SEC204 C Practical

We shall explore parts of the C programming language & development tools

1. Create a simple program then run it

   Using the editor program that prints “Hello World” to the screen.

```
# include <st

int main (voi
{
        printf
        return (0);
}
```

   Save the file as test.c & exit the editor.

   Look at the permissions for test.c & ensure you have executable permission.

   Now compile test.c using the GNU Common Compiler (GCC). This converts high level code into assembly language and then into instruction code for the processor to execute:

   > gcc –o test test.c

   Ensure that the executable code *test* has execute permission, then run it

   > ./test

2. Expand test.c, so that it calculates sum=x + y, where x, y, sum are integers. Display a formatted x, y and sum on the screen.

3. Copy datatype_sizes.c from /Desktop/booksrc to your home dir. Compile & run it.

   Add the following to the code

```
printf("Size of double: %u\n", sizeof(double) );
printf("Size of long double: %u\n", sizeof(long double) );
```

4. Create this program & call it sum.c – within it create a function “square”

```
#include <stdio.h>

int main() {
  int n;
  int sum;
  sum = 0;
  for (n = 0; n < 10; n++)
    sum = sum + square(n);
  printf("sum: %d\n", sum);
}
```

Try compiling it with the cc compiler

```
> cc -g -O0 sum.c -o sum
```

5. Write a programme that reads 2 inputs (an integer and a char) from standard input. Format the input (using conversion characters) & store in a specified variable. Then print them out (formatted) to standard output

   Compile and run the programme

6. Write a prograr integer as input from the screen and checks if it is divisible by 3 re notes). Then prints the results to the screen.

7. Write a prograr oating point numbers from the screen, and in a function, calcul hen returns this to the main programme for output.

8. Write a programme that creates a list of 8 randomly chosen integers between 0 & 20 finds the largest one & prints this to the screen.

9. Write a programme that swaps 2 numbers within a function using pointers & returns the sum. Print out the numbers before & after the function as well as the sum.

Development tools

A debugger or debugging tool is a computer program that is used to test & debug other programs.

We will run a program using a debugger to go through it step by step to observe the fetch decode execute cycle & look at register values after each step.

With your program sum.c, ensure you have a compiled program called sum.

GDB debugger

Debuggers are handy to step through a program and find the errors.

The debugger runs the program in a controlled environment, specifying runtime parameters. It allows you to:

- Stop the program at any point within the program
- Examine data elements, such as memory locations or registers
- Change elements in the program while it is running to help bug removal

When the program has been loaded into memory, you can run it from inside gdb:

```
> gdb ./sum
(gdb) run
```

GDB COMMANDS

- break - Set a breakpoint
- watch - Set a watchpoint to stop execution when a variable reaches the specific value
- info - obs…ents, such as registers, the stack, memory
- x - exa…tion
- print - Dis…s
- run - Sta…
- list - Lis…s or lines
- next - Ste…ction in the program
- step - Ste…ction in the program
- cont - Continue executing the program from the stopped point
- until - Run the program until it reaches the specified source code line (or greater)

Other commands:

https://www.gnu.org/software/gdb/documentation/

Delete the executable sum (not sum.c)

Now compile sum.c using debugging information to create the executable file test.

> *gcc –gstabs –o sum sum.c*

Open it in debugging mode with gdb.

> gdb sum

Tell the debugger where to pause - breakpoints “pause” execution.

Breakpoints are places in the program code you want the debugger to stop running the program and let you look around. Some options for breakpoints:

- A label
- A line number in the source code
- A data value when it reaches a specific value
- A function after it is performed a specific number of times

For us, a good place to start is at the start of the instruction code (after initialization) and watch things happen step-by-step:

(gdb) break main

Run the program, until it reaches main

(gdb) run

Then, instead of running the program, “next” takes you to the next instruction

(gdb) next

We can print out what is stored in the registers at different times (look at GDB commands) to see the flow of data in registers. We can look at Intel register values, before, after & between instructions.

Data Command	Description
info registers	Display the values of all registers
print	Display the value of a specific register or variable from the
x	e contents of a specific memory location

Look at register va

(gdb) info regis

Debug symbols typ nly the name of a function or global variable, but also the name of the source code file in which the symbol occurs, as well as the line number at which it is defined.

$ prefix is for immediates (constants), and the % prefix is for registers - notice the names of the registers? This shows that we are running IA32. You should recognize the general purpose registers (start with “e”) do you see any non-general registers?

Try stepping through the program, checking register values

1. Keep repeating “next”(or press up and enter) or try “step”
2. Try stepping through the program, checking register values

When bored, type “cont” or “run”, or Ctrl-Z at any point to exit debugger.

Repeat the process, putting a break at main+1, type “next” & repeat this or try “step”

The print command can also be used to display individual register values. Including a modifier can modify the output format of the print command:

- print/d to display the value in decimal
- print/t to display the value in binary
- print/x to display the value in hexadecimal

Try:

(gdb) print/x $ebx

(gdb) print/x $edx

(gdb) print/x $ecx

The “x” command is used to display the values of specific memory locations:

“x/nyz” - “n” is the number of fields to display

- “y” is the format of the output, and can be

- c for cha
- d for dec
- x for hex
- “z” is the [illegible] be displayed:
- b for byt
- h for 16-[illegible])
- w for 32-bit word

Example: use the x command to display the memory locations at the output label

(gdb) x/42cb &output

We will be looking at the registers again in a later practical.

10. Take a look at the following C programs in /Desktop/booksrc. Try to understand what they are trying to do:

notetaker.c

notesearch.c

exploit_notesearch.c

overflow_example.c

auth_overflow.c

fmt_vuln.c

Martin Read

2019