CMP SCI 3780 Project 1

Fall 2020

Due date: 9/27/2020

Total points possible: 65

The goal of this project is to reinforce what is going on in the call stack, as well as refreshing ourselves on the unix environment and using a debugger to examine memory locations. All tasks should be done on the student unix server delmar.umsl.edu. If you have never used a debugger to examine raw memory locations, now is the time!

Task 1

Your task here is to examine the size of the activation record for a function. In particular, I want to know the size in bytes on the stack that is taken up by all the data that is not simply local variables being stored. To do this, write a C++ program that contains several functions that call each other in a sequence. These functions should take in at least one variable by value. It would also be useful to set some local variables to some specific values inside of them. Then using GDB or some other debugger, set a breakpoint at the end of the lowest-level function and examine the memory locations of the stack (probably using x in gdb). You should be able to get an understanding of how much space is taken up by non-variable data, as well as the order of the functions in memory. Note that you should keep these functions fairly simple as it will make your life easier. For demonstrating your work, I want you to write up a justification for your answer, including relevant evidence. This should include screen captures of at least some display by the debugger of memory, along with a description of its interpretation.

```
[kkdd6@delmar Project-1]$ cat task1.cpp
/*
 * By: Kenan Krijestorac
 * Professor Hauschild
 * CS 3780
 * Project 1 - Task 1
 *
 * */


#include <iostream>
using namespace std;

int addFive(int);
int multiplyByTwo(int);

int main() {

        int x = 6;
        int y = 3;
        int z = 2;

        cout << addFive(x) << endl;;

        return 0;
}

int addFive(int a) {

        int x = a + 5;
        int y = multiplyByTwo(x);

        return y;
}

int multiplyByTwo(int b) {

        int x = b * 2;

        return x;
}
```

```
(gdb)
28
29              return y;
30      }
31
32      int multiplyByTwo(int b) {
33              int x = b * 2;
34
35              return x;
36      }
(gdb)
Line number 37 out of range; taskledit.cpp has 36 lines.
(gdb) b 35
Breakpoint 1 at 0x400822: file taskledit.cpp, line 35.
(gdb) r
Starting program: /home/kkdd6/CS3780/Project-1/a.out

Breakpoint 1, multiplyByTwo (b=11) at taskledit.cpp:35
35              return x;
```

```
(gdb) bt
#0  multiplyByTwo (b=11) at taskledit.cpp:35
#1  0x000000000040080b in addFive (a=6) at taskledit.cpp:27
#2  0x00000000004007cd in main () at taskledit.cpp:20
(gdb)
```

```
(gdb) f 2
#2  0x00000000004007cd in main () at taskledit.cpp:20
20              cout << addFive(x) << endl;;
(gdb) info frame
Stack level 2, frame at 0x7fffffffe4c0:
 rip = 0x4007cd in main (taskledit.cpp:20); saved rip 0x7ffff7210555
 caller of frame at 0x7fffffffe4a0
 source language c++.
 Arglist at 0x7fffffffe4b0, args:
 Locals at 0x7fffffffe4b0, Previous frame's sp is 0x7fffffffe4c0
 Saved registers:
  rbp at 0x7fffffffe4b0, rip at 0x7fffffffe4b8
(gdb)
```

```
(gdb) f 1
#1  0x000000000040080b in addFive (a=6) at taskledit.cpp:27
27              int y = multiplyByTwo(x);
(gdb) info frame
Stack level 1, frame at 0x7fffffffe4a0:
 rip = 0x40080b in addFive (taskledit.cpp:27); saved rip 0x4007cd
 called by frame at 0x7fffffffe4c0, caller of frame at 0x7fffffffe470
 source language c++.
 Arglist at 0x7fffffffe490, args: a=6
 Locals at 0x7fffffffe490, Previous frame's sp is 0x7fffffffe4a0
 Saved registers:
  rbp at 0x7fffffffe490, rip at 0x7fffffffe498
(gdb)
```

```
(gdb) p 0x7fffffffe4c0 - 0x7fffffffe470
$1 = 80
```

