# Fundamentals of Computer Programming

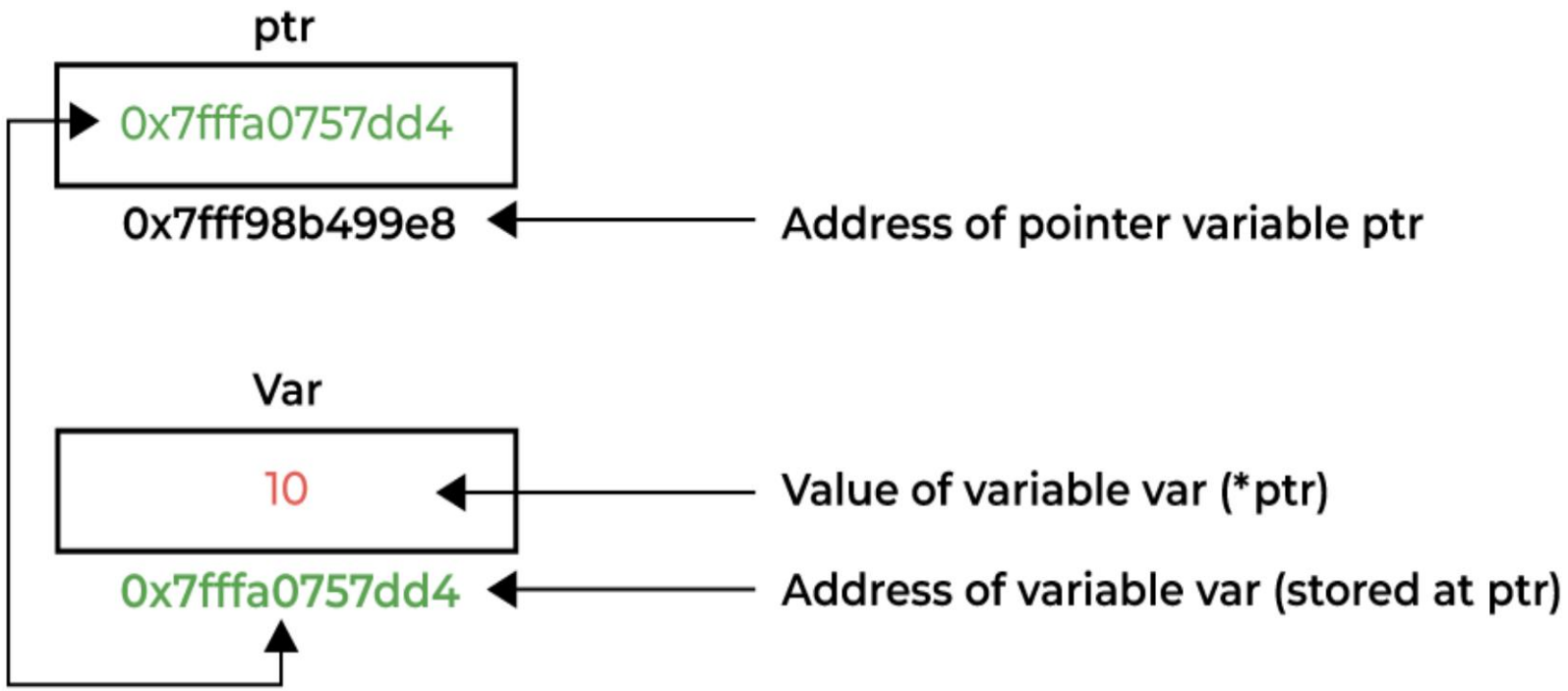
***CS-110***

***Course Instructor:***

***Dr. Momina Moetesum***

# Introduction to Pointers

*Week 13*



# Learning Objectives

01

To define pointers and their purpose in C++

02

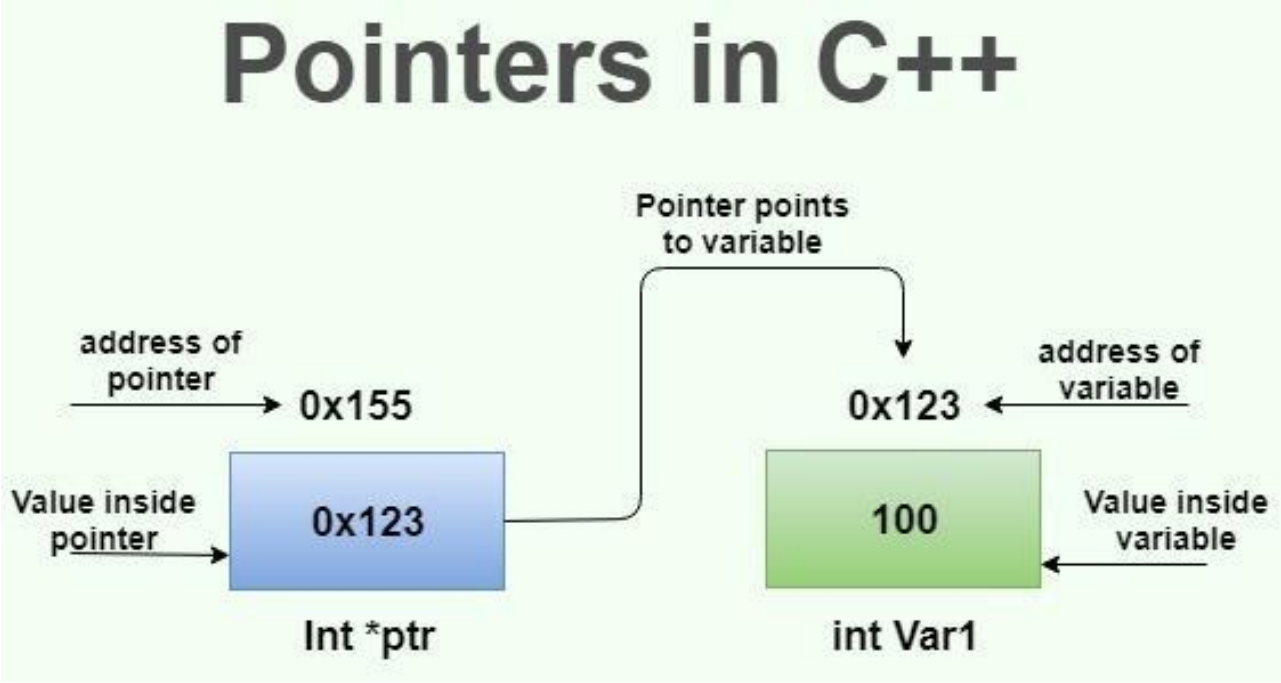
To understand Reference (&) and Dereference (*) operators and their use

03

To use pointers with arrays and functions

# Pointers in C++

- A variable is used to store a value.
- A pointer is used to store an address.
- A pointer is a variable that holds a memory address, usually the location of another variable in memory.
- Like any variable or constant, you must **declare** a pointer before you can work with it.



# Syntax & Semantics

```
#include <iostream>
using namespace std;
int main() {

int *pc, c;
c = 5;
cout << "Address of c (&c): " << &c << endl;
cout << "Value of c (c): " << c << endl << endl;
pc = &c;    // Pointer pc holds the memory address of variable c
cout << "Address that pointer pc holds (pc): "<< pc << endl;
cout << "Content of the address pointer pc holds (*pc): " << *pc << endl<<endl;

c = 11;    // The content inside memory address &c is changed from 5 to 11.
cout << "Address pointer pc holds (pc): " << pc << endl;
cout << "Content of the address pointer pc holds (*pc): " << *pc << endl<<endl;

*pc = 2;
