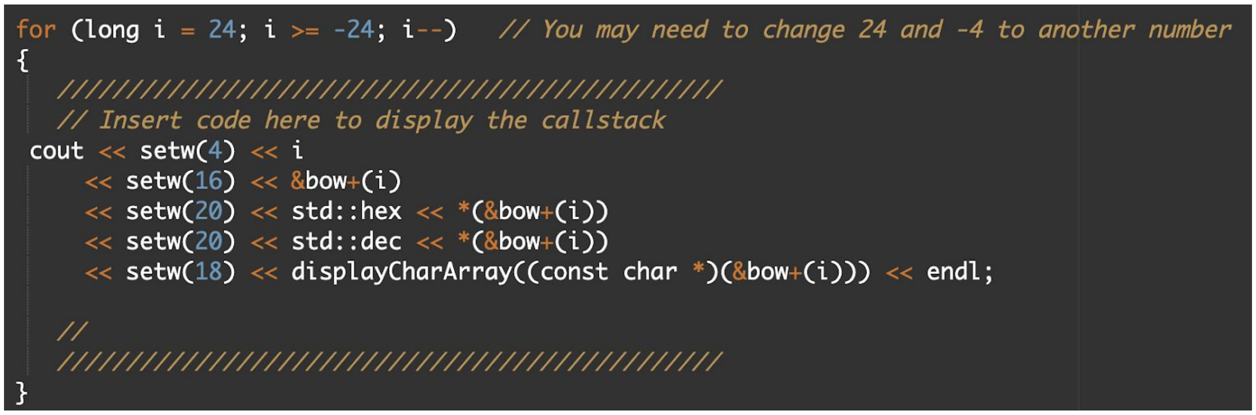
W07 - Memory Organization Lab

Ryan Arveseth, Noah Cook, Tyler DeFreitas, Logan Holland, Scott Malin, Joseph Olsen

Code Segments:

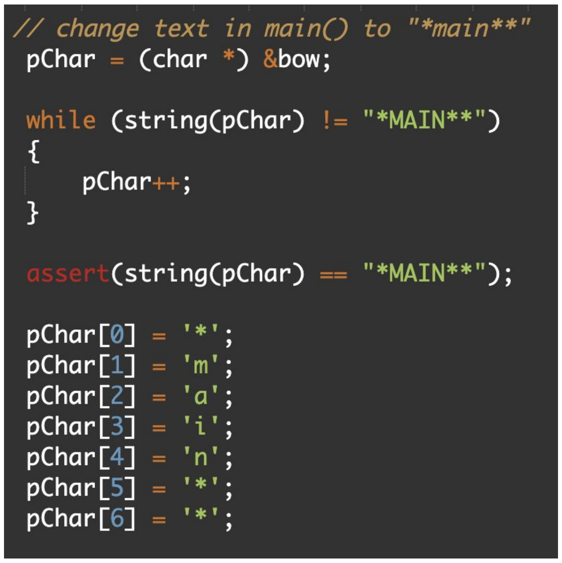
Display the Callstack:



```
for (long i = 24; i >= -24; i--)   // You may need to change 24 and -4 to another number
{
   ////////////////////////////////////////////////
   // Insert code here to display the callstack
  cout << setw(4) << i
      << setw(16) << &bow+(i)
      << setw(20) << std::hex << *(&bow+(i))
      << setw(20) << std::dec << *(&bow+(i))
      << setw(18) << displayCharArray((const char *)(&bow+(i))) << endl;

   //
   ////////////////////////////////////////////////
}
```

Description	This code segment displays the address of &bow+(i), then converts and displays it in hex, dec, and char array format.

