

CS330

C: Part II

Spring 2022

Lab 3

Today's Agenda

| 2

- char
- #define
- Typedef
- Struct
- More printf examples
- More on operands
- Switch
- While and Do-While
- Functions revisited
- User input via Scanf()
- GDB Basics

Odds and ends

- In Terminal: CTRL-C will terminate the program
- In C, 0 is False, anything non-zero evaluates to True
- In C, comments are the same as in Java

```
3  // single line comment
4
5  /*
6  block comment
7  */
8
9  /*
10 description:  this function calculates the distance between two
    numbers
11 arguments:
12     x1 represents the x1-location in pixels
13     x2 represents the x2-location in pixels
14 returns: the euclidan distance between x1 and x2
15 */
16 float calcDistance(int x1, int x2){
17     // do stuff
18 }
```

char

- Represented via a single quote mark: ‘
 - E.g. `char x = 'a';`
 - Double quotes denote strings: `char myString[] = "my string";`
- Also represented by decimal, ASCII

```
4  int main() {  
5      char x = 'a';  
6      printf("%c %d\n", x, x);  
7  }
```



```
a 97
```

- Since C treats the characters as integers, we can use them in numerical expressions. For example:

```
char ch = 'c';  
ch++; /* ch becomes 'd' */
```

#define

- The `#define` directive is used to define a macro. In most cases, a macro is a name that represents a numerical value. To define such a macro, we write:

`#define <macro_name> <value>`

NOTE: there is no semicolon at the end

For example

```
#define NUM 100
```

```
#define TRUE 1
#define FALSE 0
```

```
1  #include<stdio.h>
2
3  #define NUM 100
4
5  int main(){
6      int a, b, c;
7      a = 20 - NUM;
8      b = 20 + NUM;
9      c = 3 * NUM;
10     printf("a is %d, b is %d, c is %d\n", a, b, c);
11
12     return 0;
13 }
```

```
a is -80, b is 120, c is 300
```

typedef

- Typedef is similar to define, but used to map a new data type name to an existing type:

`typedef <existing type> <new name>;`

This has the added benefit of providing type checking

```
12  typedef int Length;  
13  
14  int main(){  
15  
16      Length l = 10;  
17      printf("the length is %d\n", l);
```

```
the length is 10
```

struct

- Struct (structure) is a collection of one or more variables (e.g. data structure)

- To declare:

```
struct [structure tag] {  
    <member type> <member name>;  
    ...  
    <member type> <member name>;  
} [list of variables] ;
```

- Can combine typedef and struct
- Ordering members largest to smallest saves space
- See also Union

```
1  #include<stdio.h>  
2  
3  struct Coordinate{  
4      int x;  
5      int y;  
6      int z;  
7  };  
8  
9  int main(){  
10     struct Coordinate c;  
11     /* access structure elements using dot (.) notation */  
12     c.x = 1;  
13     c.y = 2;  
14     c.z = 3;  
15  
16     printf("x is %d, y is %d, z is %d\n", c.x, c.y, c.z);  
17  
18     /* if using pointers, the syntax is slightly different, using -> */  
19     struct Coordinate *c_ptr;  
20     c_ptr = &c;  
21     c_ptr->x = 10;  
22     printf("x is %d, y is %d, z is %d\n", c_ptr->x, c_ptr->y, c_ptr->z);  
23  
24     return 0;  
25 }
```

```
x is 1, y is 2, z is 3  
x is 10, y is 2, z is 3
```

printf examples

```
#include <stdio.h>
int main(void)
{
    int len;
    printf("%c\n", 'w');
    printf("%d\n", -100);
    printf("%f\n", 1.56);
    printf("%s\n", "some text");
    printf("%e\n", 100.25);
    printf("%g\n", 0.0000123);
    printf("%X\n", 15);
    printf("%o\n", 14);
    printf("test%n\n", &len);
    printf("%d%%\n", 100);
    return 0;
}
```

The program outputs:

w (the character constant must be enclosed in single quotes).

-100

1.560000

some text (the string literal must be enclosed in double quotes).

1.002500e+002 (equivalent to $1.0025 \times 10^2 = 1.0025 \times 100 = 100.25$).

1.23e-005 (because the exponent is less than -4, the number is displayed in scientific form).

F (the number 15 is equivalent to F in hex).

16 (the number 14 is equivalent to 16 in octal).

test (since four characters have been printed before %n is met, the value 4 is stored into len).

