0
Press any key to continue . . .
```

Compiler | Resources | Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: D:\Programming-C AR\Unit-3\Assignment Operator in C.exe
- Output Size: 127.9559609375 KiB
- Compilation Time: 0.16s
```

Line: 1 Col: 1 Sel: 0 Lines: 13 Length: 297 Insert Done parsing in 0.015 seconds

Declaring and Initializing Pointers

- **Relational Operators**

✓ Relational operations are often used to compare the values of the variable based on which we can take decisions.

Relational Operators

Operator	Context
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

Declaring and Initializing Pointers

- **Example of Relational Operators**

D:\Programming-C AR\Unit-3\Relational Operators using Pointer in c.cpp - [Executing] - Dev-C++ 5.11

```c
#include <stdio.h>
int main()
{
    int a=20,b=10;
    // Declaring pointer variables
    int* ptr_a;
    int* ptr_b;
    // Initializing pointer variables
    ptr_a = &a;
    ptr_b = &b;
    // Performing relational operations less than operator
    if (*ptr_a < *ptr_b)
    {
        printf("%d is less than %d.", *ptr_a, *ptr_b);
    }
    // Greater than operator
    if (*ptr_a > *ptr_b)
    {
        printf("%d is greater than %d.", *ptr_a, *ptr_b);
    }
    // Equal to
    if (*ptr_a == *ptr_b)
    {
        printf("%d is equal to %d.", *ptr_a, *ptr_b);
    }
    return 0;
}
```

D:\Programming-C AR\Unit-3\Relational Operators using Pointer in c.exe

```
20 is greater than 10.
--------------------------------
Process exited after 0.007646 seconds with return value 0
```

Declaring and Initializing Pointers

- **Conditional Operators**

✓ There is only one mostly used conditional operator in C known as Ternary operator. Ternary operator first checks the expression and depending on its return value returns true or false, which triggers/selects another expression.

✓ **Example:**

```
c = (*ptr1 > *ptr2) ? *ptr1 : *ptr2;
```

✓ As shown in example, assuming *ptr1=20 and *ptr2=10 then the condition here becomes true for the expression, so it'll return value of true expression i.e. *ptr1, so variable 'c' will now contain value of 20.

✓ Considering same example, assume *ptr1=30 and *ptr2=50 then the condition is false for the expression, so it'll return value of false expression i.e. *ptr2, so variable 'c' will now contain value 50.

Declaring and Initializing Pointers

- **Example of Conditional Operators**

```c
#include <stdio.h>
int main()
{
    int a=15,b=20,result;
    int *pa, *pb;
    pa = &a;
    pb = &b;
    result = ((*pa > *pb) ? *pa : *pb);
    printf("%d is the greatest.", result);
    return 0;
}
```

```
20 is the greatest.
--------------------------------
Process exited after 0.005954 seconds with return value 0
Press any key to continue . . .
```

Declaring and Initializing Pointers

- **Unary Operators**

✓ There are mainly two operators which are given as follows.

Unary Operators

Operator	Context
++	Increments by 1
--	Decrements by 1

Declaring and Initializing Pointers

- **Example of Unary Operators**

```c
#include <stdio.h>
int main()
{
    int a = 34;
    int* pa;
    pa = &a;
    printf("Increment:\n");
    printf("Before increment a = %d\n", *pa);
    (*pa)++;
    printf("After increment a = %d", *pa);
    printf("\n\nDecrement:\n");
    printf("Before decrement a = %d\n", *pa);
    (*pa)--;
    printf("After decrement a=%d", *pa);
    return 0;
```

```
Increment:
Before increment a = 34
After increment a = 35

Decrement:
Before decrement a = 35
After decrement a=34
--------------------------------
Process exited after 0.00815 seconds with return value 0
Press any key to continue . . .
```

Declaring and Initializing Pointers

- **Bitwise Operators**

✓ Binary operators are also known as bitwise operators. It is used to manipulate data at bit level. Bitwise operators can't be used for float and double datatype.

