Pointers

Anna Simon

September 18, 2015

99 little bugs in the code.

99 little bugs in the code.

Take one down, patch it around.

127 little bugs in the code...

Adresses in memory

- Each variable used by the program is stored in a memory "cell".
- Each variable has only one address and a given address can be assigned to only one variable.
- To access the address of a variable use **address-of operator** & (ampersand)

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int myVar = 3;
6
7      cout << "The variable myVar with a value of: " << myVar << " is
           stored under the following address: " << &myVar << endl;
8
9      return 0;
10 }
```

Pointers

Pointer is a variable that stores the address of another variable.

Declaring a pointer

```
1 int* pPointer;
2 double *pMyTime;
3 char * pName;
```

* denotes a pointer, its position does not matter.
Pointer is always of the same type as the variable it is pointing to.

NOTE:

```
1 // only the variables directly preceeded by * are pointers:
2 int * pPointer, MyVar, *pPtr;
```

Accessing variable's address via pointer

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5
6      int myVar = 3;
7      int *pPointer;
8      pPointer = &myVar; //assign the address of MyVar to the pointer
9      cout <<
10     "myVar: " << myVar << endl <<
11     "&MyVar: " << &myVar << endl <<
12     "pPointer: " << pPointer << endl;
13
14     return 0;
15 }
```

Null pointer

An address should be assigned to the pointer when it is declared. If it is not possible, pointer should be declared as NULL

```
int *pPointer = NULL; //C notation, still used sometimes in C++
int *pPointer = 0;
```

to ensure it does not point to any memory cell.

Accessing the value of a variable using a pointer

To access the value a pointer is pointing to use *:

```
*pPointer;
```

NOTE: this is NOT a declaration of a pointer. This is a **dereference operator** that evaluates the content of the address the pointer is pointing to!

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

int main(){

    int myVar = 3;
    int *pPointer;
    pPointer = &myVar; //assign the address of MyVar to the pointer
    cout <<
    "myVar: " << myVar << endl <<
    "&MyVar: " << &myVar << endl <<
    "pPointer: " << pPointer << endl <<
    "*pPointer: " << *pPointer << endl;

    return 0;
}
```

Is it useful?

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   myVar = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

Is it useful?

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   myVar = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   *yourVar1 = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

Is it useful?

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   myVar = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   *yourVar1 = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

- Pointer allows for modification of the variables

Is it useful?

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   myVar = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

What will this code print to the screen?

```
1   int myVar = 3;
2   int *yourVar1;
3   int yourVar2;
4
5   yourVar1 = &myVar;
6   yourVar2 = myVar;
7
8   *yourVar1 = 6;
9
10  cout << myVar << " "
11       << *yourVar1 << " "
12       << yourVar2 << endl;
```

- Pointer allows for modification of the variables
- Assigning a value of a variable to another variable creates a copy

Size of the pointer

The size of a pointer depends on the architecture of the computer:

- a 32-bit machine uses 32-bit memory addresses - a pointer size will be 32 bits (4 bytes)
- a 64-bit machine uses 64-bit memory addresses - a pointer size will be 64 bits (8 bytes)