cout << "Address of c (&c): " << &c << endl;
cout << "Value of c (c): " << c << endl << endl;
return 0;

}
```

Output

```
/tmp/6SM3xWPehg.o
Address of c (&c): 0x7fff42484124
Value of c (c): 5

Address that pointer pc holds (pc): 0x7fff42484124
Content of the address pointer pc holds (*pc): 5

Address pointer pc holds (pc): 0x7fff42484124
Content of the address pointer pc holds (*pc): 11

Address of c (&c): 0x7fff42484124
Value of c (c): 2
```

# Declaring a Pointer

- The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address which is of 2-bytes.
- The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

```
#include <iostream>
using namespace std;
int main() {

int *ip; // pointer to a
double *dp; // pointer t
float *fp; // pointer to
char *ch; // pointer to

}
```

# Reference(&) and Dereference(*) Operators

```
int* pointVar, var;
var = 5;

// assign address of var to pointVar
pointVar = &var;

// access value pointed by pointVar
cout << *pointVar << endl;   // Output: 5
```

- Reference operator (&) as gives the addressof a variable.
- To get the value stored in the memory address, we use the dereference operator (*).
- For example: If a number variable is stored in the memory address 0x123, and it contains a value 5.
- The reference (&) operator gives the value0x123, while the dereference (*) operator gives the value 5.

