**SUBMISSON FOR H02 – Lab CS162**

**3.1. Assignment 1**

The program can run successfully and has no errors.

The result printed out on the screen will be in this format:

*<The address of a> //*

*3* // the value of a

*<The address of b>*

*3* // the value of a

*<The address of a>*

**3.2. Assignment 2**

The program runs successfully and has no errors.

The result on the screen will be:

100 // the value of x

1.2 // the value of y

400 // the value of z

*<The address of z>*

400 // the value of z

*<The address of ip1>*

*<The address of x>*

100 // the value of x

*<The address of ip2>*

*<The address of y>*

1.2 // the value of y

*<The address of fp>*

*<The address of ch>*

Z // the value of ch

*<The address of chp>*

**3.3. Assignment 3**

The result on the screen will be 2 2.

However, there’s still some problems:

1. Because b now points to a, so when we delete a, b will also be deleted, and the line “delete b” is unnecessary and that is the reason for run time error in this program.
2. Initially, b is a pointer and point to a random integer; however, when b is deleted, there still exists something (we call it garbage) in your memory.

**3.4. Assignment 4**

The program runs successfully and has no errors.

The result on the screen will be:

3 // the value of a

5 // the new value of p

**3.5. Assignment 5**

The last values of \*p, q, \*r, v and \*s are 50, 8, 8, 8, 50 respectively.

**3.6. Assignment 6**

The last values of \*p, \*q, v and nom[] are 12, 11, 11, [12, 13, 12, 10, 16] respectively.

**3.7-50. Assignement 7-50:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| A | D | D | B | B | D | A | A | D | C | B |
| **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** |
| D | A | A | D | D | B | C | B | A | A | C |
| **29** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** |
| C | C | D | B | C | C | - | - | - | - | - |
| **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** | **48** | **49** | **50** |
| A | A | C | D | A | A | B | D | C | C | A |

**Short explainations for some answers:**

**3.8. Assignment 8:**

\*x = x[n] means that x now points to ‘\0’, that’s why we have that result.

**3.9. Assignment 9:**

“s++’ means now s points to str[1] (“eace”), and “s++ +2” points to the 2nd index of “eace”, that is “ce”.

**3.11. Assignment 11:**

Since the size of integer is 4 bytes, so that the size of arr is 20 (5 integers).

\*arr or arr[0] only points to an integer, so that their size are both 4.

**3.12. Assignment 12:**

This will print out “300” normally, since the str isn’t meaning.

**3.16. Assignment 16:**

Because i is an integer, not a double.

**3.20. Assignment 20:**

When ptr += 5, this means ptr now points to str[5] (“fg”).

**3.30. Assignment 30:**

Because *cho* is the pointer, which points to *ch* (the varible with initial value is ‘A’(means 65 in ASCII)), after line *cho += a* then the value of pointer *ch* is 65 + 32 = 97.

Same, *ptr* now points to *a*, the line *\*ptr += ch* means the current value of *ptr* (it’s also the current value of *a*) is 32 + 97 = 129.

About *ch*, *ch* now has the value in integer is 97, means that the character ‘a’.

**3.31. Assignment 31:**

Variable *i* is a constant, so that it cannot be changed the value.

**3.32. Assignment 32:**

p = num; // p points to num[0]

\*p = 10; // the value of num[0] now is 10.

p++; // then p points to num[1]

\*p = 20; // num[1] = 20.

p = &num[2]; // p points to num[2]

\*p = 30; // num[2] = 30.

p = num + 3; // p points to num[3]

\*p = 40; // num[3] = 40.

p = num; // p points to num[0] again.

\*(p + 4) = 50; // num[0 + 4] = num[4] = 50.

**3.33. Assignment 33:**

int a[] = { 4, 5, 6, 7 }; // declare the array

int\* p = (a + 1); // p points to a[1]

cout << \*a + 10; // (\*a) is the value of the first element in the array (that’s 4), so that the result on the screen is 4 + 10 = 14.

**3.34. Assignment 34:**

There’s an issue with the declaration of the reference *ra*. When declaring a reference, you need to initialize it with an existing variable, for example: *&ra = a*.

**3.35. Assignment 35:**

int a[] = {1, 2, 3, 4, 5};

int\* ptr;

ptr = a; // ptr points to the first element in the array.

cout << \*(ptr + 1); // print out the second element in the array => 2.

