For GDB installation, follow this video link:

How to Download and Install C Cpp Toolset (gcc g++ gdb) in Windows 11 using mingw…

GDB Tutorial

GDB stands for GNU Project Debugger and is a powerful debugging tool for C (along with other languages like C++). It helps you to poke around inside your C programs while they are executing and also allows you to see what exactly happens when your program crashes. GDB operates on executable files which are binary files produced by the compilation process.

For demo purposes, the example below is executed on a Linux machine with the below specs.

uname -a

```
sree@ubuntu:~$ uname -a
Linux ubuntu 4.10.0-19-generic #21-Ubuntu SMP Thu Apr 6 17:04:57 UTC 2017 x86_64
 x86_64 x86_64 GNU/Linux
```

Let's learn by doing: –

Start GDB

Go to your Linux command prompt and type "gdb".

gdb

```
sree@ubuntu:~$ gdb
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

Gdb open prompt lets you know that it is ready for commands. To exit out of gdb, type quit or q.

Compile the code

Below is a program that shows undefined behavior when compiled using C99.

```
sree@ubuntu:~/debugging$ cat test.c
//sample program to show undefined behaviour
//Author : Sreeraj R
#include<stdio.h>

int main()
{
    int x;
    int a = x;
    int b = x;
    int c = a + b;
    printf("%d\n", c);
    return 0;
}
```

Note: If an object that has automatic storage duration is not initialized explicitly, its value is indeterminate, where the indeterminate value is either an unspecified value or a trap representation.

Now compile the code. (here test.c). **g flag** means you can see the proper names of variables and functions in your stack frames, get line numbers and see the source as you step around in the executable. **-std=C99 flag** implies use standard C99 to compile the code. **-o flag** writes the build output to an output file.

gcc -std=c99 -g -o test test.C

Run GDB with the generated executable

Type the following command to start GDB with the compiled executable.
gdb ./test

```
sree@ubuntu:~/debugging$ gdb ./test
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./test...done.
(gdb) r
Starting program: /home/sree/debugging/test
-15808
[Inferior 1 (process 4989) exited normally]
(gdb)
```

Useful GDB commands:

Here are a few useful commands to get started with GDB.

Command	Description
run or r	Executes the program from start to end.
break or b	Sets a breakpoint on a particular line.

disable	Disables a breakpoint
enable	Enables a disabled breakpoint.
next or n	Executes the next line of code without diving into functions.
step	Goes to the next instruction, diving into the function.
list or l	Displays the code.
print or p	Displays the value of a variable.
quit or q	Exits out of GDB.

clear	Clears all breakpoints.
continue	Continues normal execution

Display the code

Now, type “l” at gdb prompt to display the code.

```
(gdb) l
1       //sample program to show undefined behaviour
2       //Author : Sreeraj R
3       #include<stdio.h>
4
5       int main()
6       {
7           int x;
8           int a = x;
9           int b = x;
10          int c = a + b;
(gdb)
```

Set a breakpoint

Let’s introduce a break point, say line 5.

```
(gdb) b 5
Breakpoint 1 at 0x5555555546a8: file test.c, line 5.
(gdb)
```

If you want to put breakpoint at different lines, you can type “b *line_number*“.By default “list or l” display only first 10 lines.

View breakpoints

In order to see the breakpoints, type “info b”.

```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000000006a8 in main at test.c:5
(gdb)
```

Disable a breakpoint

Having done the above, let’s say you changed your mind and you want to revert. Type “disable b”.

```
(gdb) disable b
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep n   0x00000000000006a8 in main at test.c:5
(gdb)
```

Re-enable a disabled breakpoint

As marked in the blue circle, Enb becomes n for disabled. 9. To re-enable the recent disabled breakpoint. Type “enable b”.

```
(gdb) enable b
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000000006a8 in main at test.c:5
(gdb)
```

Run the code

Run the code by typing “run or r”.If you haven’t set any breakpoints, the run command will simply execute the full program.