Changing text from "*MAIN" to "*main**":**

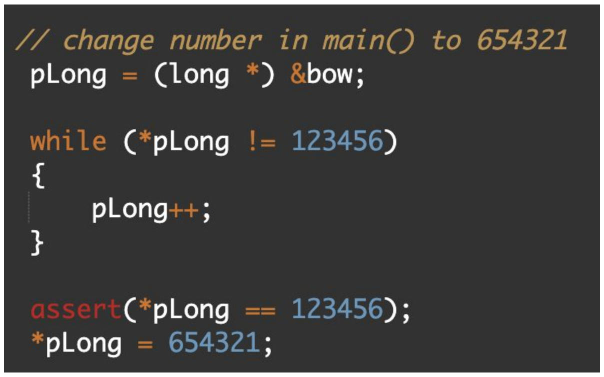


```
// change text in main() to "*main**"
 pChar = (char *) &bow;

 while (string(pChar) != "*MAIN**")
 {
     pChar++;
 }

 assert(string(pChar) == "*MAIN**");

 pChar[0] = '*';
 pChar[1] = 'm';
 pChar[2] = 'a';
 pChar[3] = 'i';
 pChar[4] = 'n';
 pChar[5] = '*';
 pChar[6] = '*';
```

Description	Most of these variable changes are similar: loop through the callstack and find the value

	we are trying to change, then change it to the new value.

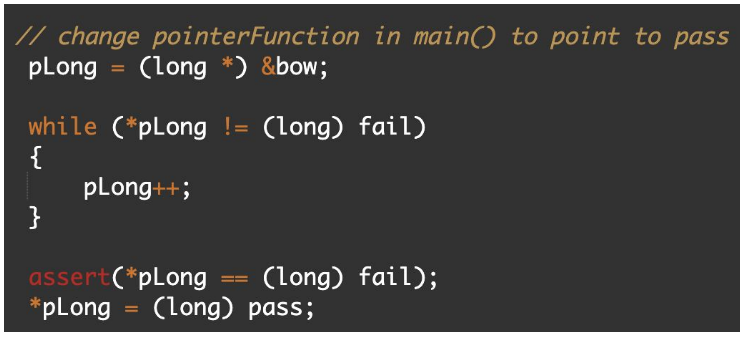
Change number from 123456 to 654321:



```
// change number in main() to 654321
 pLong = (long *) &bow;

 while (*pLong != 123456)
 {
     pLong++;
 }

 assert(*pLong == 123456);
 *pLong = 654321;
```

Description	Loop through the callstack and find the value we are trying to change - in this case, number, then change it to the new value, 123456.

Point pointerFunction to pass():



```
// change pointerFunction in main() to point to pass
 pLong = (long *) &bow;

 while (*pLong != (long) fail)
 {
     pLong++;
 }

 assert(*pLong == (long) fail);
 *pLong = (long) pass;
```

Description	Locate the pointerFunction which is pointing to fail currently, then update it to point to the pass function.

Point message to passMessage():

```
// change message in main() to point to passMessage
 pLong = (long *) &bow;

 while (*pLong != (long) failMessage)
 {
     pLong++;
 }

 assert(*pLong == (long) failMessage);
 *pLong = (long) passMessage;
```

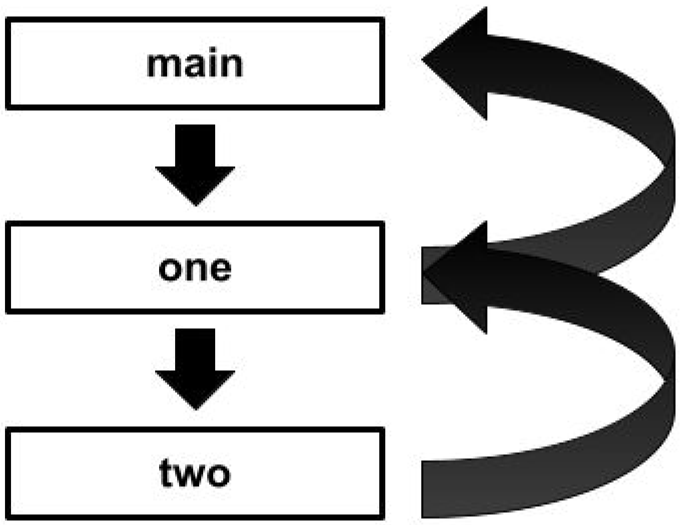
Description	Locate the message, failMessage in the stack, then change it to point to passMessage.