```
1 char *pChar=0;
2 double *pDouble=0;
3 int *pInt=0;
4
5 cout << "char " << sizeof(char) << " " << sizeof(pChar) << endl;
6 cout << "double " << sizeof(double) << " " << sizeof(pDouble) <<
      endl;
7 cout << "int " << sizeof(int) << " " << sizeof(pInt) << endl;
```

Summary of the (confusing) notation

notation	description
& nValue	

Summary of the (confusing) notation

notation	description
& nValue	address-of operator (get the memory address of `nValue`)
int *myValue	

Summary of the (confusing) notation

notation	description
& nValue	address-of operator (get the memory address of `nValue`)
int *myValue	declaration of a pointer
int *myValue = &nValue	

Summary of the (confusing) notation

notation	description
& nValue	address-of operator (get the memory address of `nValue`)
int *myValue	declaration of a pointer
int *myValue = &nValue	declares a pointer and assigns it the address of nValue
*myValue	

Summary of the (confusing) notation

notation	description
& nValue	address-of operator (get the memory address of `nValue`)
int *myValue	declaration of a pointer
int *myValue = &nValue	declares a pointer and assigns it the address of nValue
*myValue	dereference operator (accesses the value the pointer is pointing to)
*myValue = nValue2	

Summary of the (confusing) notation

notation	description
& nValue	address-of operator (get the memory address of `nValue`)
int *myValue	declaration of a pointer
int *myValue = &nValue	declares a pointer and assigns it the address of nValue
*myValue	dereference operator (accesses the value the pointer is pointing to)
*myValue = nValue2	assigns a value of nValue2 to the variable the pointer is pointing to
myValue	

Summary of the (confusing) notation

notation	description
& nValue	address-of operator (get the memory address of `nValue`)
int *myValue	declaration of a pointer
int *myValue = &nValue	declares a pointer and assigns it the address of nValue
*myValue	dereference operator (accesses the value the pointer is pointing to)
*myValue = nValue2	assigns a value of nValue2 to the variable the pointer is pointing to
myValue	returns the address the pointer is pointing to

Pointer arithmetic

C++ allows pointer arythmetic: integer addition and subtraction.
If `pPointer` points to an integer `pPointer+1` points to the next address 4 bytes (i.e. the size of integer) after the one `pPointer` points to.

Check what the following code outputs:

```
1 int nValue = 7;
2 int *pPointer = &nValue;
3
4 cout << pPointer << endl;
5 cout << pPointer+1 << endl;
6 cout << pPointer+2 << endl;
7 cout << pPointer+3 << endl;
```

Note: the addresses are given in hexadecimal notation.

Pointers and arrays

Array is actually a pointer that points to the first element of the array.

Using the dereference operator returns the zeroth element of the array:

```
1 int anArray[5] = {5, 4, 3, 2, 1};
2
3 cout << *anArray << endl;
```

Using pointer arithmetic, we can access all the elements of the array:

```
1 cout << *(naArray+1) << endl;
2
3 for (int i=0; i<5;i++)
4 {
5     cout << *(anArray+i) << endl;
6 }
```

Pointers and arrays

A pointer can be used to "scan" an array:

```
1  const int nArraySize = 7;
2  char szName[nArraySize] = "Mollie";
3  int nVowels = 0;
4  //run through addresses from the first element of the array to the
       last one
5  for (char *pnPtr = szName; pnPtr < szName + nArraySize; pnPtr++)
6  {
7      switch (*pnPtr)
8      {
9          case 'A':
10         case 'a':
11         case 'E':
12         case 'e':
13         case 'I':
14         case 'i':
15         case 'O':
16         case 'o':
17         case 'U':
18         case 'u':
19             nVowels++;
20             break;
21     }
22 }
23 cout << szName << " has " << nVowels << " vowels" << endl;
```

Struct and pointers

C-style:

```
1  struct data_t{
2      int nValue;
3      char cChar;
4  };
5  data_t data;
6  data.nValue = 55;
7  data.cChar = 'a';
8  data_t *pPointer = &data;
9
10 //brackets are important!
11 //access value stored in struct:
12 cout << "(*pPointer).nValue=" <<
13         (*pPointer).nValue << endl;
14
15 //change the value stored
16 (*pPointer).nValue = 99;
17
18 //print out the calue of the struct
       's element
19 cout << "data.nValue=" << data.
       nValue << endl;
```

C++-style:

```
1  struct data_t{
2      int nValue;
3      char cChar;
4  };
5  data_t data;
6  data.nValue = 55;
7  data.cChar = 'a';
8  data_t *pPointer = &data;
9
10
11 //access value stored in struct:
12 cout << "pPointer->nValue=" <<
       pPointer->nValue << endl;
13
14 //change the value stored
15 pPointer->nValue = 99;
16
17 //print out the value of the struct
       's element
18 cout << "data.nValue=" << data.
       nValue << endl;
```

Save the following grade distribution into a two-column text file:

grade	5	4.5	4	3.5	3	2
#Students	3	5	7	4	1	1

Write a code that reads in your data distribution into two arrays: grade and nStudents. Using a single for loop and pointer arithmetic, calculate the average grade.

Rewrite your code to use a struct that contain the same information, e.g.:

```
1 struct distr_t{
2     float grade[6];
3     int nStudents[6];
4 }
5
6 distr_t distr;
```

and accessing it using C++-style pointers for structs.