PRACTICE 7.1:

|  |
| --- |
| Given the following code: |
| int main() {  int \*x, y = 2;  float \*z = &y;  printf(“%d”, y);  } |

|  |
| --- |
| 1. Fix error of code   int main() {  int \*x = new int, y = 2;  int \*z = &y;  printf(“%d”, y);  } |
| 1. After fixing, what is displayed on screen?   Ouput: 2 (value of *y*) |

PRACTICE 7.2:

|  |
| --- |
| Explain the difference amongst the following 3 functions: |
| void swap1(int x, int y)  {  int temp = x;  x = y;  y = temp;  } |
| void swap2(int &x, int &y)  {  int temp = x;  x = y;  y = temp;  } |
| void swap3(int \*x, int \*y)  {  int temp = \*x;  \*x = \*y;  \*y = temp;  } |

Explain the differecne:

**Swap1:** int *x* and int *y* are parameters passed by value, meaning *x* and *y* are just copies of the passed parameters, so when swapping the values ​​of *x* and *y* in the *swap1* function, the values ​​of the two passed variables will not change.

Swap2: int *&x*, int *&y* are parameters passed by reference, meaning that after swapping the 2 values ​​*x, y* in the *swap2* function, the values ​​of the 2 passed variables will change.

Swap3: int *\*x*, int *\*y* passed in are 2 pointers storing the addresses of 2 variables, so when swapping the values ​​of 2 pointers, the values ​​of the 2 variables that the 2 pointers point to will be changed.

PRACTICE 7.3:

|  |
| --- |
| Explain what the following program prints: |
| int main() {  double m[100];  double \*p1, \*p2;  p1 = m;  p2 = &m[6];  printf(“%lld”, p2 – p1);  } |

Explain: The program will print the distance between the two pointers p1 and p2, which means it will print the number of elements from m[0] to m[5] so it will print 6.

PRACTICE 7.4:

|  |
| --- |
| Explain what the following program prints: |
| #include <stdio.h>  int main() {  int x = 1023;  char \*p = (char \*) &x;  printf(“%d %d %d %d\n”, p[0], p[1], p[2], p[3]);  } |

Explain:

The number 1023 of variable *x* will be analyzed into 4 bytes because variable *x* has data type *int*. So, 1023 will be stored in memory as

Byte 0: 255

Byte 1: 3

Byte 2: 0

Byte 3: 0

But because *char* accepts values ​​from -128 to 127, 255 will be printed as the number -1

Therefore, the output screen will be -1 3 0 0