Pointers

CSC100 / Introduction to programming in C/C++

README

- This script introduces C pointers in theory and practice.
- This section, including some sample code, is based on: chapter 11 in King (2008), and chapter 7 in Davenport/Vine.

Overview

- Computer memory is like a list of *locations*
- Each chunk of memory has an *address* to a location

- *Pointers* point to these addresses

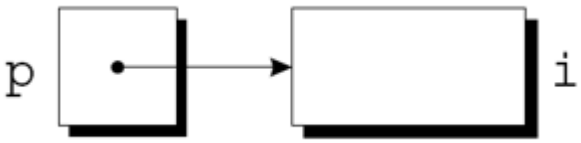

Figure 1: pointer p points to address of i

- The *address* is not the house, it's a *reference*

Figure 2: envelope = pointer to an address

- C#, Java, Pascal, Python…do not offer pointers (easily)[1]
- C and C++ offer pointer variables and operators naturally

- This gives you a lot more control over the computer (because every operation, every process involves memory management)
- Examples:
  - **String manipulation** (working with text - e.g. when creating fast-performing chat bots or AI agents)
  - **Dynamic memory allocation** - the process of assigning memory during the execution time (when a program typically competes with thousands of other processes)
- This is *mind control*: You can essentially decide what the computer should think with which part of its "brain" (great potential to mess up, too), e.g. when you mis-allocate resources.

Indirection (concept)

- Imagine you have a *variable* `iResult` that contains the *value* `75`.
- The variable is *located* at a memory address `0061FEC8`.
- Imagine you have a *pointer variable* `myPointer` that contains the address `0061FEC8` of the variable `iResult`.
- This means that `myPointer` *indirectly* points to the value `75`.
- You already worked with addresses: an *array name* `a` is a pointer to the start of the array, the address of `a[0]`.

- In 1, the conversion specifier `%p` lets us access the addresses that correspond to elements of the array `a`, and even the address for the whole array.

```
int a[2] = {100,1000};
printf("a[0] = %p\na[1] = %p\n&a   = %p\n", a[0], a[1], &a);
```

```
a[0] = 0x64
a[1] = 0x3e8
&a   = 0xbedcf178
```

- You already worked with pointers: arguments in the call of `scanf` are *pointers*: without the `&`, the function would be supplied with the *value* of `i`, not the *address*. But `scanf`'s job is to assign a memory location (an address) to the input variable.

```
int i;
scanf("%d", &i);
```

- The relationship between variable value and memory address is called *indirection*. A *pointer* provides *indirect* access to the value via the address where the value is stored.

Indirection (code)

- There are two *unary* pointer operators:
  - the *address* (or referencing) operator `&`
  - the *indirection* (or dereferencing) operator `*`
- The unary *address* operator `&` returns a computer memory address, e.g. `&iResult = 6422216` - it *references* the memory location
- The unary *indirection* operator `*` returns a value, e.g. `*myPointer = 75` if `myPointer` points at `&iResult`.

- In 1, a variable is declared and a value assigned to it. The indirect way of getting to the variable is via the pointer that points at its memory address.

```
int iResult; // declare an integer variable
iResult = 75; // iResult now has the value 75

int *myPointer; // declare an integer pointer variable
myPointer = &iResult; // myPointer points at iResult's address

printf("iResult = %d and 'lives' at &iResult = %p\n",
       iResult, &iResult);

printf("myPointer = %p points to iResult = %d\n",
       myPointer, *myPointer);
```

```
iResult = 75 and 'lives' at &iResult = 0xbeee1178
myPointer = 0xbeee1178 points to iResult = 75
```

- `[ ]`

Figure 3 illustrates these concepts. Can you describe what goes on from line to line?

