CS 31 HW

Melody Chen

#705120273

#include <iostream>  
    using namespace std;  
      
    int main()  
    {  
        int arr[4] = { 0, 1, 2, 3 };  
        int\* ptr = arr;

        \*ptr = arr[ 1 ];               // set arr[0] to 1  
        \*(ptr + 1) = arr[ 0 ] \* 10;    // set arr[1] to 10  
        ptr += 2;   
        ptr[0] = arr[ 1 ] \* 10;        // set arr[2] to 100  
        ptr[1] = 1000;                 // set arr[3] to 1000

ptr ++;

        while (ptr >= arr)  
        {  
             
            cout << " " << \*ptr;    // print values

ptr--;   
   }  
   cout << endl;   
   return( 0 );   
}

**#include <iostream>  
    using namespace std;**  
    void findLastZero(int arr[], int n, int\*& p)  
    {  
        p = nullptr;    /// default value if there isn't a 0 in the array at all  
        for (int k = n - 1; k >= 0; k--)   
        {  
            if (arr[k] == 0)      // found an element whose value is 0  
            {   
                 p = arr + k;     // change the value of p  
                 break;           // stop looping and return  
            }   
        }  
    }

    int main()  
    {  
        int nums[6] = { 10, 20, 0, 40, 30, 50 };  
        int\* ptr;

        findLastZero(nums, 6, ptr);  
        if (ptr == nullptr)  
        {  
            cout << "The array doesn't have any zeros inside it." << endl;   
        }   
        else  
        {  
            cout << "The last zero is at address " << ptr <<  endl;  
            cout << "It's at index " << ptr - nums << endl;   
            cout << "The item's value is " << \*ptr << " which is zero!" << endl;  
        }   
        return( 0 );   
    }

The function will not work correctly because the pointer was passed to the findLastZero function by value instead of by reference. So regardless of what you set the pointer to point at in the function, it does not affect the pointer in the main function. The fix to this problem is to send the pointer by reference shown above.

 #include <iostream>  
    using namespace std;

    void biggest(int value1, int value2, int \* resultPtr)  
    {  
        if( value1 > value2 )   
        {  
               \*resultPtr = value1;  
        }   
        else  
        {  
               \*resultPtr = value2;   
        }  
    }

    int main()  
    {

int x = 0;  
        int\* p~~;~~ = &x;  
        biggest(15, 20, p);   
        cout << "The biggest value is " << \*p << endl;  
        return( 0 );  
    }

The function above may not work without the fix shown in blue because originally the pointer is pointing at a garbage location, and the function sets the garbage location to a value, which will result in the program not working correctly. Therefore, if we set the pointer to point at a valid location as shown by the fix, the program will function correctly.



bool match(const char str1[], const char str2[])  
    {   
        bool result = true;   
        while (\*str1 != ‘\0’  &&  \*str2 != ‘\0’)  // zero bytes at ends   
        {  
            if (\*str1 != \*str2)  // compare corresponding characters  
            {  
                result = false;   
                break;   
            }   
            str1++;            // advance to the next character  
            str2++;  
        }   
        if (result)    
        {   
            result = (\*str1 == \*str2);    // both ended at same time?  
        }  
        return( result );  
    }

    int main()  
    {  
        char a[10] = "pointy";  
        char b[10] = "pointless"; 

        if (match(a,b))  
        {  
            cout << "They're the same!" << endl;  
        }  
    }

The problem with the function above is that when it is comparing the c-strings, instead of comparing the characters, it compares the location of the pointer. We can fix the function above by walking the arrow of the pointer so that it compares the actual character values.

1. The issue with the program is a scoping error. As the array arr in the computeFibonacci function, is created inside the function, it cannot be passed back because the compiler will automatically delete that pointer and what its pointing to after the function ends.
2. 3 – A, 4,7- B, 6 – C, 5- D, 8 – e, 1 – f, 2 – g, 9-h

 #include <iostream>  
    using namespace std;   
  
    int\* minimart(int\* a, int\* b)  
    {  
        if (\*a < \*b)  
            return a;  
        else   
            return b;  
    }

    void swap1(int\* a, int \*b)  
    {  
        int\* temp = a;  
        a = b;   
        b = temp;  
    }

    void swap2(int\* a, int \*b)  
    {  
        int temp = \*a;  
        \*a = \*b;   
        \*b = temp;  
    }

    int main()  
    {  
        int array[6] = { 5, 3, 4, 17, 22, 19 };

        int\* ptr = minimart(array, & array[2]);   
        ptr[1] = 9;  
        ptr += 2;  
        \*ptr = -1;   
        \*(array+1) = 79;

        cout << "diff=" << &array[5] - ptr << endl;

//This line will output 1 because the current position of the pointer is at array[4].

        swap1(&array[0], &array[1]);  
        swap2(array, &array[2]);

        for (int i = 0; i < 6; i++)  
            cout << array[i] << endl;

//This line will output 4, 79, 5, 9, -1, 19. Swap 1 does not do anything as a copy of the pointer is passed in and that copy is being altered. Swap 2 will swap the value and result in the above output as the values the pointers are pointing to are being changed.

        return( 0 );  
    }

**void** deleteDigits(**char** str[]){

**char**\* temp;

**while**(\*str != '\0'){

**if**(!isdigit(\*str)){

str++;

}**else**{

temp = str;

**while**(\*str!='\0'){

\*str=\*(str+1);

str++;

}

str = temp;

}

}