Bitwise Operators

Operator	Context
&	bitwise AND
|	bitwise OR
^	bitwise Exculsive-OR
<<	shift left
>>	shift right
~	complement(one's)

Declaring and Initializing Pointers

- **Example of Bitwise Operators**

```c
#include <stdio.h>
int main()
{
    int x, y, ex_or;
    int a = 1, b = 2;
    x = a & b;
    y = a | b;
    ex_or = a ^ b;
    printf("\na AND b = %d", x);
    printf("\na OR b = %d", y);
    printf("\na Exclusive-OR b = %d", ex_or);
    return 0;
}
```

```
a AND b = 0
a OR b = 3
a Exclusive-OR b = 3
--------------------------------
Process exited after 0.009117 s
econds with return value 0
Press any key to continue . . .
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: D:\Programming-C AR\Unit-3\Bitwise Operator in C.exe
- Output Size: 127.95703125 KiB
- Compilation Time: 0.16s
```

Pointers and Arrays

- ✓ Continuous memory locations are allocated for all the elements of the array by the compiler.
- ✓ The base address is the location of the first element in the array.
- ✓ For example, **int a [5] = {10, 20,30,40,50};**

Elements	a[0]	a[1]	a[2]	a[3]	a[4]
Value	10	20	30	40	50
Address	1000	1004	1008	1012	1016

base address

a= &a[0]=1000

- ✓ **Reference Link –**
- ✓ https://www.w3schools.com/c/c_pointers_arrays.php
- ✓ https://www.geeksforgeeks.org/pointer-array-array-pointer/
- ✓ https://www.tutorialspoint.com/explain-the-concepts-of-pointers-and-arrays-in-c-language

Pointers and Arrays

- If ‘p’ is declared as integer pointer, then the array ‘a’ can be pointed by the following assignment –

  p=a

  or

  p=&a[0];

- Each value of ‘a’ is accessed by using p++ to move from one element to another. When a pointer is incremented, its value is increased by the size of the datatype that it points to. This length is called the “scale factor”.
- The relationship between pointer p and variable a is shown below –

  P = &a[0] = 1000

  P+1 = &a[1] = 1004

  P+2 = &a[2] = 1008

  P+3 = &a[3] = 1012

  P+4 = &a[4] = 1016

- Address of an element is calculated using its index and the scale factor of the data type.

Pointers and Arrays

```c
#include<stdio.h>
int main()
{
    int a[5];
    int *p,i;
    printf("Enter 5 Elements");
    for(i=0;i<5;i++)
    {
        scanf("%d",&a[i]);
        p=&a[0];
    }
    printf ("Elements of the array - ");
    for(i=0;i<5;i++)
    {
        printf("%d", *(p+i));
    }
    return 0;
}
```

```
Enter 5 Elements1
2
3
4
5
Elements of the array - 12345
--------------------------------
```

