Cs 240 lab2
Xiao Qin
qin39@purdue.edu

Problem 1 (10 pts)

Compile and run the program in v1 multiple times. What output do you observe, and why?

The address changes because each time you run the program, the s changes the initialized address.

What convention does %p use to display addresses?

%p is used to display address of the value.

How many bits are there in the addresses displayed?

12 bits are there in the addresses displayed.

Write out the bit pattern (i.e., bit values) of the least significant 8 bits of the first run.

0f9b0c74

Problem 2 (10 pts)

Explain why after calling changeval1() in v2 the value of s printed on stdout continues to be 6 (not 3).

Because changeval1() contains only local variable and it will not store to any address, after you run it, it dies and discard the value.

What happens if int s is declared outside the scope of the main()? Explain.

It’s okay to declare s outside the scope of the main(), and nothing changes.

Problem 3 (10 pts)

Explain why after calling changeval2() in v2 the value of x printed on stdout changes to 3.

Because we change update the address of the parameter to contain the value 3.

What happens if we change the assignment "*a = 3" in changeval2() to "a = 3"?

The return value will be 6 instead of 3.

Explain what happens upon compilation and at run-time when executing a.out.

While compiling, I get warning.

```
main.c: In function ‘main’:
main.c:13:3: warning: assignment makes pointer from integer without a cast [-Wint-conversion]
 s = 6;
   ^
main.c:15:14: warning: passing argument 1 of ‘changeval1’ makes integer from pointer without a cast
[-Wint-conversion]
   changeval1(s);
              ^
main.c:9:6: note: expected ‘int’ but argument is of type ‘int *’
 void changeval1(int);
      ^~~~~~~~~~
main.c:16:12: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘int *’ [-
Wformat=]
   printf("%d\n",s);
           ~^
           %ls
main.c:18:14: warning: passing argument 1 of ‘changeval2’ makes integer from pointer without a cast
[-Wint-conversion]
   changeval2(s);
              ^
main.c:10:6: note: expected ‘int’ but argument is of type ‘int *’
 void changeval2(int );
```

```
      ^~~~~~~~~~
main.c:19:12: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘int *’ [-Wformat=]
   printf("%d\n",s);
           ~^
           %ls
```

When executing a.out I get 6 6.

Problem 4 (15 pts)

Code a function changeval3() with function definition, void changeval3(void), so that it changes the value of s from 6 to 3. Declare int s as a global variable. Put the code of changeval3() at the end of main.c. Compile and test that it works correctly.

I declared int *s outside main function to be global variable. Then, I assign s=3 in function changeval3. It works correctly.

Problem 5 (15 pts)

Modify the code in v3 so that the function modval() is placed in a separate file modval.c. Verify that everything works as it should by

(1) compiling the two .c files separately using the -c option,

(2) running gcc on the resultant .o files, and

(3) executing a.out.

If you don't like the filename a.out for the executable, what is the standard way of changing it to a different name?

mv a.out myprog

Problem 6 (15 pts)

Modify the code in v3 so that the function modval() is placed in a separate file modval.c as in Problem 5. However, modify the code in main.c so that modval.c is included using the C preprocessor statement #include "modval.c" instead of creating separate object files. Save the modified version of main.c in main1.c. Compile and test that the code works correctly.

It works correctly.

Problem 7 (15 pts)

Compile and run the code in v4. Explain the output produced by the program.

We get two values of 6 and the same address. Since y is address variable which stores x's address.

Suppose we add the assignment "z = &y" to the end of main(). How must z be declared in the program?

Z should be declared as double pointer.

How do we print the value of x using z? Assign z the address, then print **z;

Save the modified version of main.c in main1.c. Check that it compiles and runs correctly. Yes.

Suppose we add the assignment "w = &z" at the end of main() in main1.c. How must w be declared?

int ***w;

How do we print the value of x using w? Save the modified version in main2.c and check that it works correctly.

printf("%d\n",***w);

Problem 8 (10 pts)

Change the C++ program in v5 into a C program that compiles using gcc and run correctly. That is, it produces the same result as the C++ program compiled with c++. Save the modified program in main1.c.

Problem 9 (10 pts)

Compile and execute the code in v6. What is the meaning of segmentation fault?

Segmentation fault is when your program attempts to access memory it has either not been assigned by the operating system, or is otherwise not allowed to access.

What causes the fault to arise in main() when executing a.out? It means that in main function, there exists a step that attempts to access memory it has either not been assigned by the operating system, or is otherwise not allowed to access.

Add two additional #ifdef conditional debug printf() calls in the function changeval() to pinpoint where the seg fault occurs.

Problem 10 (10 pts)

Describe how the 1-D integer array s[4] in v7 is laid out in main memory (i.e., RAM). What is contained at s?

Continuously. S contains value of 100.

Problem 11 (15 pts)

Why does compiling and running the program in v8 result in segmentation fault?

Because h is declared as a pointer and it cannot contain value of integer.

Make a simple modification to the declaration of h so that a segmentation fault does not arise.

Save the modified version in main1.c.

declare h as int array.

Make a different modification to main.c that leaves all the existing legacy code intact but adds new code to fix the segmentation fault.

Save this version in main2.c.

make an int variable a and give h the address of a.

Problem 12 (25 pts)

What is the silent (or hidden) run-time error in v9?

Index out of boundary error.

What happens if you change the limit of the third for-loop from 6 to 7?

a.out adds 6 as one more output,

```
*** stack smashing detected ***: <unknown> terminated
Aborted
```

What programming practice reduces the likelihood of introducing such run-time bugs?

Be aware of boundary because c does not recognize it by default.

Modify the code in main.c that implements the programming practice to fix this silent run-time bug. Save the modified version in main1.c.

change 7 to 5

What happens if you change the limit of the third for-loop in main.c from 6 to 7 and you declare the array int a[5] as global?

The code compile correctly.

What does this say about the stack smashing alert provided by gcc?

The smashing alert means index out of boundary error which only apply to local function.

Problem 13 (15 pts)

What does the layout of character array a[7] in v10 look like in memory?

Like layers of 7 drawers linked together with sequence.

Use the string processing library function strcpy() (check its usage using man) to assign the string "1234" to a[7] and print its value. Can a[7] be used to store the string "abcdefg"? Explain.

If a[7] refers to the whole array, yes, the value is “1234”, and yes, “abcdefg” can be stored.

Otherwise, if a[7] refers to the 8th element, we will get segmentation fault. Because the 8th element is not assigned.

Problem 14 (20 pts)

Read the BUGS section of strcpy()'s man pages.

Explain what potential issue it is warning against.

The overflow issue.

How does this relate to the silent run-time bug of Problem 12?

Problem’s hidden error is caused by default strcpy().

Are the folks who coded strcpy() just not very smart or is there a more intrinsic problem underlying strcpy()'s vulnerability as a source of potential run-time errors? Explain.

If the destination string of a strcpy() is not large enough, then any-thing might happen. C is efficient but easy to cause error.

Bonus problem (15 pts)

When declaring variables (in general, data structures) in C, there is a choice between making them local versus global. Yes

What are the pros/cons?

The pros for making variable global is that you can access it in different function. The cons for making variable global is that it is not safe and secret.

The pros for making variable local is that you can keep it secret. The cons for making variable local is that you can not access it from out side.

What approach would you follow, and why?

I think it depend on the cases. If the information should be protected for some reason, we should declared them local. However, if the variable is used everywhere and it does not contain sensitive information, the declaring as global saves time.