Recitation 3

Office Hours: 9:00 am-12:00 noon on Monday

A gdb Tutorial

What is GDB?

1. You can check what is happening during the execution of a program and how you can find out errors
2. Investigate improper behaviour of your program. For eg. if a segmentation fault occurs, gdb allows you to find where exactly it occurred
3. Sometimes finding logical errors in a program might be difficult by just looking at the program. Using gdb makes this much easier

How does GDB help?

Using GDB, you can:

1. You can pause the program using something called break points.
2. You can inspect the values of the variables once the program has been paused. Further, you can temporarily change the values of these variables and check its impact on the program.
3. GDB supports not only C, but a host of other applications like C++, Pascal etc.

How to use GDB: Some pointers

1. Compile and build programs with debugging symbols:

```
$ gcc -g main.c
```

The -g flag will generate the debug symbols of the program. These are necessary to debug any program with GDB.

You can know more about debug symbols and gcc symbol table at:

http://www.network-theory.co.uk/docs/gccintro/gccintro_90.html

How to use GDB: Some pointers

2. Run the program with GDB:

```
$ gdb a.out

Reading symbols from a.out...done.

(gdb)
```

As you can see above, GDB read debug symbols from a.out and GDB prompt appeared where we can execute GDB commands.

Some GDB commands

Commands	**Description**
r	Start running program until a breakpoint or end of program
b [function name]	Set a breakpoint at the beginning of a specific function
b N	Set a breakpoint at line number N of source file currently executing
b file.c:N	Set a breakpoint at line number N of file "file.c"
d N	Remove breakpoint number N
info break	List all breakpoints
c	Continues/Resumes running the program until the next breakpoint or end of program

Some GDB commands

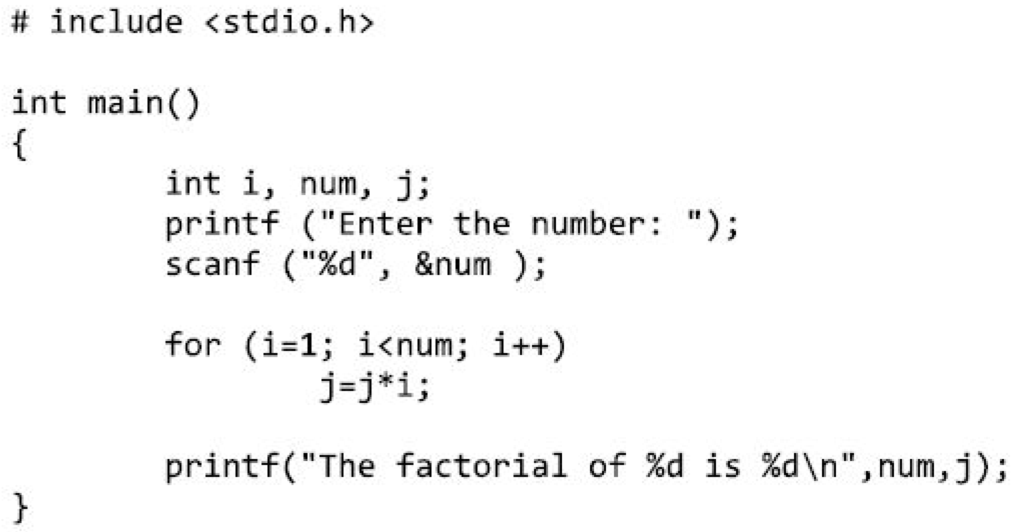
f	Runs until the current function is finished
s	Runs the next line of the program
s N	Runs the next N lines of program
n	Like s, but it does not step into functions
p variable	Prints the current value of the variable “variable”
set variable=val	Assign "val" value to the variable "variable"
q	Quit from gdb

Some more pointers

1. If you are confused about a command, type “help” with or without an argument. This gives a description of the command.

