COMP301
Data Structures & Algorithms [C++]

Dr. N. B. Gyan

Central University, Miotso. Ghana

Pointers

What Are Pointers?

A pointer is an object that can be used to access another object.

A pointer provides *indirect* access rather than *direct* access to an object. People use pointers in real-life situations all the time. Let us look at some examples.

1. When a professor says, “Do Problem 1.1 in the textbook,” the actual homework assignment is being stated indirectly.
2. A classic example of indirect access is looking up a topic in the index of a book. The index tells you where you can find a full description.

What Are Pointers?

3. A street address is a pointer. It tells you where someone resides. A forwarding address is a pointer to a pointer.
4. A *uniform resource locator* (URL), such as `http://www.cnn.com`, is a pointer. The URL tells you where a target Web page is. If the target Web page moves, the URL becomes stale, and points to a page that no longer exists.

5.? Can you think of another example?

What Are Pointers?

- In all these cases a piece of information is given out indirectly by providing a pointer to the information.
- In C/C++ a pointer is an object that stores an address (i.e., a location in memory) where other data are stored.
- An address is expected to be an integer, so a pointer object can usually be represented internally as an (`unsigned`) `int`.
- What makes a pointer object more than just a plain integer is that we can access the datum being pointed at.
- Doing so is known as *dereferencing* the pointer.

Pointer Syntax

Pointer Syntax in C++

- To have a pointer point at an object, we need to know the target object's memory address (that is, where it is stored).
- For any object `obj`, its memory address is given by applying the **unary** address-of operator `&`.
- Thus `&obj` is the memory location that stores `obj`.

Pointer Syntax

- We can declare that an object `ptr` points at an `int` object by saying

```
int *ptr;
```

- The value represented by `ptr` is an address.
- As with integer objects, this declaration does not initialize `ptr` to any particular value, so using `ptr` before assigning anything to it invariably produces bad results (e.g., a program crash).

Pointer Syntax in C++

- Suppose that we also have the declarations

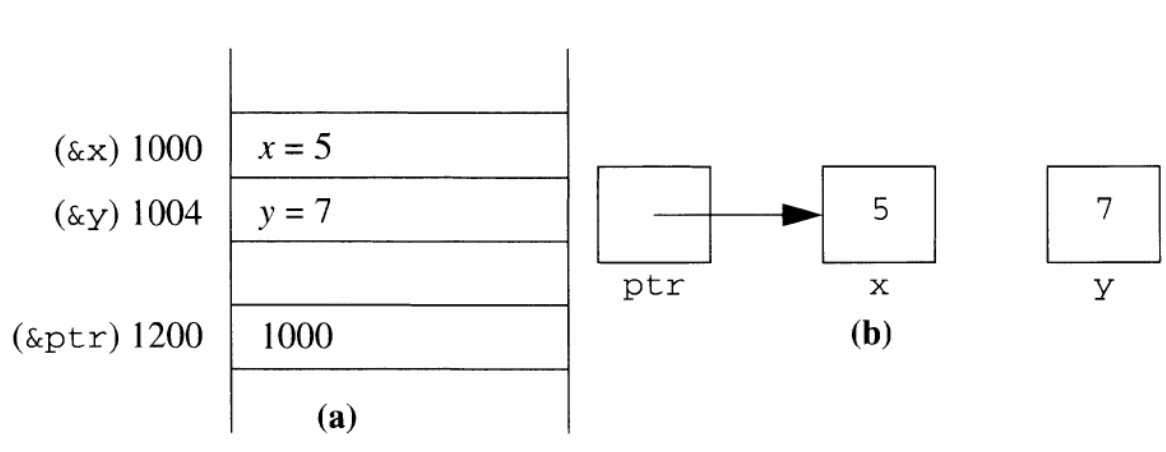
```
int x = 5;
int y = 7;
```

- We can make `ptr` point at `x` by assigning to `ptr` the memory location where `x` is stored. Thus

```
ptr = &x                    // LEGAL
```

sets `ptr` to point at `x`.

Pointer Syntax in C++



Pointers illustration: In part (**a**) a memory model shows where each object is stored. In part (**b**) an arrow is used to indicate pointing.