The two ways that I used to analyze the stack are shown in the above screenshots. The first method that I used was to set a breakpoint in the lowest-level function, which in this case is 'multiplyByTwo' because main is being executed which then calls 'addFive' and that function calls 'multiplyByTwo'. The other method I used was to analyze the memory addresses of the stack was 'disas'. However, I found it much easier just to analyze each stack frame individually. After running the 'bt' command which displays the functions that are currently on the stack, I went ahead and analyzed each stack frame individually. Since 'main()' is the highest-level function and the stack grows down, I wanted to analyze this first. I found out that the 'main()' frame starts at the address location 0x7fffffffe4c0. The next function in the frame is 'addFive' which begins at the address location 0x7fffffffe4a0. The lowest-level function, 'multiplyByTwo' is at the address location 0x7fffffffe470. In order to calculate the size of the stack, I took the address location of the main function and subtracted it from the address location of the lowest-level function (multiplyByTwo), so the size of the stack ends up being 80 bytes. The two screenshots below show the locations of the variables and where they are stored in memory. Some of the surrounding information is non-variable data and/or junk. The x and y variables within the 'addFive' and 'main' function are stored very close to each other in memory as well.

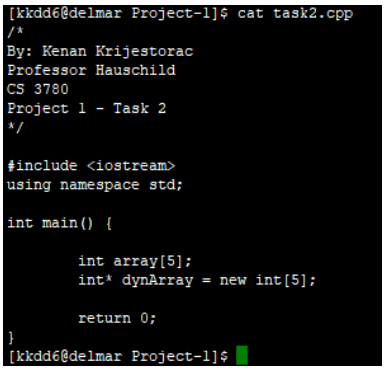
```
(gdb) info locals
x = 6
y = 3
(gdb) p &x
$1 = (int *) 0x7fffffffe4ac
(gdb) x/64db &x
0x7fffffffe4ac: 6       0       0       0       0       0       0       0
0x7fffffffe4b4: 0       0       0       0       85      5       33      -9
0x7fffffffe4bc: -1      127     0       0       0       0       0       0
0x7fffffffe4c4: 0       0       0       0       -104    -27     -1      -1
0x7fffffffe4cc: -1      127     0       0       0       0       0       0
0x7fffffffe4d4: 1       0       0       0       -83     7       64      0
0x7fffffffe4dc: 0       0       0       0       0       0       0       0
0x7fffffffe4e4: 0       0       0       0       -128    18      -75     80
(gdb) x/64db &y
0x7fffffffe4a8: 3       0       0       0       6       0       0       0
0x7fffffffe4b0: 0       0       0       0       0       0       0       0
0x7fffffffe4b8: 85      5       33      -9      -1      127     0       0
0x7fffffffe4c0: 0       0       0       0       0       0       0       0
0x7fffffffe4c8: -104    -27     -1      -1      -1      127     0       0
0x7fffffffe4d0: 0       0       0       0       1       0       0       0
0x7fffffffe4d8: -83     7       64      0       0       0       0       0
0x7fffffffe4e0: 0       0       0       0       0       0       0       0
(gdb)
```

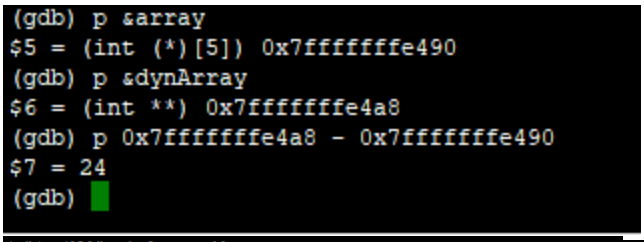
```
/*
 * By: Kenan Krijestorac
 * Professor Hauschild
 * CS 3780
 * Project 1 - Task 1
 *
 * */

#include <iostream>
using namespace std;

int addFive(int);
int multiplyByTwo(int);

int main() {
	int x = 6;
	int y = 3;
	cout << addFive(x) << endl;;
	return 0;
}

int addFive(int a) {
```

```
    int x = a + 5;
    int y = multiplyByTwo(x);
    return y;
}

int multiplyByTwo(int b) {
    int x = b * 2;
    return x;
}
```

