C++ POINTERS

Pointers

A pointer is an object, whose value refers to (or ***points to***) another value stored elsewhere in the computer memory using its memory address.

C++ pointer is a typed variable, whose stored *value* is the *address* of another variable.

Why Pointers?

They allow access to ***explicit memory locations***, which may be necessary in embedded systems.

They are needed to ***dynamically*** allocate memory for data structures of unknown size, accessed only by pointer, since they will ***not have a name***.

They are necessary to ***connect nodes*** in linked data structures – for example, linked lists.

Pointer Syntax

```
char *p;//declares a char pointer

int *q; //declares an int pointer

float *r;  //declares a float ptr

string *s;//declares a string ptr
```

Pointer Syntax

```
int * p, q;  //only p is a pointer
            //variable;
            //q is an int variable

int *p, *q; //to declare two
      //pointers, attach the *
      //to each variable's name
```

As with other types, C++ does ***not*** automatically ***initialize*** variables.

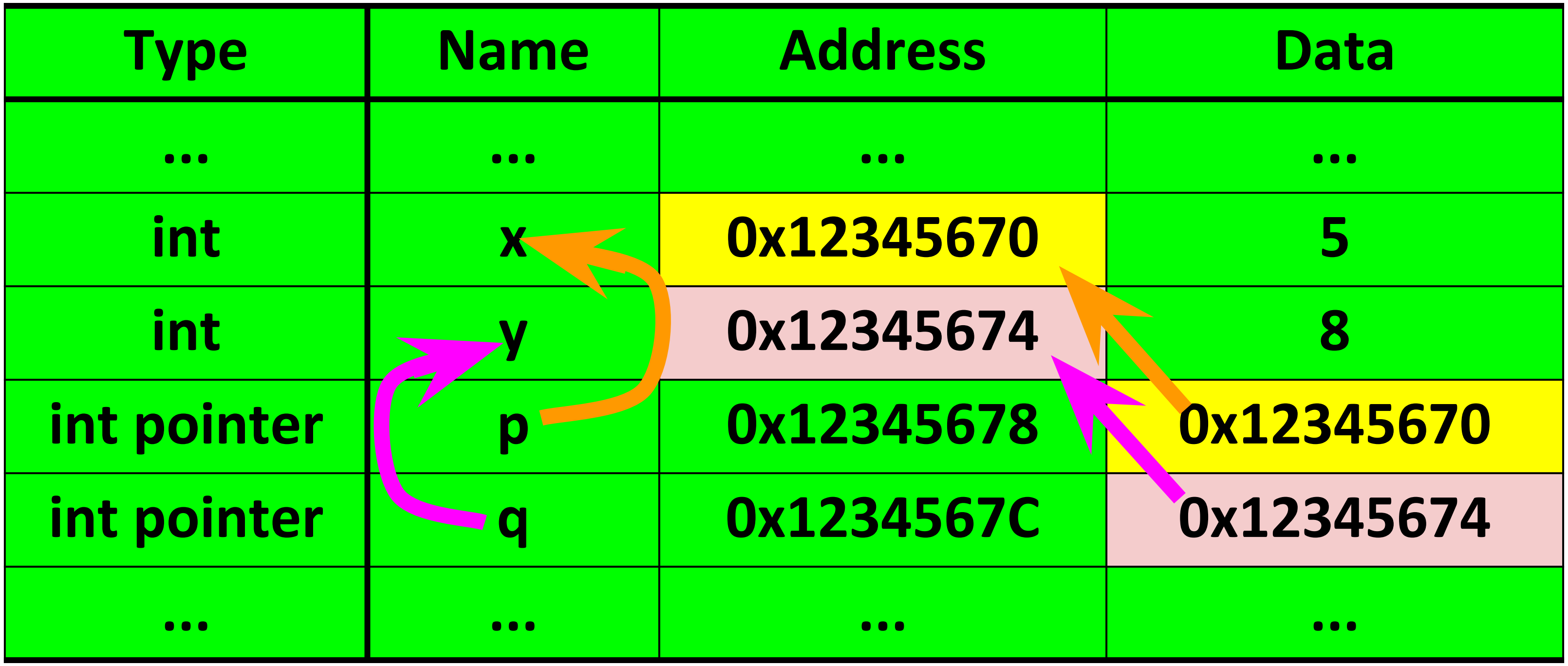
Pointer variables ***must be initialized*** to `nullptr`, unless a valid address is assigned to them at the moment of declaration.

```
int *p = nullptr; //p points to
                  //nothing (yet)
```

Address Of Operator & returns the address of its operand. This address can be assigned to a pointer variable:

```
int x = 5;
int y = 8;
int *p, *q;//declares two int ptrs
p = &x; //sets p to address of x
q = &y; //sets p to address of y
```

Memory State:



Type	Name	Address	Data
...	...	...	...
int	x	0x12345670	5
int	y	0x12345674	8
int pointer	p	0x12345678	0x12345670
int pointer	q	0x1234567C	0x12345674
...	...	...	...

Dereferencing Operator * returns the object, to which its operand points.