# Pointer Arithmetic

- Some arithmetic operators can be used with pointers:
- Increment and decrement operators ++, --
- Integers can be added to or subtracted from pointers using the operators +, -, +=, and -=
- Each time a pointer is incremented by 1, it points to the memory location of the next element of its base type.
- If “p” is a character pointer then “p++” will increment “p” by 1 byte.
- If “p” were an integer pointer its value on “p++” would be incremented by 2 bytes.

# Pointer Applications

- Pointers can be used in different ways to perform different types of task.
- Some uses, variations and topics of pointer are as follows:
  - Pointer to Arrays
  - Pointer to Pointer
  - Pointer to Functions
  - Pointer with structures
  - Dynamic Memory Allocation
  - Null Pointer

# Common Mistakes

```
int var, *varPoint;

// Wrong!
// varPoint is an address but var is not
varPoint = var;

// Wrong!
// &var is an address
// *varPoint is the value stored in &var
*varPoint = &var;

// Correct!
// varPoint is an address and so is &var
varPoint = &var;

 // Correct!
// both *varPoint and var are values
*varPoint = var;
```

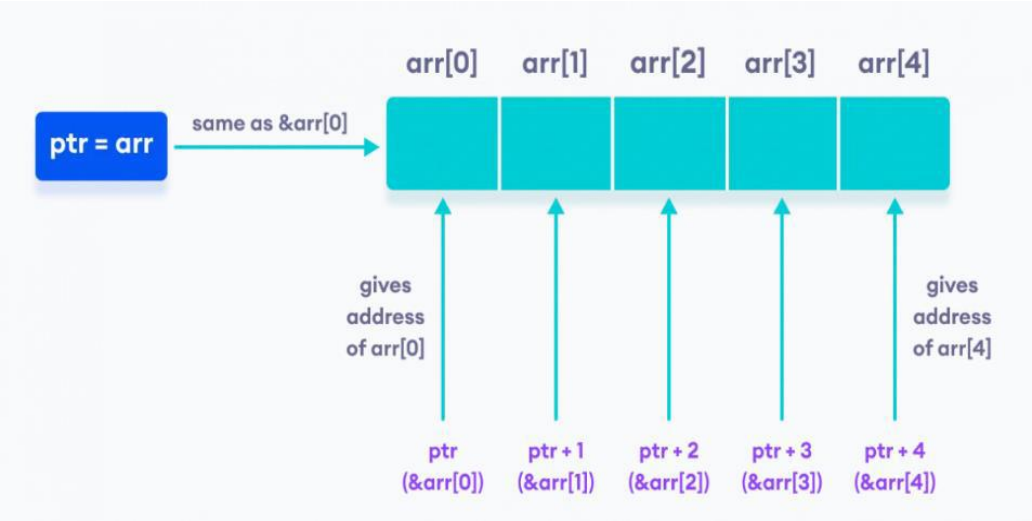
# Arrays and Pointers

- Pointers can be used to point towards an element of an array as well
- We initialize a pointer to an array with the address of array's first element i.e. &arr[0].
- However, &arr[0] == arr, therefore we can use only the name of the array

```
int *ptr;
int arr[5];
ptr = &arr[0];
```

```
int *ptr;
int arr[5];

// store the address of the first
// element of arr in ptr
ptr = arr;
```

# Reference to each element of the Array




```
int *ptr;
int arr[5];
ptr = arr;

ptr + 1 is equivalent to &arr[1];
ptr + 2 is equivalent to &arr[2];
ptr + 3 is equivalent to &arr[3];
ptr + 4 is equivalent to &arr[4];
```

Referencing to Addresses

```
// use dereference operator
*ptr == arr[0];
*(ptr + 1) is equivalent to arr[1];
*(ptr + 2) is equivalent to arr[2];
*(ptr + 3) is equivalent to arr[3];
*(ptr + 4) is equivalent to arr[4];
```

Referencing to Values

main.cpp

Run

```cpp
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;
int main()
{
    float arr[5];
    float *ptr;
    cout << "Displaying address using arrays: " << endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }
    // ptr = &arr[0]
    ptr = arr;
    cout<<"\nDisplaying address using pointers: "<< endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "ptr + " << i << " = "<< ptr + i << endl;
    }
    return 0;
}
```

Output

```
/tmp/6SM3xWPehg.o
Displaying address using arrays:
&arr[0] = 0x7fff2dc9a0f0
&arr[1] = 0x7fff2dc9a0f4
&arr[2] = 0x7fff2dc9a0f8
&arr[3] = 0x7fff2dc9a0fc
&arr[4] = 0x7fff2dc9a100
Displaying address using pointers:
ptr + 0 = 0x7fff2dc9a0f0
ptr + 1 = 0x7fff2dc9a0f4
ptr + 2 = 0x7fff2dc9a0f8
ptr + 3 = 0x7fff2dc9a0fc
ptr + 4 = 0x7fff2dc9a100
```