Display Segment Addresses

```
void DisplaySegmentAddresses()
{
    struct test_struct
    {
        int i;
        int ii;
    };

    test_struct* heap_struct = new test_struct();

    char* stack_ptr = NULL;
    test_struct** heap_ptr = &heap_struct;
    void (*code_ptr)() = &DisplaySegmentAddresses;

    std::cout << std::endl
              << "Stack Address: " << &stack_ptr << std::endl
              << "Heap Address: " << &(*(*heap_ptr)) << std::endl
              << "Code Segment Address: " << &(*code_ptr) << std::endl << std::endl;

}
```

Description:	Displays three addresses. The address of the null pointer is for the stack, the addresses of the dynamically allocated struct for the heap, and the address of the function for the code segment.

Graph of Stack



Identifying Call Stack Items

char text[8]- The char array named text is located at index and address
- (23, 0x7fff8bf68248).

number- This variable is shown multiple times on the display. The indexes and addresses are
- (19, 0x7fff8bf68228). When it is initialized.
- (13, 0x7fff8bf681f8). When it is passed to one().
- (-1, 0x7fff8bf681f8). When it is passed to two().

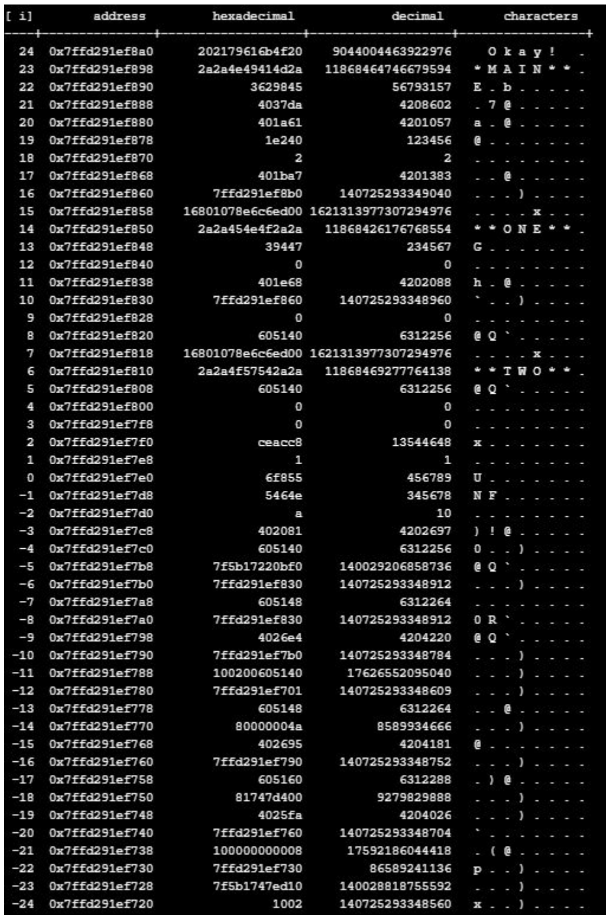
const char * message- The const char array named message is located at the index and address:
- (21 ,0x7fff8bf68238).

pointlessNumber- The int named pointlessNumber is located at the index and address:
- (22, 0x7fff8bf68240).

pointlessText- The char array named pointlessText is located at the index and address:
- (24, 0x7fff8bf68250).

Return address/frame pointer for main()- This is located at the index and address
- (15, 0x7fff8bf68208).