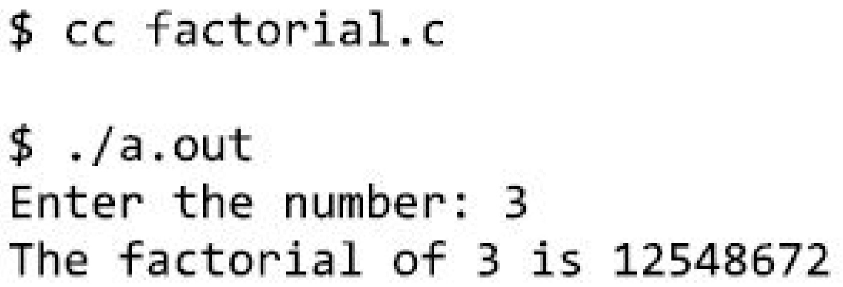
2. Don’t just run the program without stopping or breaking. A good idea is to step through the code one line at a time and figure out the root of the error.

An Example



```
# include <stdio.h>

int main()
{
        int i, num, j;
        printf ("Enter the number: ");
        scanf ("%d", &num );

        for (i=1; i<num; i++)
                j=j*i;

        printf("The factorial of %d is %d\n",num,j);
}
```

The Output



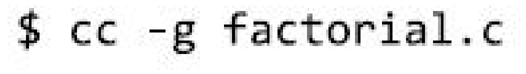
```
$ cc factorial.c

$ ./a.out
Enter the number: 3
The factorial of 3 is 12548672
```

This is not the output we were expecting!

Debugging the Program

Step 1:

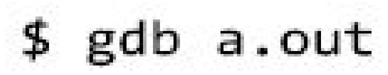
Compile your C program with the -g flag. This allows the compiler to collect the debugging information. It creates the debugging file a.out which we will use later



```
$ cc -g factorial.c
```

Step 2:

Launch the debugger:



```
$ gdb a.out
```

Step 3:

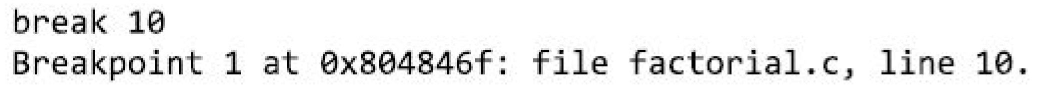
Placing breakpoints in the C program. The formats are as follows:

- break [file_name]:line_number
- break [file_name]:func_name

Why put breakpoints?

Places break point in the C program, where you suspect errors. While executing the program, the debugger will stop at the break point, and gives you the prompt to debug.

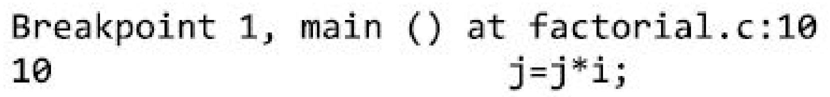
So before starting up the program, let us place the following break point in our program:



```
break 10
Breakpoint 1 at 0x804846f: file factorial.c, line 10.
```

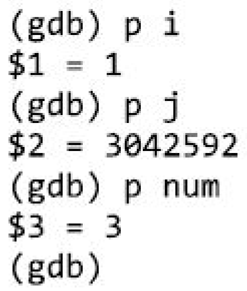
Step 4

Run the program using the ‘run’ command. Once you executed the C program, it would execute until the first break point, and give you the prompt for debugging.



```
Breakpoint 1, main () at factorial.c:10
10                          j=j*i;
```

Try printing the variables to check their values?



```
(gdb) p i
$1 = 1
(gdb) p j
$2 = 3042592
(gdb) p num
$3 = 3
(gdb)
```

