UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

E.P. De Ciencia de La Computación

C++ Self-Review Exercises Cap. 8

Bedregal Vento, Adrian Rolando

Docente:
Alvaro Henry Mamani Aliaga

12 de octubre de 2018

SELF-REVIEW EXERCISES

1 Answer each of the following:
a) A pointer is a variable that contains as its value the **memory address** of another variable.
b) A pointer should be initialized to `nullptr` or **an address**.
c) The only integer that can be assigned directly to a pointer is **0**.

2 State whether each of the following is *true* or *false*. If the answer is *false*, explain why.
a) The address operator `&` can be applied only to constants and to expressions. **(False)**. Conversely, this operator must be an *lvalue* and cannot be applied to constants or expresions that have temporary values (*rvalue*).
b) A pointer that is declared to be of type void * can be dereferenced.
c) A pointer of one type can't be assigned to one of another type without a cast operation.

3 For each of the following, write C++ statements that perform the specified task. Assume that double-precision, floating-point numbers are stored in eight bytes and that the starting address of the built-in array is at location 1002500 in memory. Each part of the exercise should use the results of previous parts where appropriate.

a) Declare a built-in array of type double called numbers with 10 elements, and initialize the elements to the values 0.0 , 1.1 , 2.2 , ..., 9.9. Assume that the constant size has been defined as 10.

```
double numbers[10] = {0.0, 1.1, 2.2, 3.3, 4.4,
                      5.5, 6.6, 7.7, 8.8, 9.9};
```

b) Declare a pointer nPtr that points to a variable of type double.

```
double* nPtr;
```

c) Use a for statement to display the elements of built-in array numbers using array subscript notation. Display each number with one digit to the right of the decimal point.

```
for(size_t i = 0; i < 10; ++i)
{
    if(i == 0)
    {
        cout << "0.0" << ' ';
        continue;
    }
    cout << numbers[i] << ' ';
}
```

d) Write two separate statements that each assign the starting address of built-in array numbers to the pointer variable nPtr.

```
nPtr = numbers;
nPtr = &numbers[0];
```

e) Use a for statement to display the elements of built-in array numbers using pointer/offset notation with pointer nPtr.

```
for(size_t i = 0; i < 10; ++i)
    cout << *(nPtr + i) << ' ';
```

f) Use a for statement to display the elements of built-in array numbers using pointer/offset notation with the built-in array's name as the pointer.

```
for(size_t i = 0; i < 10; ++i)
    cout << *(numbers + i) << ' ';
```

g) Use a for statement to display the elements of built-in array numbers using pointer/subscript notation with pointer nPtr.

```
for(size_t i = 0; i < 10; ++i)
    cout << nPtr[i] << ' ';
```

h) Refer to the fourth element of built-in array numbers using array subscript notation, pointer/offset notation with the built-in array's name as the pointer, pointer subscript notation with nPtr and pointer/offset notation with nPtr.

```
numbers[3];
*(numbers + 3);
*(nPtr + 3);
nPtr[3];
```

i) Assuming that nPtr points to the beginning of built-in array numbers , what address is referenced by nPtr + 8 ? What value is stored at that location?

```
// Garbage Data
```

j) Assuming that nPtr points to numbers[5] , what address is referenced by nPtr after nPtr -= 4 is executed? What's the value stored at that location?

```
// 1.1
```

4 For each of the following, write a single statement that performs the specified task. Assume that floating-point variables number1 and number2 have been declared and that number1 has been initialized to 7.3.

a) Declare the variable fPtr to be a pointer to an object of type double and initialize the pointer to nullptr.

```
double* fPtr = nullptr;
```

b) Assign the address of variable number1 to pointer variable fPtr.

```
fPtr = &number1;
```

c) Display the value of the object pointed to by fPtr.

```
cout << *fPtr << endl;
```

d) Assign the value of the object pointed to by fPtr to variable number2.

```
number2 = *fptr;
```

e) Display the value of number2.

```
cout << number2 << endl;
```

f) Display the address of number1.

```
cout << &number1 << endl;
```

g) Display the address stored in fPtr . Is the address displayed the same as that of number1 ?

```
cout << fPtr << endl; // Yes, it is
```

5 Perform the task specified by each of the following statements:

a) Write the function header for a function called exchange that takes two pointers to double-precision, floating-point numbers x and y as parameters and does not return a value.

```
// void exchange(double* x, float* y)
```

b) Write the function prototype for the function in part (a).

```
void exchange(double*, float*);
```

c) Write two statements that each initialize the built-in array of char s named vowel with the string of vowels, “AEIOU”.

```
char vowel[] = "AEIOU";
```