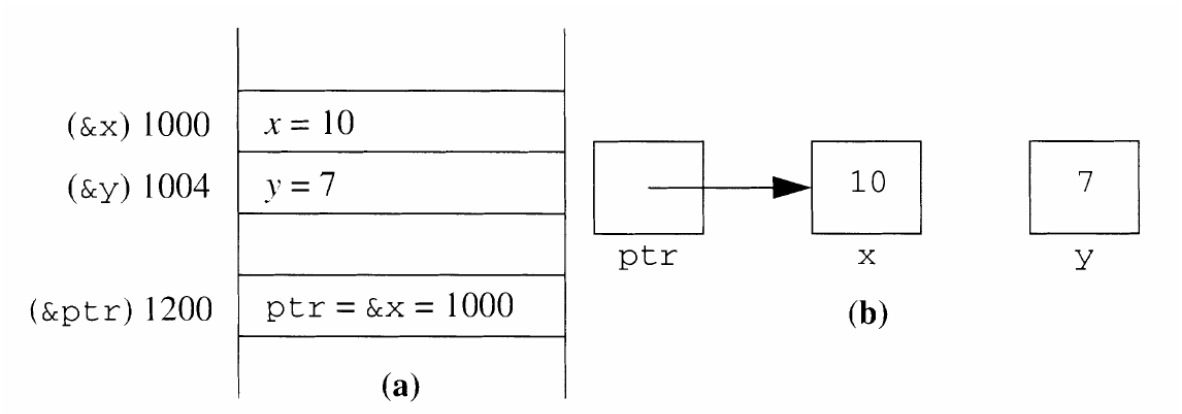
Pointer Syntax in C++

- The value of the data being pointed at is obtained by the unary **dereferencing operator** *.
- `*ptr` will evaluate to 5, which is the value of the pointed-at variable `x`.
- To dereference something that is not a pointer is illegal.
- Dereferencing works not only for reading values from an object, but also for writing new values to the object. Thus,

```
*ptr = 10          // LEGAL
```

changes the value of `x` to 10.

Pointer Syntax in C++



Result of `*ptr=10`: Unrestricted *alterations* are possible, and a *runaway* pointer can overwrite all sorts of variables unintentionally.

Pointer Syntax in C++

- Initializing a pointer at declaration time is also possible:

```
int x = 5;
int y = 7;
int *ptr = &x;          // LEGAL
```

- The declaration says that **x** is an **int** initialized to 5 , **y** is an **int** initialized to 7, and **ptr** is a pointer to an **int** and is initialized to point at **x**.

Pointer Syntax in C++

However, if the following is a start of a new program

```
int *ptr = &x;      // ILLEGAL: why?
int x = 5;
int y = 7;
```

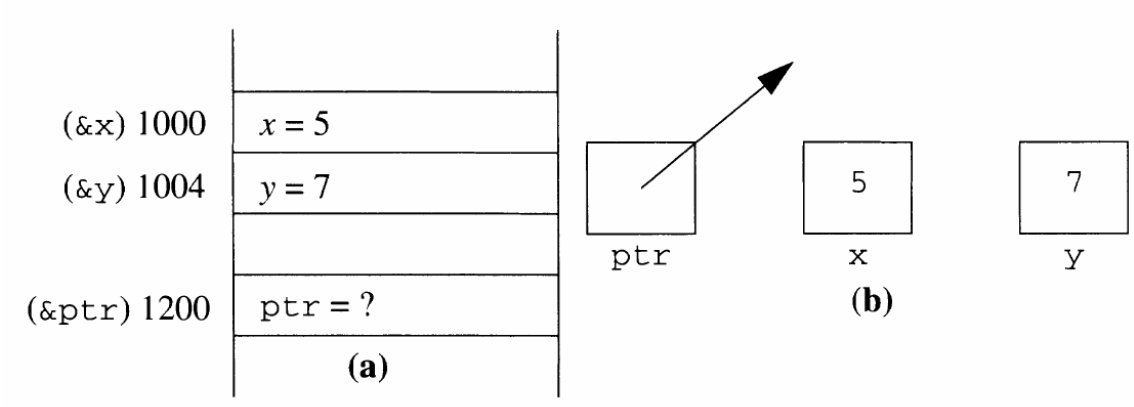
Pointer Syntax in C++

```
int x = 5;
int y = 7;
int *ptr = x;          // ILLEGAL: x not
                       // an address

int *ptr;              // LEGAL but ptr is
                       // uninitialised
```

What is the value of `ptr`?

