***Project 6***

1 a)

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

while (ptr >= arr)

{ ptr--;

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

}

}

Fixed:

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=arr; //ptr points to first element again

while (ptr <= (arr+2))

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

ptr++;

}

}

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

{ 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] = { 5, 3, 15, 6 };

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;

}

This function does not print the right output because the pointer that we send as a parameter is not initialized to anything (not pointing to anything), thus no change is made to any address.

Revised:

void findMax(int arr[], int n, int\* &pToMax) /\*passing by reference so that the changed value is printed\*/

{ 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] = { 5, 3, 15, 6 };

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;

}

c)

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{ int\* ptr;

computeCube(5, ptr);

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

}

The pointer ptr is uninitialized to anything. It does not point to any address hence the value changed in the function does not affect any variable/address.

Revised:

int main()

{ int\* ptr;

int m;

ptr=&m;

computeCube(5, ptr);

cout << "Five cubed is " << m << endl;

}

d)

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

{

while (str1 != 0 && str2 != 0)

{

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] = "Noor";

char b[15] = "Noah";

if (strequal(a,b))

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

}

str1 and str2 cannot be used to compare characters as they represent c-strings and not individual characters, so we need to use pointers that point to each character of the C string.

Revised:

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

{

while (\*str1 != ‘\0’ && \*str2 != ‘\0’)

{

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] = "Noor";

char b[15] = "Noah";

if (strequal(a,b))

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

}

e)

In the main function, ptr is assigned the address of anArray from another function. But anArray is not present in int main (it’s local to its function) and that may cause it to access junk values from other functions with the same addresses.

2

1. double \*cat;
2. double mouse[5];
3. cat=&mouse[4];
4. \*cat=42;
5. \*(mouse+3)=25;
6. cat-=3;
7. cat[1]=17;
8. cat[0]=54;
9. bol b=( \*cat == \*(cat+1));
10. bol d= (cat==mouse);

3

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

{ const double\* ptr = scores;

double tot = 0;

int t=0;

while (t!=numScores)

{ tot += \*(ptr+t);

t++;

}

return tot/numScores;

}

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

{ for (int k = 0; \*(str+k) != ‘\0’; k++)

if (\*(str+k) == chr)

return (str+k);

return nullptr;

}

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

{ for (; \*str != ‘\0’; str++)

if (\*str == chr)

return str;

return nullptr;

}

4.

**3**

**4**

**79**

**-1**

**9**

**22**

**19**

The first line prints 3 for &array[5] – ptr as the difference of the address of position 6 of the array and the last position of ptr which points to the 3rd position is an integer i.e. 3 (difference of 3 positions)

The initial array is 5, 3, 4, 17, 22, 19. Then the value of the pointer pointing to the first position is changed to -1 so the new array is -1, 3, 4, 17, 22, 19. The pointer is incremented to point to the 3rd element. ptr[1]=9 changes the 4th element to 9 as the pointer points at the 3rd element. Also, the 2nd element (array+1) is changed to 79 since array is the address of the first element. So the final array is -1, 79, 4, 9, 22, 19. Then the first swap is called (swap1) which swaps the addresses of the first and second elements but does not affect the array, as it does not swap values (and not affecting the addresses in the main function). The second swap, swaps the values of the first and third elements of the array making the array 4, 79, -1, 9, 22, 19 and printing each element in each line.

5.

void removeS(char\* s)

{

char \*a=s;

for(; \*s!='\0'; s++)

{

if(\*s=='s' || \*s=='S')

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

{ \*s=\*(s+1) ;

s++;

}

s=a;

}

}

}