# Explanation

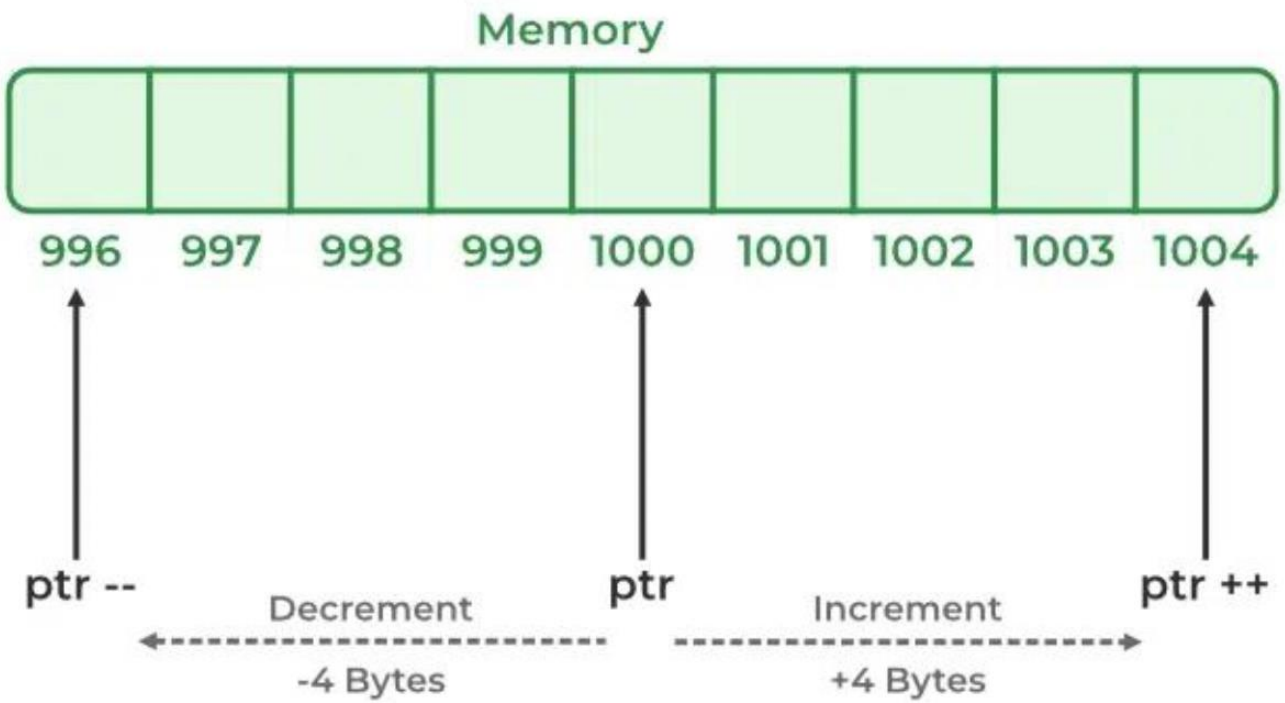
- The address between ptr and ptr + 1 differs by 4 bytes. It is because ptr is a pointer to an int data. And, the size of int is 4 bytes in a 64-bit operating system.
- Similarly, if pointer ptr is pointing to char type data, then the address between ptr and ptr + 1 is 1 byte. It is because the size of a character is 1 byte.

```
/tmp/6SM3xWPehg.o
Displaying address using arrays:
&arr[0] = 0x7fff2dc9a0f0
&arr[1] = 0x7fff2dc9a0f4
&arr[2] = 0x7fff2dc9a0f8
&arr[3] = 0x7fff2dc9a0fc
&arr[4] = 0x7fff2dc9a100
Displaying address using pointers:
ptr + 0 = 0x7fff2dc9a0f0
ptr + 1 = 0x7fff2dc9a0f4
ptr + 2 = 0x7fff2dc9a0f8
ptr + 3 = 0x7fff2dc9a0fc
ptr + 4 = 0x7fff2dc9a100
```