Pointers and Character Strings

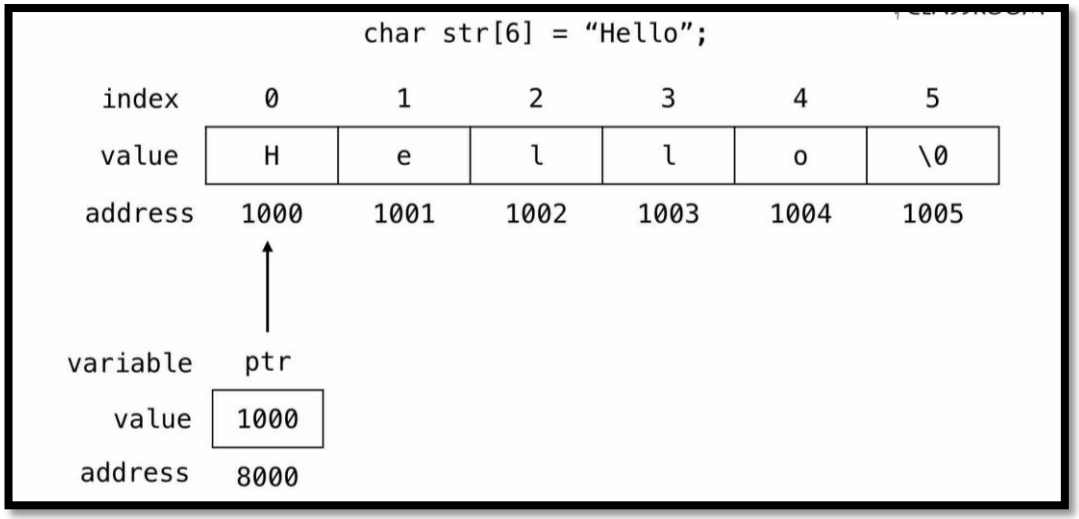
- ✓ We know that a string is a sequence of characters which we save in an array. And in C programming language the \0 null character marks the end of a string.
- ✓ **Creating a string str using char character array of size 6.**
- ✓ char str[6] = "Hello";
- ✓ **Reference Link – https://dyclassroom.com/c/c-pointers-and-strings**

char str[6] = “Hello”;

index	0	1	2	3	4	5
value	H	e	l	l	o	\0
address	1000	1001	1002	1003	1004	1005

Pointers and Character Strings

- ✓ **Creating a pointer for the string**
- ✓ The variable name of the string str holds the address of the first element of the array i.e., it points at the starting memory address.
- ✓ So, we can create a character pointer ptr and store the address of the string str variable in it. This way, ptr will point at the string str.
- ✓ In the following code we are assigning the address of the string str to the pointer ptr.
- ✓ **char *ptr = str;**



📣 Pointers and Character Strings

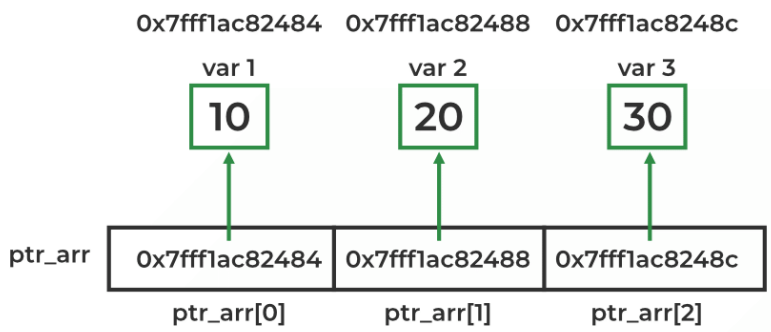
- ✓ **Accessing string via pointer**
- ✓ To access and print the elements of the string we can use a loop and check for the \0 null character.
- ✓ In the following example we are using while loop to print the characters of the string variable str.

```c
#include <stdio.h>
int main()
{
    char str[6] = "Hello";
    char *ptr = str;
    while(*ptr != '\0')
    {
     printf("%c", *ptr);
     // move the ptr pointer to the next memory location
     ptr++;
    }
   return 0;
}
```

```
Hello
--------------------------------
Process exited after 0.006166 seco
nds with return value 0
Press any key to continue . . .
```

Array of Pointer

- ✓ a pointer array is a homogeneous collection of indexed pointer variables that are references to a memory location.
- ✓ It is generally used in C Programming when we want to point at multiple memory locations of a similar data type in our C program. We can access the data by dereferencing the pointer pointing to it.
- ✓ **Syntax:** pointer_type *array_name [array_size];
- ✓ **pointer_type:** Type of data the pointer is pointing to.
- ✓ **array_name:** Name of the array of pointers.
- ✓ **array_size:** Size of the array of pointers.
- ✓ **Reference Link – https://www.geeksforgeeks.org/array-of-pointers-in-c/**



