Reference: Big C++.

Exercises 7:

Exercise R7.1.

1. int\* p = new int;

// Nothing Wrong!

1. p = 5;

// Assign a pointer to a value, not the int that the pointer p points to!

1. \*p = \*p + 5;

// Wrong, p points now to 5 which will be inaccessible to access what’s in there

1. Employee e1 = new Employee ("Hacker, Harry", 34000);

// Wrong, e1 is not a pointer and new returns a pointer not a variable

1. Employee e2;

// Define a variable e2 on the stack. Nothing wrong

1. e2->set\_salary (38000);

// e2 is a variable not a pointer and to apply the member function set\_salary on it, you should use the dot operator.

1. delete e2;

// e2 is not a pointer, it’s a variable that lives on the stack and cannot be reclaimed by the programmer. It’s automatically reclaimed by the end of the function.

1. Time\* pnow = new Time ();

// Nothing Wrong!

1. Time\* t1 = new Time (2, 0, 0);

// Nothing Wrong!

1. cout << t1->seconds\_from (pnow);

//Wrong, it should be like this: std::cout << t1->seconds\_from(\*pnow);

1. delete \*t1;

// Wrong, you delete the pointer not what it points to!

1. cout << t1->get\_seconds ();

// t1 is a deleted pointer, you can’t access what it now points to!

1. Employee\* e3 = new Employee ("Lin, Lisa", 68000);

// Nothing Wrong!

1. cout << e3.get\_salary ();

// e3 is a pointer not an Employee object. Instead use (\*e3).get\_salary(); or e3->get\_salary();

1. Time\* t2 = new Time (1, 25, 0);

// Nothing is wrong

1. cout << \*t2.get\_minutes ();

// Use (\*t2).get\_minutes(); or t2->get\_minutes ();

1. delete t2;

// Nothing is wrong

Exercise R7.2.

int\* a = new int;

\*a = 5;

std::cout << \*a << "\n";

//No problem, prints 5

int\* b = a;

delete a;

std::cout << \*b << "\n";

// Random number, No crash

int\* c = NULL;

\*c = 5;

std::cout << \*c << "\n";

// Program Crashes

int\* d = a; // a now contains random value after it has been deleted

\*d = 5;

std::cout << \*d << "\n";

// Program Crashes

Exercise R7.3.

Dereference the pointer a will result in \*a = 5

Dereference the pointer b after a was deleted will result in a random number

Exercise R7.4.

* Memory leak will happen
* Deleting a pointer twice will lead the program to crash.

Exercise R7.5.

* harry.get\_salary(); Should return 35000
* boss.get\_salary(); Should return 45000
* pharry->get\_salary(); Should return 45000
* pboss->get\_salary(); Should return 45000

Exercise R7.6.

int\* p = new int;

double\* q = new double;

std::cout << "p: " << (unsigned long)p <<"\n";

p = p + 1;

std::cout << "p + 1: " << (unsigned long)p << "\n";

std::cout << "q: " << (unsigned long)q << "\n";

q = q + 1;

std::cout << "q + 1: " << (unsigned long)q << "\n";

Output:

p: 54294056

p + 1: 54294060

p + 1 differs from p by 4 bytes which is the size of an int in 86x system.

q: 54325392

q + 1: 54325400

q + 1 differs from q by 8 bytes which is the size of a double variable.

Exercise R7.7.

Casting a double\* pointer to an int\* pointer makes no sense despite they both have the same size, but each one points to different things (which have different sizes).

So for example:

double values[] = { 2, 3, 5, 7, 11, 13 };

int\* p = static\_cast<int\*>(values);

If this code worked, and int\* p now points to the first element of values array. Then to get the second element of the array, you will need to add 4 (size of int) to reach it. But you must add 8 to get the second element. Because it was defined as an array of doubles.

Exercise R7.8.

void f(int p[])

{

int\* q;

const int\* r;

int s[10];

p = q; legal, pointer to pointer

p = r; legal

p = s;

q = p;

q = r;

q = s;

r = p;

r = q;

r = s;

s = p;

s = q;

s = r;

}