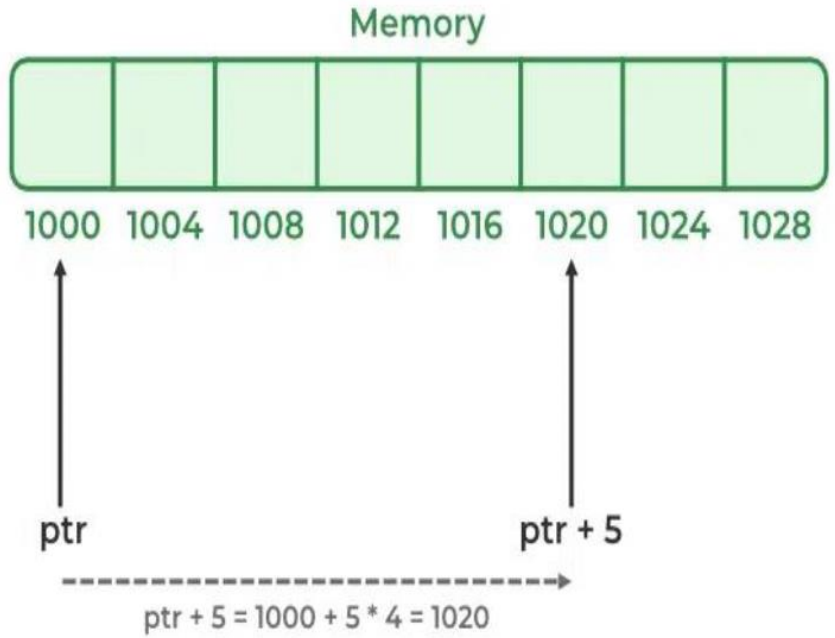
# Using Array name as a Pointer

```cpp
// C++ Program to insert and display data entered by using pointer notation.

#include <iostream>
using namespace std;

int main() {
    float arr[5];

   // Insert data using pointer notation
    cout << "Enter 5 numbers: ";
    for (int i = 0; i < 5; ++i) {

        // store input number in arr[i]
        cin >> *(arr + i) ;

    }

    // Display data using pointer notation
    cout << "Displaying data: " << endl;
    for (int i = 0; i < 5; ++i) {

        // display value of arr[i]
        cout << *(arr + i) << endl ;

    }

    return 0;
}
```

# Pointer Increment & Decrement



# Pointer Addition




```cpp
#include <iostream>
using namespace std;

int main()
{

    int num = 20;
    int* ptr = #

    cout << "Address stored in ptr: " << ptr << endl;

    // Adding the integer value 1 to the pointer ptr
    ptr = ptr + 1;
    cout << "Adding 1 to ptr: " << ptr << endl;

    // Adding the integer value 2 to the pointer ptr
    ptr = ptr + 2;
    cout << "Adding 2 to ptr: " << ptr << endl;

    return 0;
}
```

**Output**

```
Address stored in ptr: 0x7ffdb8634a94
Adding 1 to ptr: 0x7ffdb8634a98
Adding 2 to ptr: 0x7ffdb8634aa0
```

1000 + (5 * 4(size of an integer)) = 1020

# Other Operations

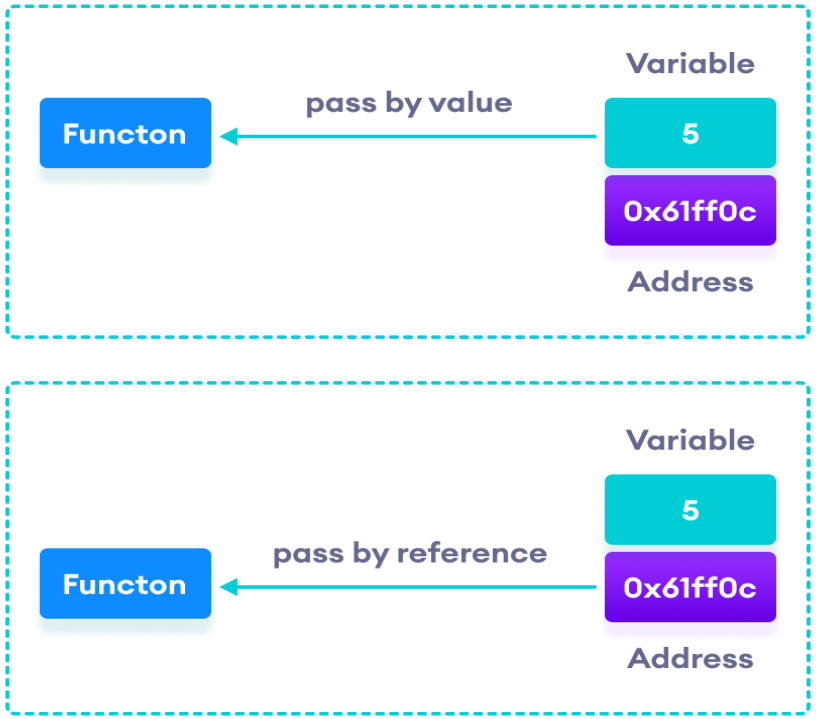
In C++, we can perform a comparison between the two pointers using the relational operators(**>, <, >=, <=, ==, !=**).