**3.36. Assignment 36:**

int a = 5;

int\* ptr;

ptr = &a; // ptr points to a

\*ptr = \*ptr \* 3; // it means that a = a \* 3 => the current value of a is 15

cout << a; => The result printed out on the screen is 15.

**3.37. Assignment 37:**

int i = 6, \* j, k;

j = &i; // j points to i

cout << i \* \*j \* i + \*j; // it means 6 \* 6 \* 6 + 6 = 222.

**3.38. Assignment 38:**

int x = 20, \* y, \* z;

// Adress of x: 500

// integer is 4 byte size

y = &x; // *y points to x; y = address of x = 500*

z = y;

\*y++; // *This line effectively does nothing, since you are using the post-increment operator (++ after the variable), the value returned is the value of y before the increment. Therefore, this expression does not affect the value of x.*

\*z++; // same

x++;

cout << x << " " << y << " " << z; // *x = 21, y = 500, z = 500 (500 is the address of x).*

**3.39. Assignment 39:**

int x = 10;

int\* y, \*\* z;

y = &x;

z = &y; // z points to the pointer y

cout << x << " " << \*y << " " << \*\*z;

=> The result will be x = 10, y = 10, z = 10.

**3.40. Assignment 40:**

char a[] = { 'A', 'B', 'C', 'D' };

char\* ppp = &a[0]; /// ppp points to a[0]

\*ppp++; // ppp now points to a[1]

cout << \*++ppp << endl; // ++ppp means ppp now points to a[2], then operator \* means take the value of current ppp, which is ‘C’.

cout << -- \* ppp; // this line means decrease the value of ppp by 1, means that the result on the screen will be ‘B’.

**3.41. Assignment 41:**

A pointer has to point a something when declaring it.

**3.42. Assignment 42:**

int a = 36;

int\* ptr;

ptr = &a; // ptr now points to a

cout << \*&ptr << " " << &\*ptr; // first, \*&ptr means take the value of address of ptr, and the second clause means take the address of the value of ptr. They’re both address.

**3.43. Assignment 43:**

The addresses are the same because they’re all the address of variable num in the memory.

**3.44. Assignment 44:**

Since k is the address of pointer j, and \*k is the address of i in the memory; and \*\*k is the value of varible i (because k points to j, and j points to i).

**3.45. Assignment 45:**

Thẻ’re all the value of float x (y points to x, and z = y, means z also points to x).

**3.46. Assignment 46:**

int track[] = { 10, 20, 30, 40 }, \* striker;

striker = track; // striker now points to track[0]

track[1] += 30; // track[1] = 20 + 30 = 50.

cout << "Striker > " << \*striker << endl; // print out track[0] = 10.

\*striker -= 10; // means now, track[0] = 10 – 10 = 0 (since striker now points to track[0]

striker++; // striker now points to track[1]

cout << "Next@ > " << \*striker << endl; // print out 50, since track[1] = 50

striker += 2; // striker now points to track[1]

cout << "Last@ > " << \*striker << endl; // print out 40, since track[3] = 40

cout << "Reset to " << track[0]; // print out 0, since track[1] = 0.

**3.47. Assignment 47:**

Because *cho* is the pointer, which points to *ch* (the varible with initial value is ‘A’(means 65 in ASCII)), after line *cho += a* then the value of pointer *ch* is 65 + 32 = 97.

Same, *ptr* now points to *a*, the line *\*ptr += ch* means the current value of *ptr* (it’s also the current value of *a*) is 32 + 97 = 129.

About *ch*, *ch* now has the value in integer is 97, means that the character ‘a’.

**3.48. Assignment 48:**

Variable *i* is a constant, so that it cannot be changed the value.

**3.49. Assignment 49:**

char\* str[] = {"AAAAA", "BBBBB", "CCCCC", "DDDDD"}; // \*str is a array pointer of string.

char\*\* sptr[] = { str + 3, str + 2, str + 1, str }; // pstr points to str strings in reverse order.

char\*\*\* pp; // \*\*\*p is a pointer that points to sptr

pp = sptr; // now pp points to sptr[0] = str + 3

++pp; // now pp points to sptr[1] = str + 2 (“CCCCC”)

cout << \*\*++pp + 2; // ++pp means now pp points to sptr[2] = str + 1 (“BBBBB”). And (\*\*++pp) + 2 points the 2nd index of “BBBBB” => BBB will be result on the screen.

**3.50. Assignment 50:**

The first loop means that, for each i, we swap the value of x[i] and x[4 – i], so that when we print out the array of x, the array will be in the inverse order.