Zhenming (Mark) Yang

Prof. Z

Embedded Design

November 7th 2023

Homework 3

Programming:

```
-bash-4.2$ ./bits
Input the first integer: hi
Incorrect data type! Please try again: hello
Incorrect data type! Please try again: 5
Input the second integer: -12
The integer 5 in binary is: 00000000000000000000000000000101
The integer -12 in binary is: 11111111111111111111111111110100
5 is not divisible by 4 and is positive.
-12 is divisible by 4 and is negative.
The integer 5 in binary is: 00000000000000000000000000000101
The integer 5 in binary is: 00000000000000000000000000000101
-bash-4.2$
```

The program successfully send an error message when the user inputs anything that is not int and prompts another input. It also successfully converts the integer into binary format in two's complement. It also checks if the number is divisible by 4 and if it is positive (or negative) by checking the specific bit. It also manipulates the last bit of the first number to a 1 and print the new number in its binary format.

Short Answers

A.

```
struct Money
{
    int value;
    string name;
    float exchange_rate;
};
```

```
bool worthMore(Money m1, Money m2)
{
    return m1.value * m1.exchange_rate > m2.value * m2.exchange_rate;
}
```

B. $4 * 2 * 10 = 80$. The size of one int is 4 bytes. Point object has 2 int which makes it 8 bytes. And the array has 10 Points which makes 80.

C. The function created the Point struct on the stack instead of the heap. Therefore, the program can't access the data stored in result even if we passed the address out. To fix this problem, we can either return the object value out:

```
Point init(int x, int y)
{
    Point result;
    result.x = x;
    result.y = y;
    return result;
}
```

Or create the pointer struct on the Heap to the pointer outside the function:

```
Point *init(int x, int y)
{
    Point *result = new Point();
    result->x = x;
    result->y = y;
    return result;
}
```

D. The new/delete calls the constructor and destructor when dynamically allocating memory for the object. However, malloc/free doesn't do that which can cause errors and even memory leaks.

E. Link list does not need to change the size which requires copying and pasting the old data to a bigger array. This process takes time and memory and link list completely negate that. However, it is harder to access the stored data in a link list because the objects are all located separately in the memory which requires the program to go through each linked item to get to the specified index.

F.

```
struct DLL
{               // A Doubly Linked List (has forward and backward links)
    DLL *left;  // The node to my left (nullptr if first element)
    DLL *right; // The node to my right (nullptr if last element)
    int val;    // my value
};
/**
 * Inserts val as the idx element of list (counting from 0).
 * Returns a pointer to the beginning of the list.
 */
DLL *insert(DLL *list, int val, unsigned int idx)
{
    if (!list)
    {
        return nullptr;
    }
    DLL *tmp = list;
    if (!idx)
    {
        DLL *new_node = new DLL();
        new_node->val = val;
        new_node->left = nullptr;
        new_node->right = tmp;
        tmp->left = new_node;
        return new_node;
    }
    while (idx > 1 && tmp->right)
    {
        tmp = tmp->right;
        idx--;
    }
    if (idx > 1)
    {
        std::cerr << "Error! Index was too big" << std::endl;
        return list;
    }
    DLL *new_node = new DLL();
    new_node->val = val;
    new_node->left = tmp;
    new_node->right = tmp->right;
    if (tmp->right)
    {
        tmp->left = new_node;
    };
    tmp->right = new_node;
    return list;
}
```