We generally use this operation to check whether the two-pointer as pointing to the same memory location or not.

# Pointers and Functions

- Pointers can be used with functions for the following purposes:
- To pass arguments by reference
- To return multiple values from a function
- To pass arrays as arguments to a function
- Pointers to functions

# Call by Reference using Pointers




```cpp
#include <iostream>
using namespace std;
// function prototype with pointers as parameters
void swap(int*, int*);
int main() {
    // initialize variables
    int a = 1, b = 2;
    cout << "Before swapping" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    // call function by passing variable addresses
    swap(&a, &b);
    cout << "\nAfter swapping" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    return 0;
}
// function definition to swap numbers
void swap(int* n1, int* n2) {
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

# Using Pointers to Return Multiple Values from a Function

```
Output

/tmp/6SM3xWPehg.o
10000
10
```

```cpp
#include <iostream>
#include<cmath>
using namespace std;
// function prototype with pointers as parameters
void fun(int n, int* square, double* root)
{
    *square = n * n;
    *root = sqrt(n);
}

int main()
{

    int n = 100;
    int sq;
    double sq_root;
    fun(n, &sq, &sq_root);
    cout << sq <<endl<< sq_root;
    return 0;
}
```

main.cpp

Run

Output

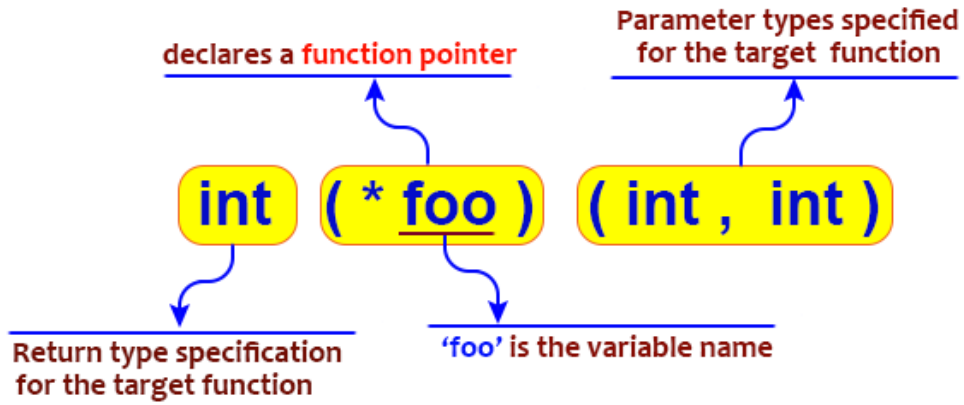
```cpp
#include <iostream>
#include<chrono>
using namespace std;
// function prototype with pointers as parameters
void fun(int *p, int s)
{
    srand(time(0));
    for(int i=0;i<s;i++)
    {
        p[i]=rand()%255;
    }
}
int main()
{
    int n[10]={0};
    int *ptr=n;
    fun(ptr, 10);
    for(int x=0;x<10;x++)
    {
        cout<<n[x]<<endl;
    }
    return 0;
}
```