Step 5

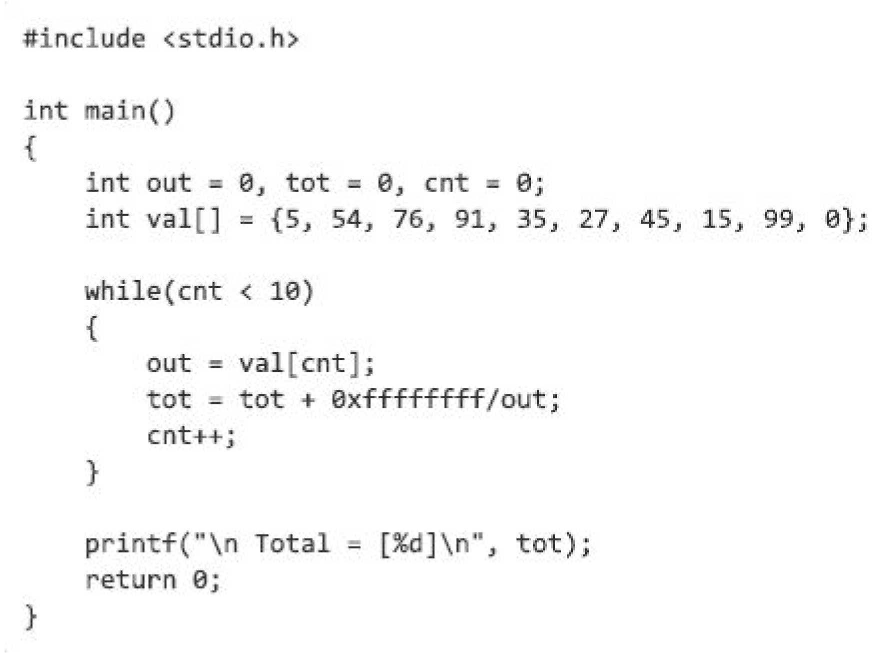
Clearly, we didn’t set j to an initial value. This is one of the reasons we got an error. We can set the value of j while debugging:

```
set variable j = 1
```

Reference for the first example

http://www.thegeekstuff.com/2010/03

Another Example



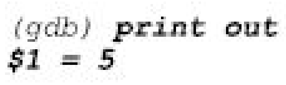
```c
#include <stdio.h>

int main()
{
    int out = 0, tot = 0, cnt = 0;
    int val[] = {5, 54, 76, 91, 35, 27, 45, 15, 99, 0};

    while(cnt < 10)
    {
        out = val[cnt];
        tot = tot + 0xffffffff/out;
        cnt++;
    }

    printf("\n Total = [%d]\n", tot);
    return 0;
}
```

```
$ ./gdb-test
Floating point exception (core dumped)
```

```
gdb ./gdb-test
```