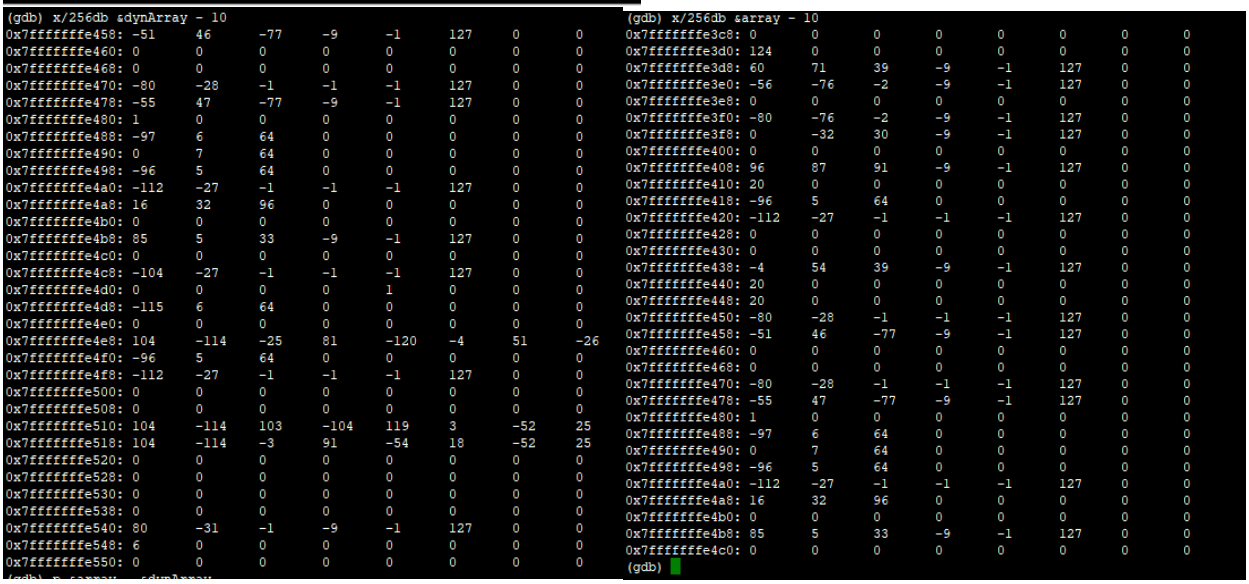
Task 2:

Write another program (in C++) that will allocate a local static array of integers and then a dynamic array of integers. Are they stored next to each other? You can examine this by examining the memory addresses where they are located. On some systems the size of a dynamic array is stored in the bytes previous to a dynamically allocated array. Through some experiments on your own, try to see if this is true on delmar. Is this true or not true also for the local array? As in the first part, describe the procedure you used to test for this.





```
(gdb) p &array
$5 = (int (*)[5]) 0x7fffffffe490
(gdb) p &dynArray
$6 = (int **) 0x7fffffffe4a8
(gdb) p 0x7fffffffe4a8 - 0x7fffffffe490
$7 = 24
(gdb)
```



```
(gdb) x/256db &dynArray - 10
0x7fffffffe458: -51     46      -77     -9      -1      127     0       0
0x7fffffffe460: 0       0       0       0       0       0       0       0
0x7fffffffe468: 0       0       0       0       0       0       0       0
0x7fffffffe470: -80     -28     -1      -1      -1      127     0       0
0x7fffffffe478: -55     47      -77     -9      -1      127     0       0
0x7fffffffe480: 1       0       0       0       0       0       0       0
0x7fffffffe488: -97     6       64      0       0       0       0       0
0x7fffffffe490: 0       7       64      0       0       0       0       0
0x7fffffffe498: -96     5       64      0       0       0       0       0
0x7fffffffe4a0: -112    -27     -1      -1      -1      127     0       0
0x7fffffffe4a8: 16      32      96      0       0       0       0       0
0x7fffffffe4b0: 0       0       0       0       0       0       0       0
0x7fffffffe4b8: 85      5       33      -9      -1      127     0       0
0x7fffffffe4c0: 0       0       0       0       0       0       0       0
0x7fffffffe4c8: -104    -27     -1      -1      -1      127     0       0
0x7fffffffe4d0: 0       0       0       0       1       0       0       0
0x7fffffffe4d8: -115    6       64      0       0       0       0       0
0x7fffffffe4e0: 0       0       0       0       0       0       0       0
0x7fffffffe4e8: 104     -114    -25     81      -120    -4      51      -26
0x7fffffffe4f0: -96     5       64      0       0       0       0       0
0x7fffffffe4f8: -112    -27     -1      -1      -1      127     0       0
0x7fffffffe500: 0       0       0       0       0       0       0       0
0x7fffffffe508: 0       0       0       0       0       0       0       0
0x7fffffffe510: 104     -114    103     -104    119     3       -52     25
0x7fffffffe518: 104     -114    -3      91      -54     18      -52     25
0x7fffffffe520: 0       0       0       0       0       0       0       0
0x7fffffffe528: 0       0       0       0       0       0       0       0
0x7fffffffe530: 0       0       0       0       0       0       0       0
0x7fffffffe538: 0       0       0       0       0       0       0       0
0x7fffffffe540: 80      -31     -1      -9      -1      127     0       0
0x7fffffffe548: 6       0       0       0       0       0       0       0
0x7fffffffe550: 0       0       0       0       0       0       0       0
(gdb) x/256db &array - 10
0x7fffffffe3c8: 0       0       0       0       0       0       0       0
0x7fffffffe3d0: 124     0       0       0       0       0       0       0
0x7fffffffe3d8: 60      71      39      -9      -1      127     0       0
0x7fffffffe3e0: -56     -76     -2      -9      -1      127     0       0
0x7fffffffe3e8: 0       0       0       0       0       0       0       0
0x7fffffffe3f0: -80     -76     -2      -9      -1      127     0       0
0x7fffffffe3f8: 0       -32     30      -9      -1      127     0       0
0x7fffffffe400: 0       0       0       0       0       0       0       0
0x7fffffffe408: 96      87      91      -9      -1      127     0       0
0x7fffffffe410: 20      0       0       0       0       0       0       0
0x7fffffffe418: -96     5       64      0       0       0       0       0
0x7fffffffe420: -112    -27     -1      -1      -1      127     0       0
0x7fffffffe428: 0       0       0       0       0       0       0       0
0x7fffffffe430: 0       0       0       0       0       0       0       0
0x7fffffffe438: -4      54      39      -9      -1      127     0       0
0x7fffffffe440: 20      0       0       0       0       0       0       0
0x7fffffffe448: 20      0       0       0       0       0       0       0
0x7fffffffe450: -80     -28     -1      -1      -1      127     0       0
0x7fffffffe458: -51     46      -77     -9      -1      127     0       0
0x7fffffffe460: 0       0       0       0       0       0       0       0
0x7fffffffe468: 0       0       0       0       0       0       0       0
0x7fffffffe470: -80     -28     -1      -1      -1      127     0       0
0x7fffffffe478: -55     47      -77     -9      -1      127     0       0
0x7fffffffe480: 1       0       0       0       0       0       0       0
0x7fffffffe488: -97     6       64      0       0       0       0       0
0x7fffffffe490: 0       7       64      0       0       0       0       0
0x7fffffffe498: -96     5       64      0       0       0       0       0
0x7fffffffe4a0: -112    -27     -1      -1      -1      127     0       0
0x7fffffffe4a8: 16      32      96      0       0       0       0       0
0x7fffffffe4b0: 0       0       0       0       0       0       0       0
0x7fffffffe4b8: 85      5       33      -9      -1      127     0       0
0x7fffffffe4c0: 0       0       0       0       0       0       0       0
(gdb)
```

No, the local static array of integers and the dynamic array of integers are not stored next to each other. The two arrays are located 24 bytes away from each other in memory. I printed the address location of both of the arrays to determine this. The array is located within the stack and the dynamically allocated array is in the heap. After analyzing the memory previous to the dynamically allocated array, I was not able to discover the size of the dynamically allocated array (I also ended up changing the size of the dynamically allocated array to see if I would be able to find the unusual number I had entered for the size, but I was still unsuccessful). I checked the memory before the array for the dynamically allocated array and the local static array and was unable to locate the size of them in the delmar system.