```
/tmp/6SM3
187
172
123
160
7
126
201
184
230
121
```

# Passing Arrays in Functions using Pointers

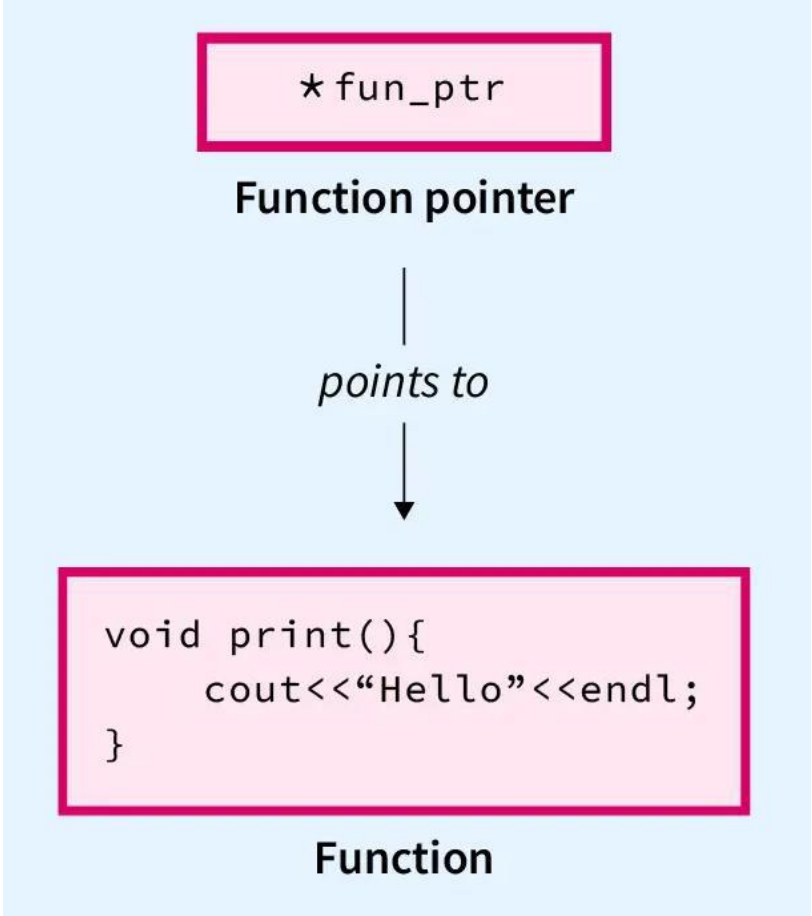
# Pointers to functions

- A pointer to a function points to the address of the executable code of the function.
- You can use pointers to call functions and to pass functions as arguments to other functions.
- You cannot perform pointer arithmetic on pointers to functions.



# Function Pointers

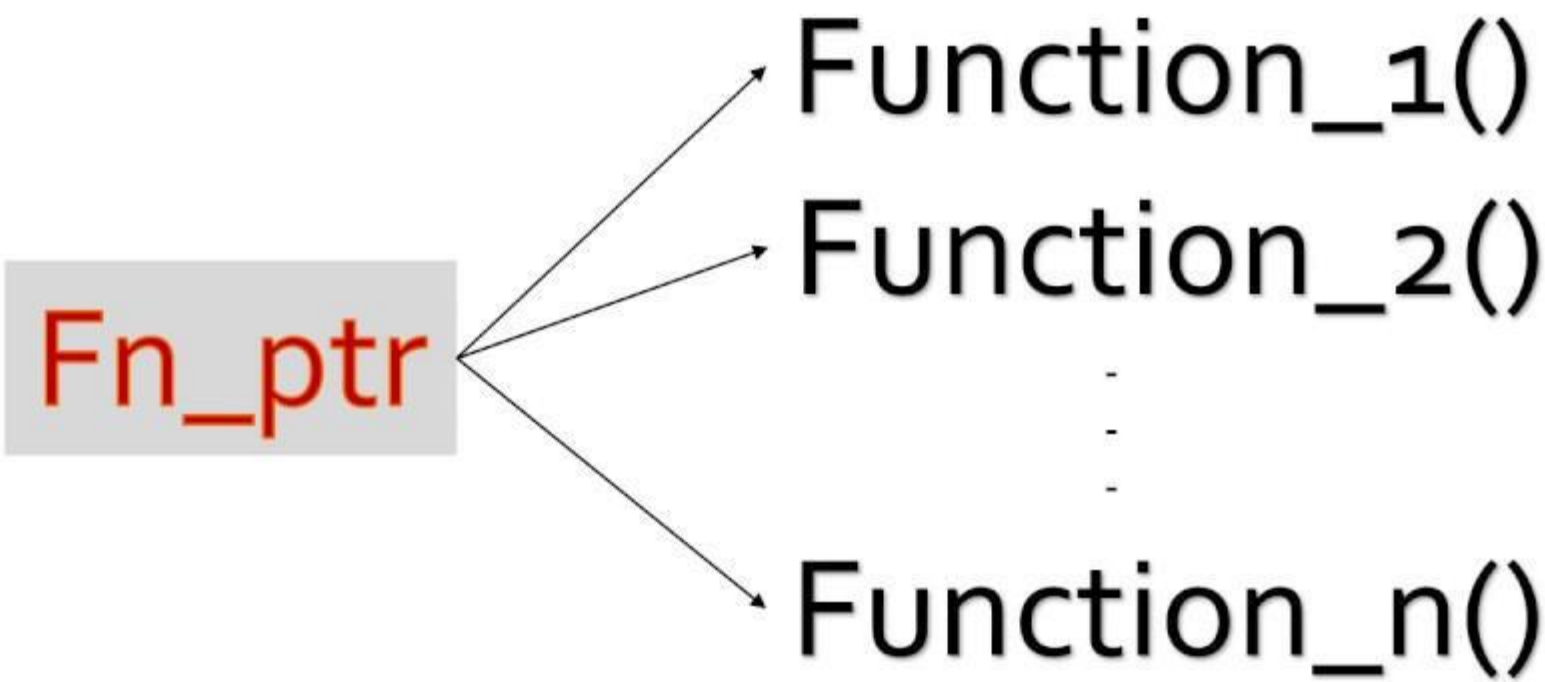
- Pointers to Functions
  - Contains address of function
  - Similar to how Array name is address of first element
  - Function name is starting address of code that defines function
- Function Pointers can be
  - Passed to functions
  - Returned to functions
  - Function name is starting address of code that defines function



# Function Pointer

*Concept*

- ✓ Function/Subroutine/Procedure Pointer
- ✓ A Function Pointer points to code, not data.
- ✓ You can also say "pointers to functions".
- ✓ Holds the starting address of executable code.



# Function Pointers

What would be the address of a function?

<< Program.cpp >>

