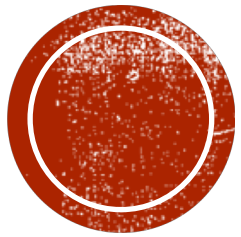
# PAINLESSLY INTRODUCING C++ FOR JAVA PROGRAMMERS

Sridhar Radhakrishnan

University of Oklahoma



# INTRODUCTION TO POINTERS

# POINTERS

- C++ has pointers, whereas Java doesn't have pointers, Java has references.
- **C++ Pointer**: A pointer is a variable that stores a memory address, for the purpose of acting as an alias to what is stored at that address.
- **Java Reference**: A reference is a variable that refers to address of the memory location where the object is stored.
- A pointer is a reference, but a reference is not necessarily a pointer.
- Pointers give direct access to memory itself and make C++ a very powerful language.
- Each pointer variable points to a particular type of variable.
- It is called a pointer because we can think of it as an arrow pointing to the variable which actually occupies that address.

# POINTERS

- For example, we may have a “pointer to int”, which holds the address of memory which is occupied by an int variable.
- **Example:**

```
int a = 10;

int* pInt;

pInt = & a;
```

pInt - pointer of type integer.

*pInt - refer to the same value 10.


- pInt holds the hexadecimal address 10001A of the memory location of the variable **a**.
- We can thus say that pInt points to the address of the memory.

# POINTERS

```java
import java.io.*;
class Rectangle1 {
  double length;
}
public class ReferenceEx {
  public static void main (String[] args) {
    // r1 is reference variable which contain
    // the address of Actual Rectangle Object.
    Rectangle r1 = new Rectangle();
    // r1 and r2 both are referring same object
    Rectangle r2 = r1;
    r1.length = 10;
    r2.length = 20;
    System.out.println ("r1: " + r1.length);
    System.out.println ("r2: " + r2.length);
  }
}
```

**OUTPUT:**

```
r1: 20.0
r2: 20.0
```

```cpp
#include <iostream>
using namespace std;
int main () {
  int r1 = 10; // actual variable declaration.
  int *r2;     // pointer variable
  r2 = &r1;    // store address of r1 in pointer r2
  cout << "Value of r1: ";
  cout << r1 << endl;
  cout << "Address stored in r2: ";
  // prints address stored in *r2 variable
  cout << r2 << endl;
  cout << "Value of r2: ";
  // prints the value of the address in *r2
  cout << *r2 << endl;
  return 0;
}
```

**OUTPUT:**

```
Value of r1: 10
Address stored in r2: 0x7ffee4a7a418
Value of r2: 10
```

# POINTERS

```
#include <iostream>
using namespace std;
int main () {
int firstvalue = 5,
secondvalue = 15;
int * p1, * p2;
p1 = &firstvalue;   // p1 = address of firstvalue
p2 = &secondvalue;  // p2 = address of secondvalue
*p1 = 10;           // value pointed to by p1 = 10
*p2 = *p1;          // value pointed to by p2 = value pointed to by p1
p1 = p2;           // p1 = p2 (value of pointer is copied)
*p1 = 20;          // value pointed to by p1 = 20
cout << ”Firstvalue is " << firstvalue << endl;
cout << ”Secondvalue is " << secondvalue << endl;
return 0;
}
```

**OUTPUT:**

```
Firstvalue is 10
Secondvalue is 20
```

# POINTERS

```
#include <iostream>
using namespace std;
int main () {
  int iVal1 = 10, iVal2 = 20;      // a normal int variables
  int *pInt;                       // a pointer to int – right and here it doesn't point to anything
  cout << iVal1 << endl ;          // outputs 10
  cout << iVal2 << endl;           // outputs 20
  pInt = &iVal1;                   // now pInt contains the address of the memory referred to by iVal1
 *pInt = 50;                       // *pInt is a reference to an int, just like iVal1; since *pInt and
                                   // iVal1 refer to the same memory address, changing the value of *pInt
                                   // will also change the value of iVal1
  cout << iVal1 << endl;           // This outputs 50 rather than 10
  pInt = &iVal2;                   // now pInt contains the address of the memory referred to by iVal2
  *pInt = -10;                     // now *pInt and iVal2 refer to the same memory address, so changing the
                          //value of *pInt will also change the value of iVal2, but not the value of iVal1
  cout << iVal2 << endl;           // This outputs -10 rather than 20
  cout << *pInt<< endl;            // This also outputs -10
  return 0;
}
```

# NULL POINTERS

- Every pointer variable always contains some value.
- If the variable is not initialized, that value is random, based on whatever bytes happened to be in memory at the location where the variable is placed.
- If we know what value we need for the variable, we can initialize it to that value.
- If we do not know the value we need when the variable is created, we can initialize it to the special value **NULL**.
- The C++ language recognizes that a **NULL pointer** is **invalid**.

# DYNAMIC MEMORY ALLOCATION OF POINTERS

- The free store consists of all memory allocated to your program but not set aside for purposes such as code, static data, and stack.
- You get memory from the free store for your pointers by making calls to new. It returns the address of the first byte of the allocated memory, and you normally assign that address to a pointer variable.
- You **deallocate memory** using **delete**.

```
#include <iostream>
using namespace std;
  int main () {
  int *pointer;
  pointer = new int;
  *pointer = 24;
  cout << "The Number is : ";
  cout << *pointer;
  delete pointer;
}
```

**OUTPUT:**

```
The Number is : 24
```

# DYNAMIC MEMORY ALLOCATION OF POINTERS

```
#include <iostream>
using namespace std;
int main () {
     int sum = 0;
     int* marks = new int[5];
     cout << "Enter the 5 numbers" << endl;
     for (int i = 0; i < 5; i++)
          cin >> marks[i];
     for (int i = 0; i < 5; i++)  // calculating sum
          sum = sum + marks[i];
     cout << "sum is " << sum << endl;
     delete[] marks;
     return 0;
}
```

**OUTPUT:**

```
Enter the 5 numbers
45 23 5 6 3
sum is 82
```