Project 6

1.a)

int main()

{

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

\*ptr = 30;

\*ptr + 1 = 20; // BUG! Cannot assign value to another value.

ptr += 2;

ptr[0] = 10;

while (ptr >= arr) // BUG! Traverses the array from the end.

{

ptr--;

cout << \*ptr << endl; // print values

}

}

**Corrected code:**

int main()

{

    int arr[3] = {5,10,15};

    int\* ptr = arr;

    \*ptr = 30; // set arr[0] to 30

    \*(ptr+1) = 20; // set arr[1] to 20

    ptr += 2;

    ptr[0] = 10; // set arr[2] to 10

    ptr -= 2;

    while(ptr <= &arr[2])

    {

        cout << \*ptr << endl;

        ptr++;

    }

}

1.b) The function will not set the pToMax parameter to point to the maximum item in the array so that the caller knows that item’s location as the pointer is passed by value and not by reference. Hence, a copy of the pointer is made by the function. This copy is then made to point to the maximum item in the array. However, this change in the pointer does not reflect in the actual parameter passed to the function. Thus, the pointer must be passed by reference for the changes to reflect in the actual parameter after the function is called.

void findMax(int arr[], int n, int\*& pToMax)

{ //changed to pass by reference

    if (n <= 0)

        return;      // no items, no maximum!

    pToMax = arr;

    for (int i = 1; i < n; i++)

    {

        if (arr[i] > \*pToMax)

            pToMax = arr + i;

    }

}

int main()

{

    int nums[4] = { 50, 3, 15, 20 };

    int\* ptr;

    findMax(nums, 4, ptr);

    cout << "The maximum is at address " << ptr << endl;

    cout << "It's at position " << ptr - nums << endl;

    cout << "Its value is " << \*ptr << endl;

}

1.c) The program will not work as required as the integer pointer ptr in the main function is uninitialized during the function call. Following an uninitialized pointer is undefined behavior as it can point to any random memory location. Thus, the pointer must be initialized to point to a certain memory location before being used.

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

int\* ptr= new int;

// pointer is dynamically initialized

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

delete ptr;

// deletes the dynamic memory pointed to by the pointer

}

1.d) A C string is an array of characters. When an array is passed to a function it is converted to a pointer to the base address of the actual array. In the implementation of the function, we are comparing the addresses that the pointers are pointing to instead of the values at those addresses. Since the arrays do not exist in the same memory locations, these comparisons will never result in a true value. Thus, the pointers must be dereferenced in order to carry out the comparisons.

// return true if two C strings are equal

bool strequal(const char str1[], const char str2[])

{

    while (\*str1 != 0  &&  \*str2 != 0) //pointers are dereferenced

    {

        if (\*str1 != \*str2)  // compare corresponding characters

            return false;

        str1++;            // advance to the next character

        str2++;

    }

    return \*str1 == \*str2;   // both ended at same time?

}

int main()

{

    char a[15] = "Zhu";

    char b[15] = "Zhu";

    if (strequal(a,b))

        cout << "They're the same person!\n";

    else

        cout << "They're not the same person!\n";

}

1.e) The program is creating a pointer to an integer array that is local only to the specific function (local variable) in which it is declared and initialized. This local variable is deleted as soon as the function is done executing. (It goes out of scope) Thus, the pointer points to a memory location that may be used to store different data at any point of time. Thus, the program will not work as required.

2.a) double\* cat;

2.b) double mouse[5];

2.c) cat= &mouse[4];

2.d) \*cat=25;

2.e) \*(mouse+3)=42;

2.f) cat-=3;

2.g) cat[1]=54;

2.h) cat[0]=27;

2.i) bool b= (\*cat == \*(cat+1));

2.j) bool d= (cat == mouse);

3.a) double mean(const double\* scores, int numScores)

{

double tot = 0;

int i=0;

while (i!=numScores)

{

tot += \*(scores+i);

i++;

}

return tot/numScores;

}

3.b) // This function searches through str for the character chr.

// If the chr is found, it returns a pointer into str where

// the character was first found, otherwise nullptr (not found).

const char\* findTheChar(const char\* str, char chr)

{

for (int k = 0; \*str!= 0; k++,str++)

if (\*str == chr)

return str;

return nullptr;

}

3.c) const char\* findTheChar(const char\* str, char chr)

{

while(\*str!=0)

{

if(\*str==chr)

return str;

str++;

}

return nullptr;

}

4)

Dry run:

array= {5,3,4,17,22,19}

*int\* ptr = maxwell(array, &array[2]);*

Returns array as 5>4

Therefore, ptr=array

*\*ptr=-1;*

Value at ptr=-1

Therefore, array= {-1,3,4,17,22,19}

*ptr+=2;*

Now, ptr=&array[2]

*ptr[1]=9;*

Pointer considers the address it points to, to be its base address

Therefore, value at ptr[1]=9

Therefore, array= {-1,3,4,9,22,19}

*\*(array+1)=79;*

The value one position to the right of array’s base address is changed to 79

Therefore, array= {-1,79,4,9,22,19}

*cout << &array[5] – ptr << endl;*

The difference between the two memory addresses is printed

The two memory locations are &array[5] and ptr=&array[2] which have a difference of 3

Therefore, 3 is printed and the cursor is moved to the new line

Output: 3

*swap1(&array[0], &array[1]);*

Copies of the pointers to the two memory addresses are made to point to each other’s memory address.

No tangible change.

*swap2(array, &array[2]);*

The values at the memory locations array and &array[2] are swapped

Therefore, array= {4,79,-1,9,22,19}

*for (int i = 0; i < 6; i++)*

*cout << array[i] << endl;*

The elements of array are printed on separate lines

Therefore,

**Final Output:**

3

4

79

-1

9

22

19

5)

void removeS(char\* str)

{

    while(\*str!=0)

    {

        while(\*str=='s' || \*str=='S')

        {

            char\* i=str;

            while(\*i!=0)

            {

                \*i=\*(i+1);

                i++;

            }

        }

        str++;

    }

}