```
int main()
{
  cout<<"Hello World";
return 0;
}
```

Source code

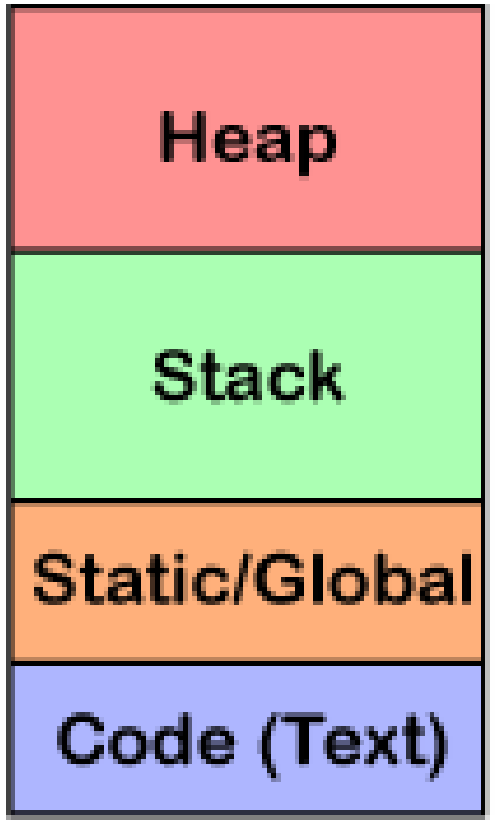
Compiler

<< Program.exe >>

```
10010010
11001100
11100011
10000011
10101010
```

Machine code

Application's memory



# Referencing and Dereferencing of the Function Pointer in C++

- Similar to the pointer used with variables we perform referencing and dereferencing with a function pointer.
- **Referencing:** When pointer is allocated the address of the function to be associated with it then this process is referred to as referencing.
- **Dereferencing:** When we use the (*)operator to get the value stored in the pointer.

# Example

```cpp
// C++ program to implementation
// Function Pointer
#include <iostream>
using namespace std;

int multiply(int a, int b) { return a * b; }

int main()
{
    int (*func)(int, int);

    // func is pointing to the multiplyTwoValues function

    func = multiply;

    int prod = func(15, 2);
    cout << "The value of the product is: " << prod << endl;

    return 0;
}
```

Output

```
The value of the product is: 30
```

# Passing a function pointer as a parameter

- When declaring a function pointer to store the memory address of the function but, when we want to pass the return value to the next function.
- •We have two methods to perform this task. First, either pass the value we got or second pass the function pointer that already exists.

# Example

```cpp
// C++ Program for demonstrating
// function pointer as pointer
#include <iostream>
using namespace std;

const int a = 15;
const int b = 2;

// Function for Multiplication
int multiply() { return a * b; }

// Function containing function pointer
// as parameter
void print(int (*funcptr)())
{
    cout << "The value of the product is: " << funcptr()
         << endl;
}

// Driver Function
int main()
{
    print(multiply);
    return 0;
}
```

# Types of Pointers

- Wild Pointer
- Null Pointer
- Void Pointer
- Dangling Pointer

**For examples and details check the following page:**

**https://www.geeksforgeeks.org/cpp/cpp-pointers/**

# Acknowledgment

- Content of these slides are taken from:
  - https://www.geeksforgeeks.org/
  - https://www.tutorialspoint.com/
  - https://www.programiz.com/
  - https://www.w3schools.com/