```
(gdb) r
Starting program: /home/sree/debugging/test

Breakpoint 1, main () at test.c:8
8           int a = x;
(gdb)
```

Print variable values

To see the value of variable, type “print *variable_name* or p *variable_name*“.

Print variable values

The above shows the values stored at x at time of execution.

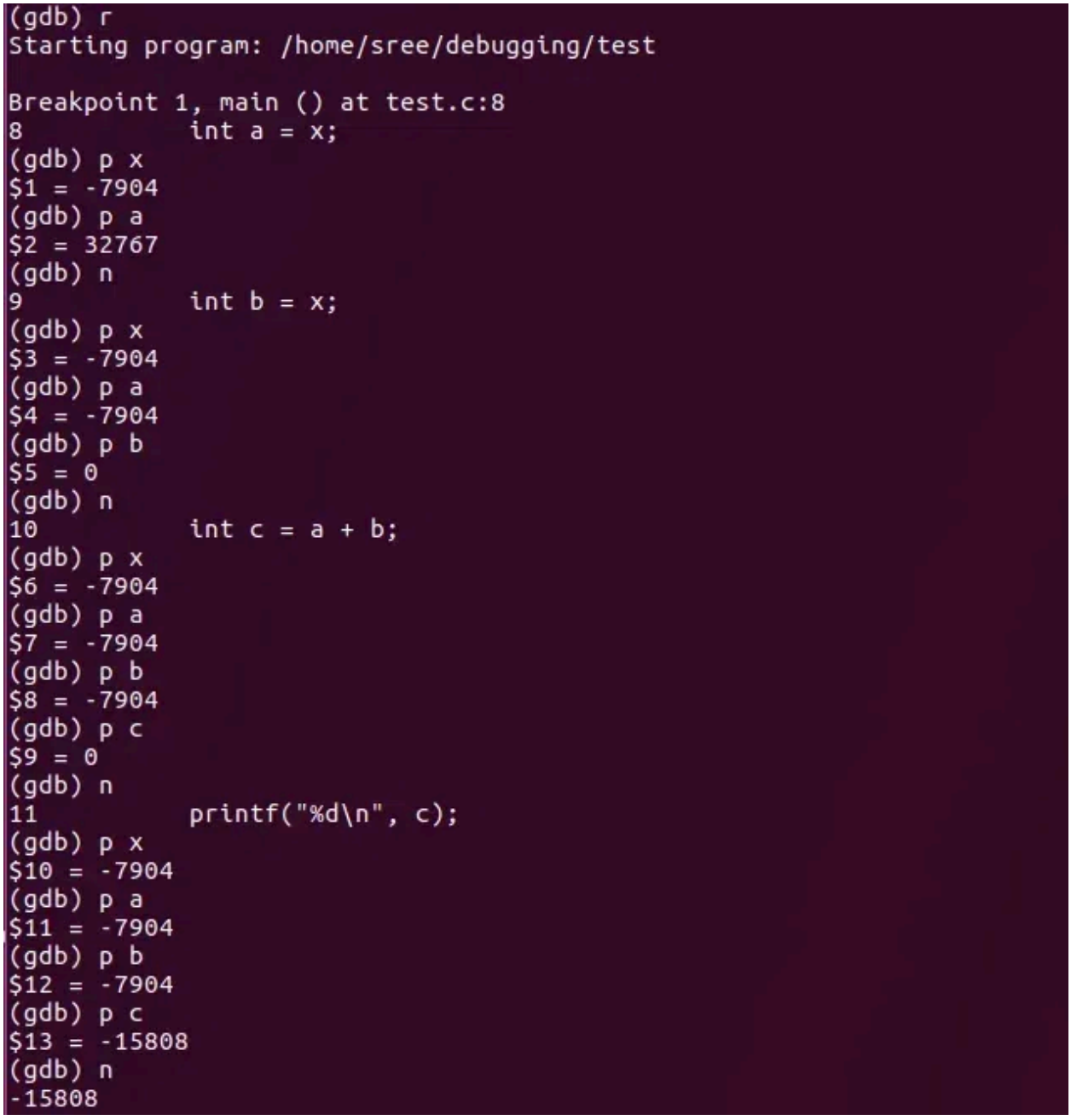
Change variable values

To change the value of a variable in gdb and continue execution with the changed value, type “set *variable_name*“.

Debugging output

Below screenshot shows the values of variables from which it’s quite understandable the reason why we got a garbage value as output. At every execution of **./test** we will be receiving a different output.

Exercise: Try using set x = 0 in gdb at first run and see the output of c.



Debugging output

GDB offers many more ways to debug and understand your code like examining stack, memory, threads, manipulating the program, etc. I hope the above example helps you get started with gdb.

Printing the stack

Using '**backtrace**' we can print the current function and all the functionality that were called leading up to it.

```
(gdb) backtrace
#0  sort (v=0x7fffffffde10, size=9) at merge.cc:10
#1  0x0000000000400815 in main () at merge.cc:20
```

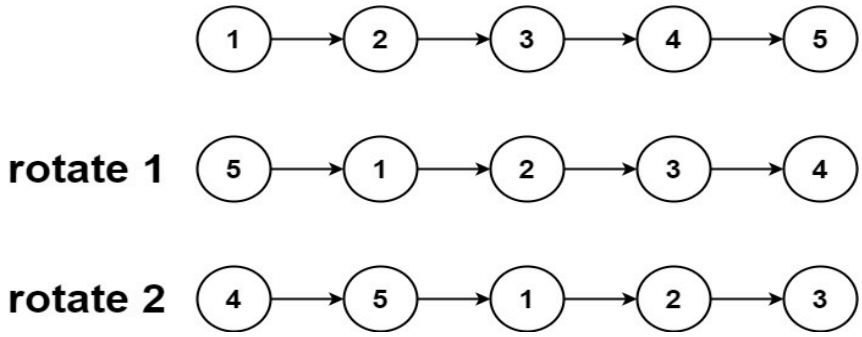
Using '**info frame**' we can print the entire stack trace, local variables and arguments in the current frame etc.

Evaluative Question

1.[5 points] Given the head of a linked list, rotate the list to the right by k places. Before rotation, construct the linked list using the `LinkedList` class, utilizing helper functions like `**add(node)**` to build the list. Then, implement a function to perform the rotation.

Input: head = [1,2,3,4,5], k = 2

Output: [4,5,1,2,3]



Practice Questions

1. In the quaint village of Meadowvale, three friends, Sarah, Ben, and Mia, stumbled upon a chest filled with enchanted stones, each inscribed with a number. Intrigued, they decided to embark on a mathematical quest. Their challenge was to calculate the median of every window of k stones, moving from left to right, and print the sequence of medians

obtained as the window slides. Can Sarah, Ben, and Mia uncover the secrets hidden within the shifting windows of stones?

Write a cpp code for the desired output.

Example

Input:

8 3

2 4 3 5 8 1 2 1

Output:

3 4 5 5 2 1