6 Find the error in each of the following program segments. Assume the following declarations and statements:

```
int *zPtr; // zPtr will reference built-in array z
void *sPtr = nullptr;
int number;
int z[ 5 ] = { 1, 2, 3, 4, 5 };
```

```
++zPtr;
// zPtr was not initialized

// use pointer to get first value of a built-in array
number = zPtr;
// The pointer is still a pointer

// assign built-in array element 2 (the value 3) to number
number = *zPtr[ 2 ];
// zPtr is the name of the array, we cannot dereference
// and then use the [] operator

// display entire built-in array z
for ( size_t i = 0; i <= 5; ++i )
    cout << zPtr[ i ] << endl;
// Goes away bounds

// assign the value pointed to by sPtr to number
    number = *sPtr;
// Tries to dereference a void pointer

++z;
// We cannot modify an array name
```

7 **(True or False)** State whether the following are true or false. If false, explain why.

a) Two pointers that point to different built-in arrays cannot be compared meaningfully. **True**
b) Because the name of a built-in array is implicitly convertible to a pointer to the first element of the built-in array, built-in array names can be manipulated in the same manner as pointers. **False**. Despite a built-in array name can be implicitly converted to a pointer, it doesn't have the same behaviour, if we try to do that, we'll get a compiling error.

8 **(Write C++ Statements)** For each of the following, write C++ statements that perform the specified task. Assume that unsigned integers are stored in two bytes and that the starting address of the built-in array is at location 1002500 in memory.

a) Declare a built-in array of type unsigned int called values with five elements, and initialize the elements to the even integers from 2 to 10. Assume that the constant SIZE has been defined as 5.

```
1 const unsigned int SIZE = 5;
2 unsigned int values[SIZE];
3 for(unsigned int i = 1; i <= SIZE; ++i)
4     values[i] = i * 2;
```

b) Declare a pointer vPtr that points to an object of type unsigned int.

```
1 unsigned int* vPtr;
```

c) Use a for statement to display the elements of built-in array values using array subscript notation.

```
1 for(unsigned int i = 0; i < SIZE; ++i)
2     cout << values[i] << endl;
```

d) Write two separate statements that assign the starting address of built-in array values to pointer variable vPtr.

```
1 vPtr = &values[0];
2 vPtr = values;
```

e) Use a for statement to display the elements of built-in array values using pointer/offset notation.

```
1 for(unsigned int i = 0; i < SIZE; ++i)
2     cout << *(vPtr + i) << endl;
```

f) Use a for statement to display the elements of built-in array values using pointer/offset notation with the built-in array's name as the pointer.

```
1 for(unsigned int i = 0; i < SIZE; ++i)
2     cout << *(values + i) << endl;
```

g) Use a for statement to display the elements of built-in array values by subscripting the pointer to the built-in array.

```
for(unsigned int i = 0; i < SIZE; ++i)
    cout << vPtr[i] << endl;
```

h) Refer to the fifth element of values using array subscript notation, pointer/offset notation with the built-in array name's as the pointer, pointer subscript notation and pointer/offset notation.

```
values[4];
*(values + 4);
vPtr[4];
*(vPtr + 4);
```

i) What address is referenced by vPtr + 3 ? What value is stored at that location?

```
// The 5th value in the array
```

j) Assuming that vPtr points to values[4] , what address is referenced by vPtr -= 4 ? What value is stored at that location?

```
// The 1th value in the array
```

9 **(Write C++ Statements)** For each of the following, write a single statement that performs the specified task. Assume that long variables value1 and value2 have been declared and value1 has been initialized to 200000.

a) Declare the variable longPtr to be a pointer to an object of type long.

```
long* longPtr;
```

b) Assign the address of variable value1 to pointer variable longPtr.

```
longPtr = &value1;
```

c) Display the value of the object pointed to by longPtr.

```
cout << *longPtr;
```

d) Assign the value of the object pointed to by longPtr to variable value2.

```
value2 = *longPtr;
```

e) Display the value of value2.

```
cout << value2 << endl;
```

f) Display the address of value1.

```
cout << value1 << endl;
```

g) Display the address stored in longPtr . Is the address displayed the same as value1 's?

```
cout << *longPtr << endl; // Yes, it is
```

10 **(Function Headers and Prototypes)** Perform the task in each of the following statements:

a) Write the function header for function zero that takes a long integer built-in array parameter bigIntegers and does not return a value.

```
void zero(long bigIntegers[])
```

b) Write the function prototype for the function in part (a).

```
void zero(long*);
```

c) Write the function header for function add1AndSum that takes an integer built-in array parameter oneTooSmall and returns an integer.

```
int add1AndSum(int oneTooSmall[])
```

d) Write the function prototype for the function described in part (c).

```
int add1AndSum(int*);
```