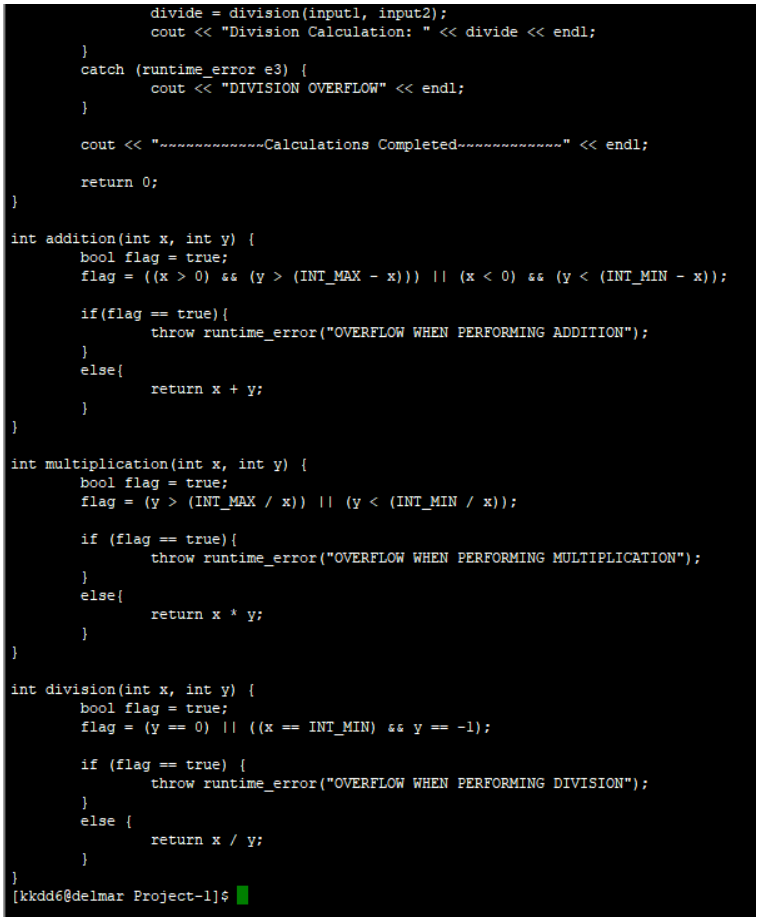
```
/*
By: Kenan Krijestorac
Professor Hauschild
CS 3780
Project 1 - Task 2
*/
#include <iostream>
using namespace std;
int main() {
    int array[5];
    int* dynArray = new int[5];
```

```
    return 0;
}
```

Task 3:

Write a program that prompts the user for two numbers and stores them in signed integers. The program should then add those two numbers together and store the result in a signed integer and display the result. Your program should then multiply them by each other and store the result in another integer and display the result. Then do the same but with dividing the first number by the second. Display an error message to the screen if an operation has happened that does not result in a correct calculation. In other words, make sure to test your code for error cases. You can safely assume your program will only be given integers.

```
[kkdd6@delmar Project-1]$ cat task3.cpp
/*
By: Kenan Krijestorac
Professor Hauschild
CS 3780
Project 1 - Task 3
*/

#include <iostream>
#include <bits/stdc++.h>
using namespace std;

int addition(int, int);
int multiplication(int, int);
int division(int, int);

int main() {

        int input1, input2, add, multiply, divide;

        cout << "Enter first integer: " << endl;
        cin >> input1;

        //Tests to see if the first integer is larger or less than the bounds of int
        if ((input1 < INT_MIN) || (input1 > INT_MAX)) {
                cout << "ERROR: OVERFLOW" << endl;
                return -1;
        }

        cout << "Enter second integer: " << endl;
        cin >> input2;

        //Tests to see if the second integer is larger or less than the bounds of int
        if ((input1 < INT_MIN) || (input1 > INT_MAX)) {
                cout << "ERROR: INPUT INTEGER OVERFLOW" << endl;
                return -1;
        }

        cout << "~~~~~~~~~~~~Calculating~~~~~~~~~~~~" << endl;

        //Checks to see if an exception is generated when addition is performed
        try {
                add = addition(input1, input2);
                cout << "Addition Calculations: " << add << endl;
        }
        catch (runtime_error e1) {
                cout << "ADDITION OVERFLOW" << endl;
        }

        //Checks to see if an exception is generated when multiplication is performed
        try {
                multiply = multiplication(input1, input2);
                cout << "Multiplication Calculation: " << multiply << endl;
        }
        catch (runtime_error e2) {
                cout << "MULTIPLICATION OVERFLOW" << endl;
        }

        //Checks to see if an exception is generated when division is performed
        try {
```



```
                divide = division(input1, input2);
                cout << "Division Calculation: " << divide << endl;
        }
        catch (runtime_error e3) {
                cout << "DIVISION OVERFLOW" << endl;
        }

        cout << "~~~~~~~~~~~~Calculations Completed~~~~~~~~~~~~" << endl;

        return 0;
}

int addition(int x, int y) {
        bool flag = true;
        flag = ((x > 0) && (y > (INT_MAX - x))) || (x < 0) && (y < (INT_MIN - x));

        if(flag == true){
                throw runtime_error("OVERFLOW WHEN PERFORMING ADDITION");
        }
        else{
                return x + y;
        }
}

int multiplication(int x, int y) {
        bool flag = true;
        flag = (y > (INT_MAX / x)) || (y < (INT_MIN / x));

        if (flag == true){
                throw runtime_error("OVERFLOW WHEN PERFORMING MULTIPLICATION");
        }
        else{
                return x * y;
        }
}

int division(int x, int y) {
        bool flag = true;
        flag = (y == 0) || ((x == INT_MIN) && y == -1);

        if (flag == true) {
                throw runtime_error("OVERFLOW WHEN PERFORMING DIVISION");
        }
        else {
                return x / y;
        }
}
[kkdd6@delmar Project-1]$
```