100% (to display the % character, we must write it twice).

printf Precision

```
#include <stdio.h>
int main(void)
{
    float a = 1.2365;
    printf("%f\n", a);
    printf("%.2f\n", a);
    printf("%.3f\n", a);
    printf("%.1f\n", a);
    return 0;
}
```

1.236500
1.24
1.237
1

++ and --

- Similar to Java

```
int a=25;
```

```
a++; /* is equal to a = a+1; */
```

```
a--; /* is equal to a = a-1; */
```

Can also use ++a, order matters:

```
43  int a = 5;
44  int b;
45  printf("a++ = %d\n", a++); // prints 5, afterwards a evals to 6
46  b = a++; // a starts off 6, is assigned to b, then incremented to 7
47  printf("1) a = %d, b = %d\n", a, b); // a is 7, b is 6
48  a = 5; // reset
49  b = ++a; // a starts of 5, is incremented to 6, then assigned to b
50  printf("2) a = %d, b = %d\n", a, b); // a is 6, b is 6
```

a++ = 5

1) a = 7, b = 6

2) a = 6, b = 6

Compound Assignment Operators

```
#include <stdio.h>
int main(void)
{
    int a = 4, b = 2;
    a += 6;
    a *= b+3;
    a -= b+8;
    a /= b;
    a %= b+1;
    printf("Num = %d\n", a);
    return 0;
}
```

switch statement

```
#include <stdio.h>
int main(void)
{
    int a;
    printf("Enter number: ");
    scanf("%d", &a);
    switch(a)
    {
        case 1:
            printf("One\n");
            break;
        case 2:
            printf("Two\n");
            break;
        default:
            printf("Other\n");
            break;
    }
    printf("End\n");
    return 0;
}
```

while loop

```
#include <stdio.h>
int main(void)
{
    int i = 10;
    while (i != 0)
    {
        printf("%d\n", i);
        i--;
    }
    return 0;
}
```

do-while loop

```
#include <stdio.h>
int main(void)
{
    int i = 1;
    do
    {
        printf("%d\n", i);
        i++;
    } while (i <= 10);
    return 0;
}
```

Difference between while and do-while ?
When to use for vs while ?

Functions

- Need to define before use, can use a declaration statement
- Pass by Value vs Pass by Reference

Pass By Value

```
1  #include <stdio.h>
2
3  void myFunc(int x){
4      printf("inside myFunc x = %d (%p) \n", x, &x);
5      x++;
6      printf("inside myFunc x is now %d (%p)\n", x, &x);
7      return;
8  }
9
10 int main(){
11     int x = 5;
12     printf("inside main, x is %d (%p)\n", x, &x);
13     myFunc(x);
14     printf("inside main, x is %d (%p)\n", x, &x);
15     return 0;
16 }
```

```
inside main, x is 5 (0x7ffd49630c2c)
inside myFunc x = 5 (0x7ffd49630c0c)
inside myFunc x is now 6 (0x7ffd49630c0c)
inside main, x is 5 (0x7ffd49630c2c)
```


Input/Output: Terminal Input via scanf()

Syntax:

```
int scanf(const char *restrict format, ...);  
int scanf(<conversion spec string>, <pointer arg>);
```

- The scanf() function is used to read data from stdin (*standard input stream*) and store that data in program variables.
- The scanf() function accepts a variable list of parameters. The first is a format string similar to that of printf(), followed by the memory addresses of the variables in which the input data will be stored.
- Typically, the format string contains only conversion specifiers. The conversion characters used in scanf() are the same as those used in printf().

Input/Output: Terminal Input via scanf() Example

Syntax:

```
int scanf(const char *restrict format, ...);  
int scanf(<conversion spec>, <pointer arg>);
```

```
48      /* take as terminal input */  
49      int n;  
50      printf("Please enter an integer:\n");  
51      //scanf("%d", &n)  
52      int returnCode;  
53      if ( (returnCode = scanf("%d", &n)) < 0){  
54          printf("there was an error on input\n");  
55      }
```

Using gdb to better understand your program

`gdb` is an extremely powerful tool for debugging programs, helping you identify any runtime or logical errors (e.g. segmentation faults, core dumps, array out-of-bounds exceptions).

To debug your programs using **`gdb`**, you will need to compile with the **`-g`** option. This will instruct the compiler to generate the executable with source level debug info.

```
gcc -g name.c -o name
```