Pointer Syntax in C++



Uninitialised pointer: the value is undefined because it was never initialized. Thus the value of `*ptr` is also undefined.

Pointer Syntax in C++

```
*ptr = x      // Semantically incorrect
              // but runs
```

- The compiler is quiet because the statement says that the `int` to which `ptr` is pointing should get the value of `x`. For instance, if `ptr` is `&y`, then `y` is assigned the value of `x`.
- This assignment is perfectly legal, but it does not make `ptr` point at `x`.
- Moreover, if `ptr` is uninitialized, dereferencing it is likely to cause a run-time error.

Pointer Syntax in C++: The NULL Pointer

- Sometimes it becomes necessary to state explicitly that a pointer is pointing nowhere, as opposed to an undefined location.
- The `NULL pointer` points at a memory location that is guaranteed to be incapable of holding anything. Consequently, a `NULL pointer` cannot be dereferenced.
- Pointers are best initialized to the `NULL pointer` because in many cases they have no default initial values (these rules apply to other predefined types as well).

Implications of the Precedence of `*`, `&`, and `[ ]`

- The dereferencing operator, `*`, and the address-of operator, `&`, are grouped in a class of **prefix operators**.
- These operators include the unary minus (`-`), the not operator (`!`), the bitwise complement operator, $\sim$, and the prefix increment and decrement operators (`++` and `--`), as well as `new`, `delete`, and `sizeof`.
- The prefix unary operators have *higher* precedence than *almost* all other operators.

Implications of the Precedence of `*`, `&`, and `[ ]`

- The exceptions are the scope operators and the **postfix operators**, such as the postfix increment and decrement operators (`++` and `--`), the function call operator `( )`, and the array access operator `[ ]`.
- Because of precedence rules, `*x++` is interpreted as `*(x++)`, not `(*x)++`.
- Note that a dereferenced pointer behaves just like the object that it is pointing at.

Pointer Syntax in C++

- Thus, after the following three statements, the value stored in `x` is 15:

```
x = 5;
ptr = &x;
*ptr += 10;
```

- As a result of **precedence rules** performing arithmetic not only on the dereferenced values, but also on the (undereferenced) pointers themselves is possible.

Pointer Syntax in C++

- For example, the following two statements are very different:

```
*ptr += 1;
*ptr++;
```

- In the first statement the `+=` operator is applied to `*ptr`, but in the second statement the `++` operator is applied to `ptr`.
- And the result of applying the `++` operator to `ptr` is that `ptr` will be changed to *point at a memory location one memory unit larger than it used to*.

Pointer Syntax in C++

Third, if `ptr1` and `ptr2` are pointers to the same type, then

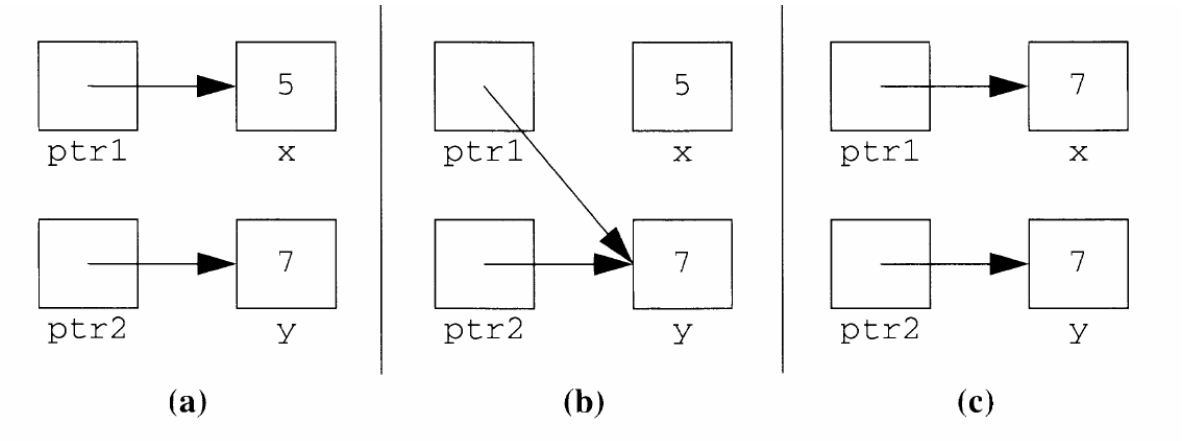
```
ptr1 = ptr2;
```

sets `ptr1` to point to the same location as `ptr2`, *whereas*

```
*ptr1 = *ptr2;
```

assigns the dereferenced `ptr1` the value of the dereferenced `ptr2` .