11 **(Find the Code Errors)** Find the error in each of the following segments. If the error can be corrected, explain how.

```
// a)
int* number;
cout << number << endl;
// Solution:
/* We can assign another variable to be pointed
by number and in the cout segment put an * before
numer */
```

```
// b)
double* realPtr;
long* integerPtr;
integerPtr = realPtr;
cout << number << endl;
// Solution:
/* Change the type of double* to long*
or viceversa */
```

```
// c)
int* x, y;
x = y;
// Solution:
/* Put a * before y */
```

```
// d)
char s[] = "this is a character array";
for( ; *s != '\0'; ++s)
    cout << *s << ' ';
// Solution:
/* Create another pointer instead s
because s is an array name and cannot
be modified like that */
```

```
// e)
short* numPtr, result;
void* genericPtr = numPtr;
result = *genericPtr + 7;
// Solution:
/* Dereference numPtr instead genericPtr
due to we cannot dereference a void pointer */
```

```
// f)
double x = 19.34;
double xPtr = &x;
cout << xPtr << endl;
// Solution:
/* xPtr is not actually a pointer, we must
put a * before its name and in the cout
part we should also put a * before xPtr
in order to dereference it */
```

(Simulation: The Tortoise and the Hare)

```
// The Tortoise and the Hare
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void moveTortoise(int* tortoisePtr, int mt)
{
    if(mt <= 5)
        *tortoisePtr += 3;
    else if(mt <= 7)
        *tortoisePtr -= 6;
    else
        *tortoisePtr += 1;

    if(*tortoisePtr < 1)
        *tortoisePtr = 1;
}

```

```
20 void moveHare(int* harePtr, int mt)
21 {
22     if(mt <= 2)
23         *harePtr += 0;
24     else if(mt <= 4)
25         *harePtr += 9;
26     else if(mt <= 5)
27         *harePtr -= 12;
28     else if(mt <= 8)
29         *harePtr += 1;
30     else
31         *harePtr -= 2;
32
33     if(*harePtr < 1)
34         *harePtr = 1;
35 }
36
37 void printPositions(int* tortoisePtr, int* harePtr)
38 {
39     if(*tortoisePtr == *harePtr)
40         cout << "OUCH!!!     " << *tortoisePtr
41              << "\t\t" << *harePtr << endl;
42     else
43         cout << "            " << *tortoisePtr
44              << "\t\t" << *harePtr << endl;
45 }
46
47 int main()
48 {
49     int tortoise = 1, hare = 1, rnd;
50     srand(time(0));
51     cout << "\n\t\tBANG!!!\n" << "              "
52          << "AND THEY'RE OFF!!!\n" << endl;
53     cout << "\tTortoise" << "\tHare" << endl;
54     while(tortoise < 70 && hare < 70)
55     {
56         rnd = 1 + rand() % 10;
57         moveTortoise(&tortoise, rnd);
58         moveHare(&hare, rnd);
59         printPositions(&tortoise, &hare);
60     } cout << endl;
61     if(tortoise >= hare)
62         cout << "\t TORTOISE WINS!!! YAY!!! " << endl;
63     else
64         cout << "\t Hare wins" << endl;
65
66     return 0;
67 }
```

13 **(What Does This Code Do?)** What does this program do?

```
// Ex. 8.13: ex08_13.cpp
// What does this program do?
#include <iostream>
using namespace std;

void mystery1( char *, const char * ); // prototype

int main()
{
    char string1[ 80 ];
    char string2[ 80 ];

    cout << "Enter two strings: ";
    cin >> string1 >> string2;
    mystery1( string1, string2 );
    cout << string1 << endl;
} // end main

// What does this function do?
void mystery1( char *s1, const char *s2 )
{
    while ( *s1 != '\0' )
        ++s1;

    for ( ; ( *s1 = *s2 ); ++s1, ++s2 )
        ; // empty statement
} // end function mystery1

/* ANSWER: This program concatenates two strings
and save them into the first one */
```

14 **(What Does This Code Do?)** What does this program do?

```
// Ex. 8.14: ex08_14.cpp
// What does this program do?
#include <iostream>
using namespace std;

int mystery2( const char * ); // prototype

int main()
{
    char string1[ 80 ];

    cout << "Enter a string: ";
    cin >> string1;
    cout << mystery2( string1 ) << endl;
} // end main

// What does this function do?
int mystery2( const char *s )
{
    unsigned int x;

    for ( x = 0; *s != '\0'; ++s )
        ++x;

    return x;
} // end function mystery2

// ANSWER: Return and print the length of a string
```