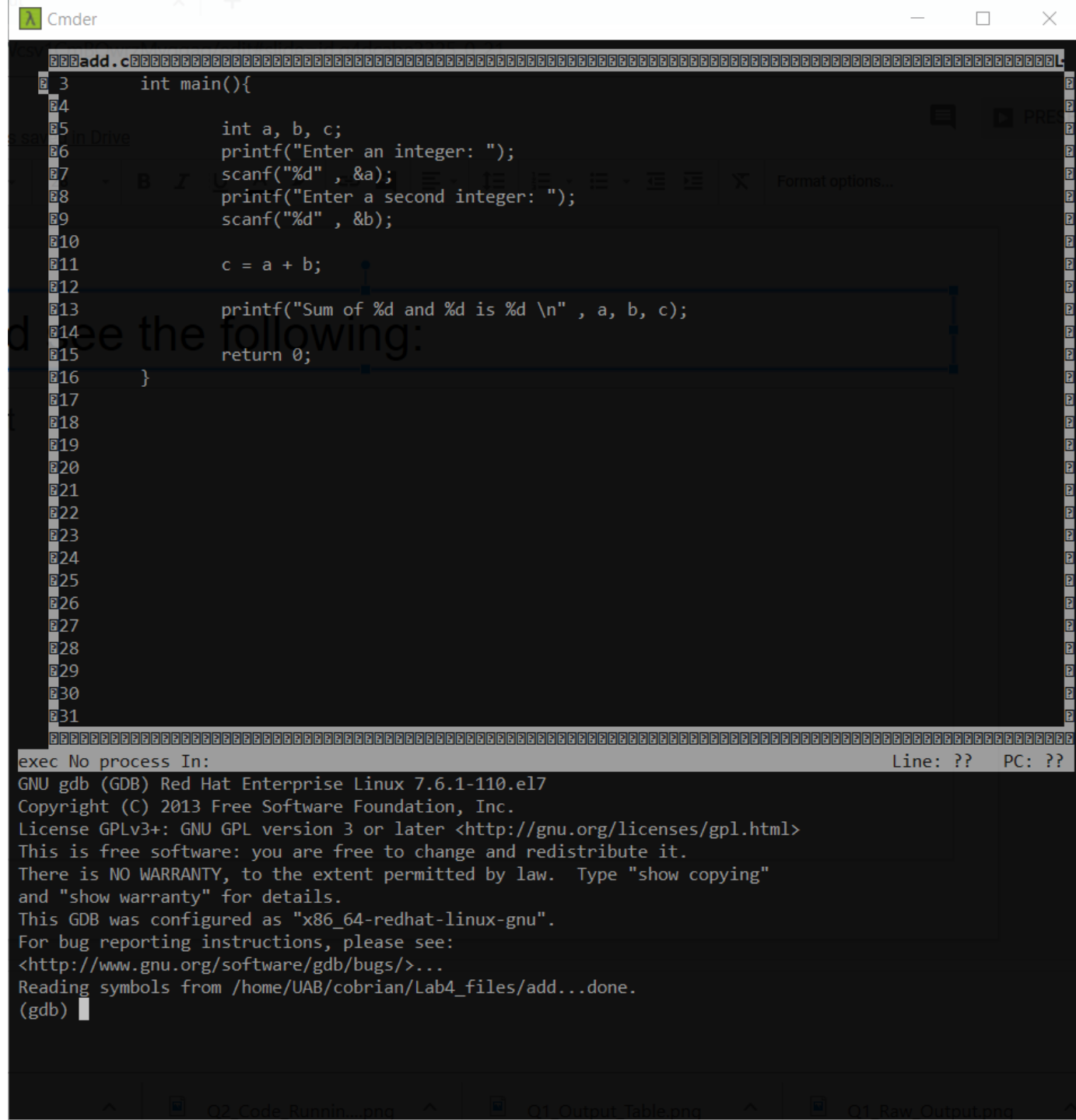
Now you can enter the **`gdb`** debugger with one of the following commands:

```
gdb name            gdb -tui name
```

The TUI version is recommended for new users and shows the source code and `gdb` command line in one window

You should see something similar to the picture, with **-tui**

The program here shows you how to take two numbers from user input and compute their sum!



The image shows a C code editor window with a dark theme. The code is a simple program to add two numbers. Below the code editor is a GDB terminal window showing the output of the program.

```
add.c
1  int main(){
2
3      int a, b, c;
4      printf("Enter an integer: ");
5      scanf("%d", &a);
6      printf("Enter a second integer: ");
7      scanf("%d", &b);
8
9      c = a + b;
10
11     printf("Sum of %d and %d is %d \n", a, b, c);
12     return 0;
13 }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

exec No process In: Line: ?? PC: ??

GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-110.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <<http://gnu.org/licenses/gpl.html>>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<<http://www.gnu.org/software/gdb/bugs/>>...
Reading symbols from /home/UAB/cobrian/Lab4_files/add...done.
(gdb)

GDB Basics

- To set a breakpoint:
 - (b)reak <line number>
 - or to break at the start of a function: (b)reak <name of function>
 - To remove a breakpoint: clear <line number>
- run
- Inspect variables
 - To display once: (p)rint <variable name>
 - To display at each step: display <variable name>
- Step through the code
 - (n)ext (don't dive into functions, step-over)
 - (s)tep (dive into functions, step-into)
 - (c)ontinue (continue running until the end or next breakpoint)
- RETURN (or ENTER) will repeat the last command entered
- (q)uit

Resources:

Beej's Quick Guide: <https://beej.us/guide/bggdb/>
<https://www.gnu.org/software/gdb/documentation/>
Manual: <http://sourceware.org/gdb/current/onlinedocs/gdb.pdf>