Array of Pointer

```c
#include <stdio.h>
int main()
{
    // declaring some temp variables
    int var1 = 10;
    int var2 = 20;
    int var3 = 30;
    // array of pointers to integers
    int* ptr_arr[3] = { &var1, &var2, &var3 };
    // traversing using loop
    for (int i = 0; i < 3; i++)
    {
        printf("Value of var%d: %d\tAddress: %p\n", i + 1, *ptr_arr[i], ptr_arr[i])
    }
    return 0;
}
```

```
Value of var1: 10       Address: 000000000062FE18
Value of var2: 20       Address: 000000000062FE14
Value of var3: 30       Address: 000000000062FE10

--------------------------------
Process exited after 0.006877 seconds with return value 0
Press any key to continue . . .
```

Array of Pointers to Character

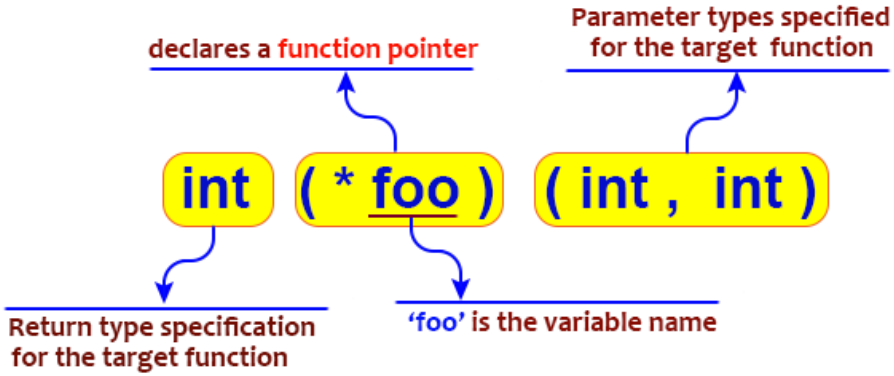
```c
// C Program to print Array of strings without array of pointers
#include <stdio.h>
int main()
{
    char str[3][10] = {"ARR","Rami","Ankit"};
    printf("String array Elements are:\n");
    for (int i = 0; i < 3; i++) {
        printf("%s\n", str[i]);
    }
    return 0;
}
```

```
String array Elements are:
ARR
Rami
Ankit

--------------------------------
Process exited after 0.007257 seconds with return value 0
Press any key to continue . . .
```

Pointers and Functions

- ✓ The C language makes extensive use of pointers, as we have seen. They can be used to provide indirect references to primitive types, to create dynamically sized arrays, to create instances of structs on demand, and to manipulate string data, among other things. Pointers can also be used to create references to functions. In other words, a function pointer is a variable that contains the address of a function.
- ✓ **Reference Link –**
- ✓ https://www.geeksforgeeks.org/function-pointer-in-c/
- ✓ https://www.w3resource.com/c-programming/c-pointers-and-functions.php



Pointers and Functions

```c
#include <stdio.h>
void fun(int a)
{
    printf("Value of a is %d\n", a);
}
int main()
{
    // fun_ptr is a pointer to function fun()
    void (*fun_ptr)(int) = &fun;
    // Invoking fun() using fun_ptr
    (*fun_ptr)(10);
    return 0;
}
```

```
Value of a is 10

--------------------------------
Process exited after 0.007609 seconds with return
 value 0
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: D:\Programming-C AR\Unit-3\Function and Pointer.exe
- Output Size: 128.4736328125 KiB
- Compilation Time: 0.17s
```

Any Query Contact Us

Faculty Name- Ankit Rami

Email – ankitramiblog@gmail.com

Contact No – +91 8460467193

Website - amit.arinfoway.com

Subscribe Our YouTube Channel

https://www.youtube.com/channel/UCWbJh2iQ8w-8nrU0Xpjpw7g