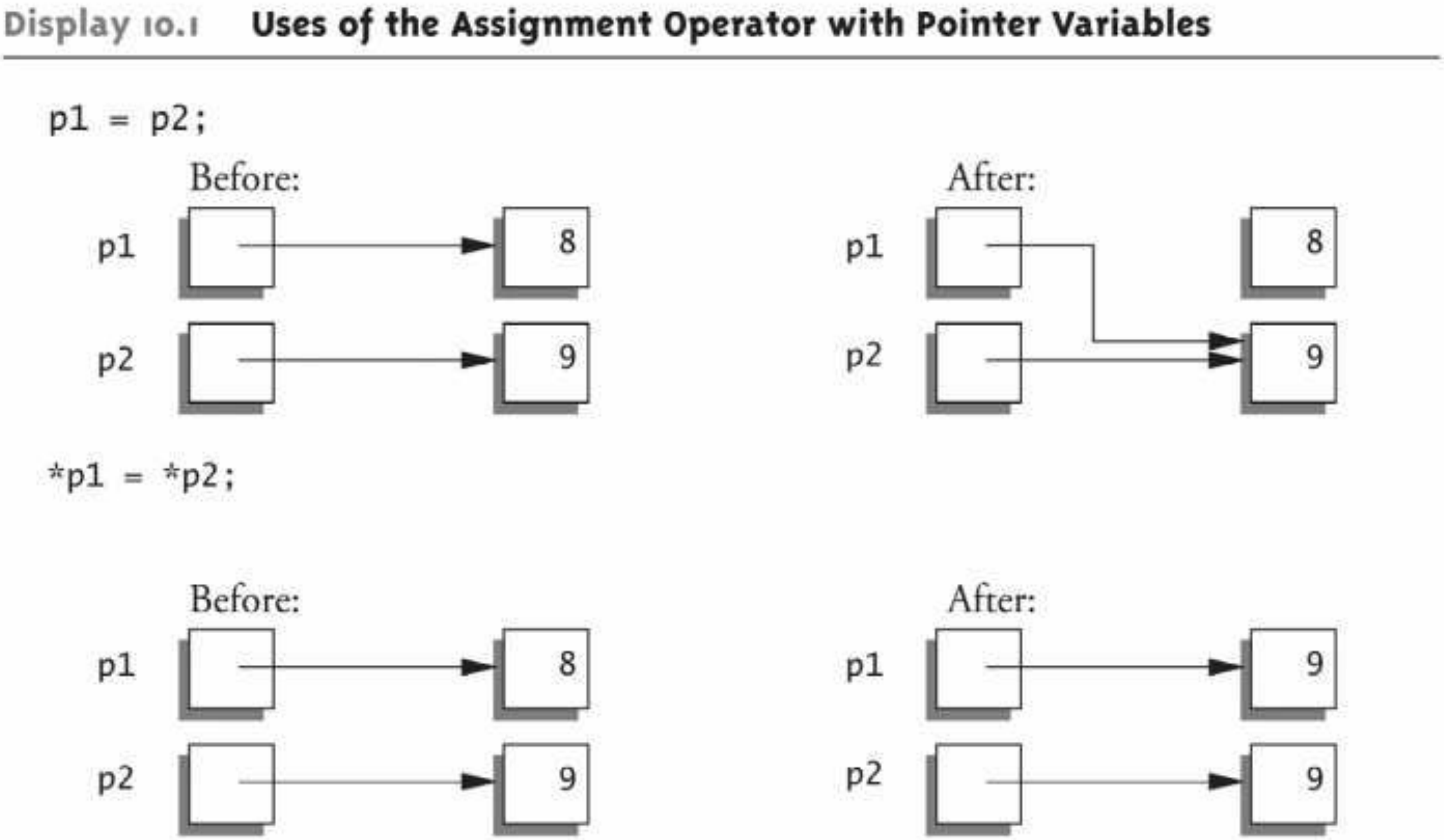
```
//p points to x, therefore:

cout << *p << endl; // x=5, y=8

y = *p; // x=5, y=5

*p = 4; // x=4, y=5
```

```
int a1 = 8, a2 = 9;
int *p1 = &a1, *p2 = &a2;
```



Display 10.1 Uses of the Assignment Operator with Pointer Variables

```
p1 = p2;
```

```
*p1 = *p2;
```

```
class Student
{ public:
    string name;
    int id;
    float gpa;
};

Student student1; //variable of
      //type Student
Student* student1Ptr; //pointer
    //to a variable of type Student
```

```
student1Ptr = &student1; //stores
        //address of student
        //in studentPtr

(*student1Ptr).gpa = 3.5;//stores
        //3.5 in member gpa of
        //student, using . operator

//or use -> member access operator:
student1Ptr->gpa = 3.5; // ->
        //is shorthand for ( *).
```

Functions and Pointers

A function can return a value of type pointer:

```
int* testExp(...)
{
  . . .
  //can return a pointer to
  //a dynamic array
}
```

Functions and Pointers

A pointer variable can be passed as a parameter either by value or by reference, in which case **&** is used:

```
void swapPtrs(int* &p, int *q) {
  //can make the caller function's
  //p point elsewhere, but not q
  int *tmp = p;
  p = q;
  q = tmp;
}
```

Functions and Pointers

```
int main() {
  int x = 3;
  int y = 4;
  int *p = &x;
  int *q = &y;
  swapPts(p, q);
  cout << *p << " " << *q << endl;
}

//prints 4 4
```

Pointers vs References

In a function call **&** indicates that an argument is **passed by reference**:

```
foo(int & x); int y = 5; foo(y);
```

The formal parameter is a reference (or alias) to the actual parameter, thus `foo` can change that variable by referring to its memory location using alias `x`.

Though address of a variable is passed it is used to create an alias on the fly. But only a **pointer variable can store this address for future use!**