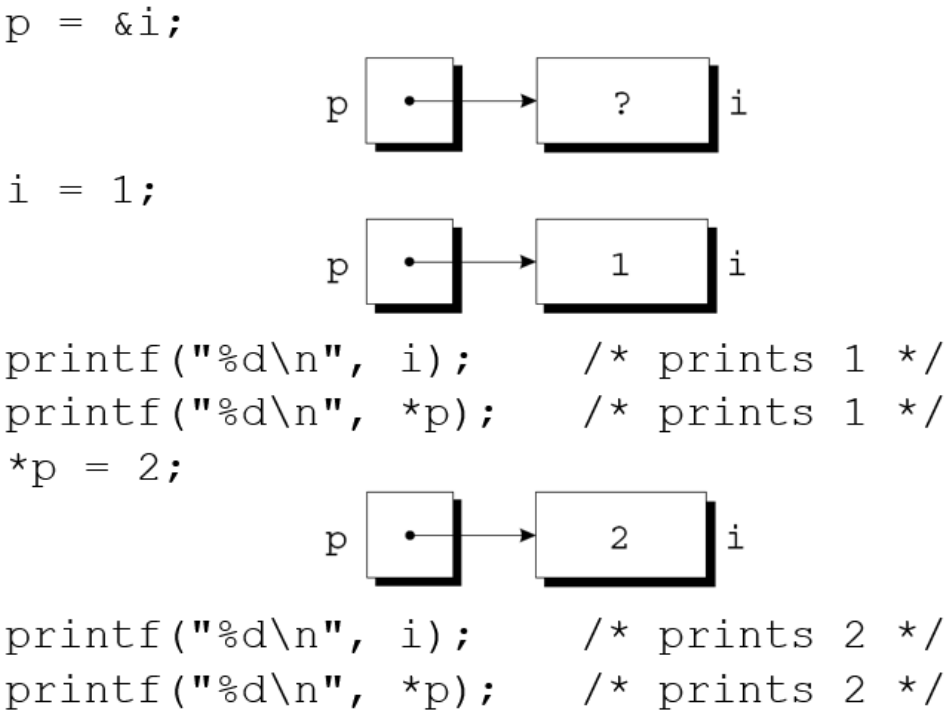

Figure 3: Graphical illustration of the indirection operator (Source: King)

1. The pointer `p` points to the address `&i` of the variable `i`.
2. `i` is initialized with the value `1`. `p` still points at it.
3. To change the value of `i` indirectly using the pointer `p`, we assign `*p = 2`. The indirection operator `*` designates a pointer.
4. To check that `i` indeed has been changed, we print it.
5. `*p` also prints the value of `i`.

* and & are inverse to one another

- Address and indirection operator are *inverse* to one another (i.e. they reverse each other's operation - applying both amounts to doing nothing).
- Applying indirection `*` to an address *dereferences* it.

- Applying referencing `&` to a pointer extracts its address.

```
int iResult = 75, *myPointer = &iResult; // declaring and initializing

// print variable and dereferenced pointer
printf("iResult = %d => *&iResult = %d\n",
       iResult, *&iResult);
// print pointer and address of pointer
printf("myPointer = %p => &*myPointer = %p\n",
       myPointer, &*myPointer);
```

```
iResult = 75 => *&iResult = 75
myPointer = 0xbe8a1178 => &*myPointer = 0xbe8a1178
```

- Applying `&` to a variable produces a pointer to the variable (cp. `scanf`)

- Applying `*` to the pointer takes us back to the original variable (dereferences the pointer)

```
j = *&i  // same as j = i
```

Pointers must be initialized

- Non-initialized pointers lead to invalid data or expressions.
- Pointer variables should always be initialized with:
  - another variable's memory address (e.g. `&i`), OR
  - with 0, OR
  - with the keyword `NULL`.

- Here are some *valid* pointer initializations - `printf` uses the conversion specifier `%p` for pointers.

```
double *ptr1;  // declarations
int *ptr2;
int *ptr3;
double x = 3.14; // initialize variable

ptr1 = &x; // initialize with address
ptr2 = 0;  // initialize with 0
ptr3 = NULL; // initialize with NULL

printf("%p %d %g\n", ptr1, ptr2, ptr3);
```

```
0xbeb5b168 0 3.14
```

- `[ ]`

Here are a few non-valid initializations:

- can you tell why?
- can you right the wrongs?

```
int i = 5; // declare and initialize i
int *iPtr; // declare pointer iPtr

iPtr = i;   // wrong because ...
iPtr = 7;   // wrong because ...
```

- Solution:

```
int i = 5; //
int *iPtr;

iPtr = &i;   // pointer initialized with memory address
*iPtr = 7;   // value of i indirectly changed

printf("%p %p %d\n", iPtr, &i, i);
```

```
0xbedd4178 0xbedd4178 7
```

Let's practice!

- Download the practice notebook pointers.org from GitHub:

```
wget tinyurl.com/bdd7rcpr -O pointers.org -o log
```

- Download also the diagram indirection.png from GitHub:

```
wget tinyurl.com/3m6pu7kf -O indirection.png -o log
```

References

- Davenport/Vine (2015) C Programming for the Absolute Beginner (3ed). Cengage Learning.
- Kernighan/Ritchie (1978). The C Programming Language (1st). Prentice Hall.
- King (2008). C Programming - A modern approach (2e). W A Norton. URL: knking.com.
- Orgmode.org (n.d.). 16 Working with Source Code [website]. URL: orgmode.org

Footnotes:

[1] Python e.g. is actually written in C - its default implementation is called CPython. However, in Python, usability was favored over machine performance, so pointers are not implemented at the user level. C underlies most of the much-used modern programming languages and their (internal) memory management.

Author: Marcus Birkenkrahe
Created: 2022-06-21 Tue 22:22