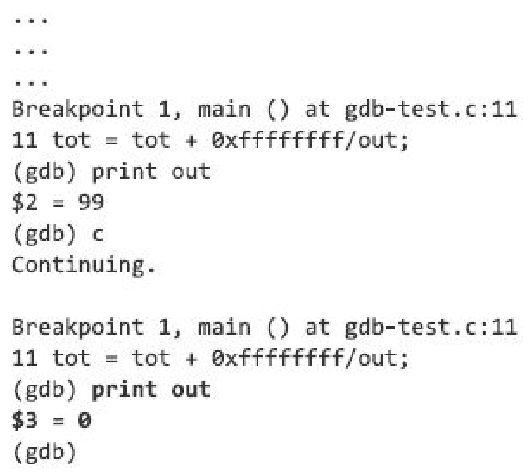
```
(gdb) break 11
```

```
run
```



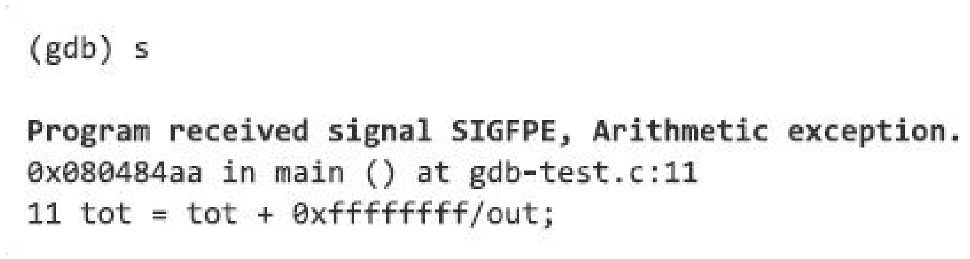
```
(gdb) print out
$1 = 5
```

Till now everything seems fine. Continue running the program using ‘c’.



```
...
...
...
Breakpoint 1, main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
(gdb) print out
$2 = 99
(gdb) c
Continuing.

Breakpoint 1, main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
(gdb) print out
$3 = 0
(gdb)
```

Confirming the error

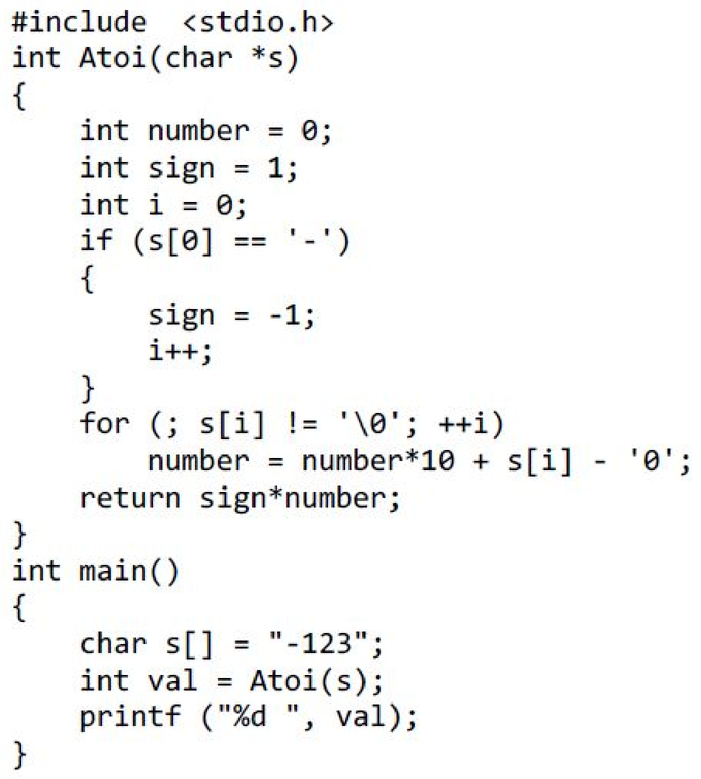


```
(gdb) s

Program received signal SIGFPE, Arithmetic exception.
0x080484aa in main () at gdb-test.c:11
11 tot = tot + 0xffffffff/out;
```

Reference for second example

http://www.unknownroad.com/rtfm/gdbtut/gdbuse.html

Last Week’s Solution: Your own atoi()



```
#include  <stdio.h>
int Atoi(char *s)
{
    int number = 0;
    int sign = 1;
    int i = 0;
    if (s[0] == '-')
    {
        sign = -1;
        i++;
    }
    for (; s[i] != '\0'; ++i)
        number = number*10 + s[i] - '0';
    return sign*number;
}
int main()
{
    char s[] = "-123";
    int val = Atoi(s);
    printf ("%d ", val);
}
```

This weeks questions

0. Consider:

```
int i = 5;
int *ip = &i;
```

.. what is ip? What is its value?

1. Write some code that declares two arrays of size 10 that are string literals.
   Make a pointer to one of the arrays, cast it to be an int pointer, and print out its value.
   Make a new integer, set it equal to the value of your int pointer, then make a pointer to that integer, cast it to be a char pointer, and print out 8 chars.
   What happened? Why?

2. Write some code that declares two arrays of size 10 that are string literals.

Create a pointer that points to the beginning of the first array, then in a loop, increment the pointer and print out the char it points to, out to index 20.
What happened? Why?