Pointer Syntax in C++



(a) Initial state; (b) `ptr1=ptr2` (c) `*ptr1=*ptr2` starting from initial state.

Pointer Syntax in C++

Similarly, the expression

```
ptr1 == ptr2
```

is true *if* the two pointers are pointing at the *same* memory location, *whereas*

```
*ptr1 == *ptr2
```

is true *if* the values stored at the two indicated addresses are *equal*.

Reference Variables

- In addition to the pointer type, C++ has the **reference type**.
- A reference type is an alias for another object and may be viewed as a pointer constant that is always dereferenced implicitly.
- For instance, in the following code, `cnt` becomes a synonym for a longer, hard-to-type variable:

```
int longVariableName = 0;
int & cnt = longVariableName;

cnt += 3;
```

Reference Variables

- Reference variables must be initialized when they are declared.
- They cannot be changed to reference another variable because an attempted reassignment via

```
cnt = someOtherObject;
```

assigns to the object `1ongVariableName` the value of `someOtherObject`.

Reference Variables

- Reference variables are like pointer constants in that the value they store is the address of the object they refer to.
- One important case is that a reference variable can be used as a *formal parameter*, which acts as an alias for an actual argument.

Reference Variables

Illustrating the use of pointers and references: `swapStyles.cpp`.

Verify that indeed `swapWrong(a, b)` does not work. Why is this so?

Pointers vs. References

The differences between reference and pointer types are summarized as follows.

- In the function declaration, reference parameters are used instead of pointers.
- In the function definition, reference parameters are implicitly dereferenced, so no `*` operators are needed (their placement would generate a syntax error).

- In the function call to `swapRef`, no `&` is needed because an address is implicitly passed by virtue of the fact that the corresponding formal parameters are references.
- The code involving the use of reference parameters is much more readable.

Assignment

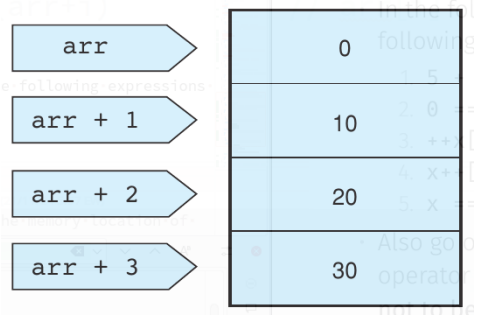
Questions

1. Give a short, one-line explanation for what the following expressions do.
   1.1 `*x + 5` (2 marks)
   1.2 `*x == 0` (2 marks)
   1.3 `*x * 3` (2 marks)
   1.4 `*x / *y` (2 marks)
   1.5 `*pv++` is equivalent to `*(pv++)`. Why? (2 marks)

A `void*` pointer (also called a *typeless pointer*) represents a memory address without establishing a certain type. When you use a typeless pointer for memory access, you must therefore name the type being accessed explicitly by means of *type casting*.

Questions

Consider and study carefully its application in `arrPtr.cpp` given on vcampus and compare with the following output:



arr[0], arr[1], arr[2], arr[3]

Use the above information for the questions below:

Questions

2. In the following, if `x` is a pointer what do the following expressions mean?
   2.1 `5 + x[0]` (2 marks)
   2.2 `0 == x[0]` (2 marks)
   2.3 `++x[0]` (2 marks)
   2.4 `x++[0]` (2 marks)
   2.5 `x == &x[0]` (2 marks)
3. Also go on the internet and familiarise yourself with operator precedence rules in C/C++. (Note: **This is not to be submitted**.)

DEADLINE: BEFORE class next week.
Submission to be done on vschool, **in pdf**.
Note this time that every handwritten document will be rejected.

See you next week, God willing 🙏🏾