Return address/frame pointer one()- This is located at the index and address
- (7,   0x7fff8bf681c8.)



```
[ i]       address         hexadecimal            decimal        characters
----+----------------+------------------+-------------------+-----------------+
  24  0x7ffd291ef8a0     202179616b4f20    9044004463922976     O k a y !   .
  23  0x7ffd291ef898     2a2a4e49414d2a   11868464746679594   * M A I N * * .
  22  0x7ffd291ef890            3629845            56793157   E . b . . . . .
  21  0x7ffd291ef888             4037da             4208602   . 7 @ . . . . .
  20  0x7ffd291ef880             401a61             4201057   a . @ . . . . .
  19  0x7ffd291ef878              1e240              123456   @ . . . . . . .
  18  0x7ffd291ef870                  2                   2   . . . . . . . .
  17  0x7ffd291ef868             401ba7             4201383   . . @ . . . . .
  16  0x7ffd291ef860       7ffd291ef8b0     140725293349040   . . . ) . . . .
  15  0x7ffd291ef858   16801078e6c6ed00 1621313977307294976   . . . . x . . .
  14  0x7ffd291ef850     2a2a454e4f2a2a   11868426176768554   * * O N E * * .
  13  0x7ffd291ef848              39447              234567   G . . . . . . .
  12  0x7ffd291ef840                  0                   0   . . . . . . . .
  11  0x7ffd291ef838             401e68             4202088   h . @ . . . . .
  10  0x7ffd291ef830       7ffd291ef860     140725293348960   ` . . ) . . . .
   9  0x7ffd291ef828                  0                   0   . . . . . . . .
   8  0x7ffd291ef820             605140             6312256   @ Q ` . . . . .
   7  0x7ffd291ef818   16801078e6c6ed00 1621313977307294976   . . . . x . . .
   6  0x7ffd291ef810     2a2a4f57542a2a   11868469277764138   * * T W O * * .
   5  0x7ffd291ef808             605140             6312256   @ Q ` . . . . .
   4  0x7ffd291ef800                  0                   0   . . . . . . . .
   3  0x7ffd291ef7f8                  0                   0   . . . . . . . .
   2  0x7ffd291ef7f0             ceacc8            13544648   x . . . . . . .
   1  0x7ffd291ef7e8                  1                   1   . . . . . . . .
   0  0x7ffd291ef7e0              6f855              456789   U . . . . . . .
  -1  0x7ffd291ef7d8              5464e              345678   N F . . . . . .
  -2  0x7ffd291ef7d0                  a                  10   . . . . . . . .
  -3  0x7ffd291ef7c8             402081             4202697   ) ! @ . . . . .
  -4  0x7ffd291ef7c0             605140             6312256   0 . . ) . . . .
  -5  0x7ffd291ef7b8       7f5b17220bf0     140029206858736   @ Q ` . . . . .
  -6  0x7ffd291ef7b0       7ffd291ef830     140725293348912   . . . ) . . . .
  -7  0x7ffd291ef7a8             605148             6312264   . . . . . . . .
  -8  0x7ffd291ef7a0       7ffd291ef830     140725293348912   0 R ` . . . . .
  -9  0x7ffd291ef798             4026e4             4204220   @ Q ` . . . . .
 -10  0x7ffd291ef790       7ffd291ef7b0     140725293348784   . . . ) . . . .
 -11  0x7ffd291ef788       100200605140       17626552095040   . . . ) . . . .
 -12  0x7ffd291ef780       7ffd291ef701     140725293348609   . . . ) . . . .
 -13  0x7ffd291ef778             605148             6312264   . . @ . . . . .
 -14  0x7ffd291ef770          80000004a          8589934666   . . . ) . . . .
 -15  0x7ffd291ef768             402695             4204181   @ . . . . . . .
 -16  0x7ffd291ef760       7ffd291ef790     140725293348752   . . . ) . . . .
 -17  0x7ffd291ef758             605160             6312288   . ) @ . . . . .
 -18  0x7ffd291ef750          81747d400          9279829888   . . . ) . . . .
 -19  0x7ffd291ef748             4025fa             4204026   . . . ) . . . .
 -20  0x7ffd291ef740       7ffd291ef760     140725293348704   ` . . . . . . .
 -21  0x7ffd291ef738       100000000008       17592186044418   . ( @ . . . . .
 -22  0x7ffd291ef730       7ffd291ef730         86589241136   p . . ) . . . .
 -23  0x7ffd291ef728       7f5b1747ed10     140028818755592   . . . ) . . . .
 -24  0x7ffd291ef720               1002     140725293348560   x . . ) . . . .
```