Introduction to Computing

Lecture 13

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Memory Address, References and Pointers

Memory Address

Getting Memory Address

When a variable is created in C++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.

To access it, use the **&** operator, and the result will represent where the variable is stored:

Example:

```
int a = 2;

cout << &a; // Outputs 0x6dfed4
```

Reference Variable

Reference Variable

A reference variable is a "reference" to an existing variable, and it is created with the & operator:

```
int a = 2;
int &b= a; // reference to a
```

- The reference variable can only be initialized at the time of its creation
- The reference variable returns the address of the variable preceded by the reference sign ‘&’
- The reference variable can never be reinitialized again in the program
- The reference variable can never refer to NULL

Pointers

Pointer Variable

A pointer is a variable that stores the memory address as its value.

- A pointer variable points to a data type of the same type
- It is created with the ***** operator.
- The address of the variable you're working with is assigned to the pointer

Example:

```
int *a ;
int b = 2;
a= &b; // a stores the address of b
```

Pointers

Accessing Memory Address and Value using Pointer Variable

- Pointer variable holds the address of a variable, so its not a problem
- We can also get the value of the variable through pointer, by using the * operator (**the dereference operator**).
- We can also change the value of the variable by using the * operator

Example:

```
int *a ;
int b = 2;
a= &b;        // a stores the address of b
cout << *a;   // using dereference operator we get value of 'b'
*a = 3;       // using dereference operator we set value of 'b'
cout << b;    // we get 3
```

Pointers

Accessing Memory Address and Value using Pointer Variable

- Poi
- W
  op
- W

Exa

Note:

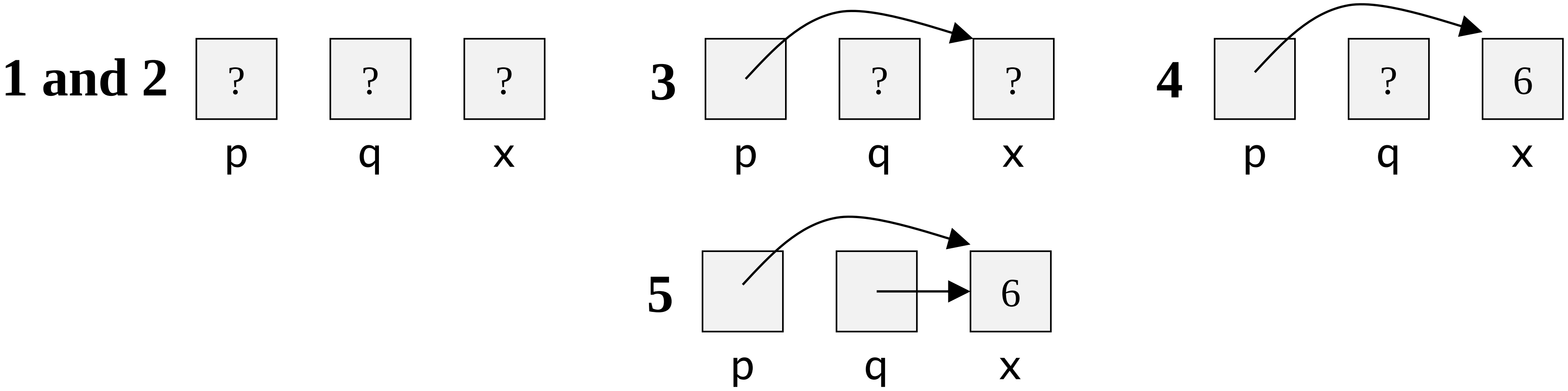
The * sign can be confusing here, as it does two different things in our code:

- When used in **declaration** (string* ptr), it creates a pointer variable.
- When **not used in declaration**, it act as a dereference operator.

```
int
int
a= &
cout << *a;    // using dereference operator we get value of 'b'
*a = 3;        // using dereference operator we set value of 'b'
cout << b;     // we get 3
```

Pointers (Recap)

1.	Pointer variables	`int *p, *q;`
2.	Static allocation	`int x;`
3.	Address-of operator	`p = &x;`
4.	Memory cell to which P points	`*p = 6;`
5.	Pointer operations	`q = p;`



“Pass by Value” and “Pass by Reference”

Pass by Value:

- Makes a copy in memory of the actual parameters

- Use pass by value when you are only **using** the parameter for some computation, not changing it

Pass by Reference:

- Forwards the actual parameters

- Use pass by reference when you are **changing** the parameter passed in the program

“Pass by Value”

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

“Pass by Reference”

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}


int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

“Pass by Value”

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Declaration

“Pass by Reference”

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}


int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Function Declaration

“Pass by Value”

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

int add(int a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}
```

Function Definition

```
int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
return 0;
}
```

“Pass by Reference”

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

int add(int* a)
{
  int b = 0;
  *a = *a + 1;
  b=*a;

  return b;
}
```

Function Definition

```
int main() {
  int x = 0;
  int result = add(&x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

"Pass by Value"   Pass by Pointer ~~"Pass by Reference"~~

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

int add(int a)
{
   int b = 0;
   a = a + 1;
   b=a;

   return b;
}


int main() {
   int x = 0;
   int result = add(x);
   cout << result << endl;
   cout << x << endl;
   return 0;
}
```

Function Calling

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

int add(int* a)
{
   int b = 0;
   *a = *a + 1;
   b=*a;

   return b;
}


int main() {
   int x = 0;
   int result = add(&x);
   cout << result << endl;
   cout << x << endl;
   return 0;
}
```

Function Calling

Another way for “Pass by Reference”

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

int add(int &a)
{
  int b = 0;
  a = a + 1;
  b=a;

  return b;
}


int main() {
  int x = 0;
  int result = add(x);
  cout << result << endl;
  cout << x << endl;
  return 0;
}
```

Reference Variable:

Reference variable is an alias for a variable which is assigned to it.

Different from pointer:

- The reference variable can only be initialized at the time of its creation
- The reference variable returns the address of the variable preceded by the reference sign ‘&’
- The reference variable can never be reinitialized again in the program
- The reference variable can never refer to NULL