/*

By: Kenan Krijestorac

Professor Hauschild

CS 3780

Project 1 - Task 3

*/

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

int addition(int, int);
int multiplication(int, int);
int division(int, int);
```

```
int main() {

	int input1, input2, add, multiply, divide;

	cout << "Enter first integer: " << endl;
	cin >> input1;

	//Tests to see if the first integer is larger or less than the bounds of int
	if ((input1 < INT_MIN) || (input1 > INT_MAX)) {
		cout << "ERROR: OVERFLOW" << endl;
		return -1;
	}

	cout << "Enter second integer: " << endl;
	cin >> input2;

	//Tests to see if the second integer is larger or less than the bounds of int
	if ((input1 < INT_MIN) || (input1 > INT_MAX)) {
		cout << "ERROR: INPUT INTEGER OVERFLOW" << endl;
		return -1;
	}

	cout << "~~~~~~~~~~~~Calculating~~~~~~~~~~~~" << endl;

	//Checks to see if an exception is generated when addition is performed
	try {
		add = addition(input1, input2);
		cout << "Addition Calculations: " << add << endl;
	}
	catch (runtime_error e1) {
		cout << "ADDITION OVERFLOW" << endl;
	}
```

```
	//Checks to see if an exception is generated when multiplication is performed
	try {
		multiply = multiplication(input1, input2);
		cout << "Multiplication Calculation: " << multiply << endl;
	}
	catch (runtime_error e2) {
		cout << "MULTIPLICATION OVERFLOW" << endl;
	}

	//Checks to see if an exception is generated when division is performed
	try {
		divide = division(input1, input2);
		cout << "Division Calculation: " << divide << endl;
	}
	catch (runtime_error e3) {
		cout << "DIVISION OVERFLOW" << endl;
	}

	cout << "~~~~~~~~~~~~Calculations Completed~~~~~~~~~~~~" << endl;

	return 0;
}

int addition(int x, int y) {
	bool flag = true;
	flag = ((x > 0) && (y > (INT_MAX - x))) || (x < 0) && (y < (INT_MIN - x));

	if(flag == true){
		throw runtime_error("OVERFLOW WHEN PERFORMING ADDITION");
	}
	else{
		return x + y;
	}
```

```
}

int multiplication(int x, int y) {
    bool flag = true;
    flag = (y > (INT_MAX / x)) || (y < (INT_MIN / x));

    if (flag == true){
        throw runtime_error("OVERFLOW WHEN PERFORMING MULTIPLICATION");
    }
    else{
        return x * y;
    }
}

int division(int x, int y) {
    bool flag = true;
    flag = (y == 0) || ((x == INT_MIN) && y == -1);

    if (flag == true) {
        throw runtime_error("OVERFLOW WHEN PERFORMING DIVISION");
    }
    else {
        return x / y;
    }
}
```

Submission:

For submission I want all source code and a single document describing the experiments and the answers to the questions (and justifications) on the previous tasks. This document should be in either pdf or word. Embed any screenshots into those documents, I do not want pictures, jpeg, etc. Also include in this document any sourcecode at the end (yes, this would mean a separate source file and the source code embedded in this document also).