Debugging using gdb

Debugging

- According to the Software Engineering Institute and the IEEE, every significant piece of software will initially contain defects.
- These mistakes lead to programs and libraries that don’t perform as required.
- Bug tracking, identification, and removal can consume a large amount of a programmer’s time during software development.

Debugging

Types of Errors

1. Specification errors: If a program is incorrectly specified, it will inevitably fail to perform as required.

-You can detect and remove many specification errors by reviewing the requirements.

Cont…

2. Design errors: Programs of any size need to be designed before they're created.

-Take time to think about how to construct the program, what data structures you'll need, and how they will be used.

3. Coding errors.

General Debugging Techniques

- Testing: Finding out what defects or bugs exist
- Stabilization: Making the bugs re-producable
- Localization: Identifying the line(s) of code responsible
- Correction: Fixing the code
- Verification: Making sure the fix works

Contd…

- If code results in incorrect output the follow:
  - Code Inspection: Code inspection is also a term for the more formal process of a group of developers tracing through a few hundred lines of code in detail.
  - Can use the compiler to check errors

2. Instrumentation:

-Adding code to a program for the purpose of collecting more information about the behavior of the program as it runs.

-e.g. use printf() to print out the values of variables at different stages in a program’s execution.

GNU Debugger gdb

GDB allows you to run the program up to a certain point, then stop and print out the values of certain variables at that point, or step through the program one line at a time and print out the values of each variable after executing each line.

Debugger

COMMANDS

breakpoint	b
list	l
frame	f
next	n
step	s
backtrace	b
print	p
info	i
watch	

Debugging

```
nano  swap.cpp
 g++ swap.cpp
```

instruct compiler to include debugging information

```
g++ -g swap.cpp -o swap
```

```
gdb swap
```

Debugging

1. set breakpoint

```
b  main
b swap
```

2. start debugging

```
start
```

Debugging

3. to see source code

```
l      {to list the code
```

4. if you want to know on which command/frame you are now, use frame command: f

```
f
#0
```

Debugging

5. go to next command:

n {to go to next line

6. to step in side the function swap

s {step next

in code values of x and y are passed by value but required is pass by reference

So this is a bug

Debugging

7. now go in side the function by pressing n   {next

8. to see the entire transition and path use the backtrace command

```
bt
```

```
#0
#1
```

Debugging

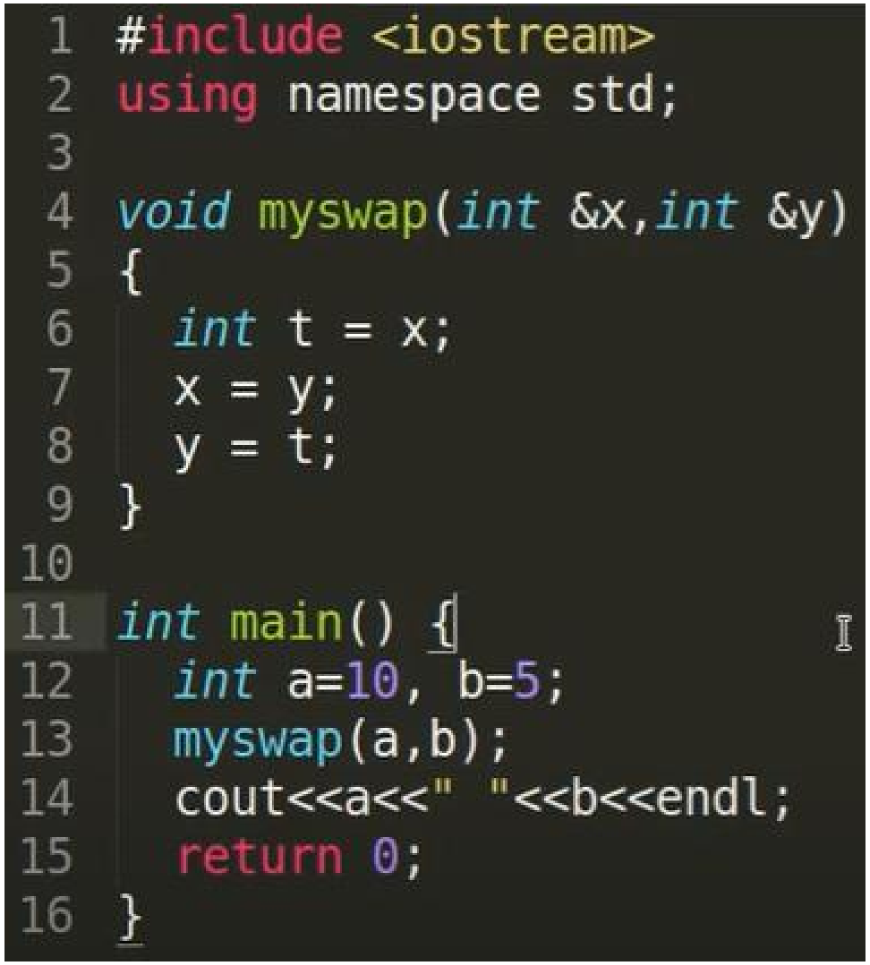
9. if you want to go back to main. use frame command with frame number.

f 1

f 0

press n for next

Corrected



```cpp
#include <iostream>
using namespace std;

void myswap(int &x,int &y)
{
  int t = x;
  x = y;
  y = t;
}

int main() {
  int a=10, b=5;
  myswap(a,b);
  cout<<a<<" "<<b<<endl;
  return 0;
}
```

Debugging with gdb

Quit the gdb first

```
q
```

Compile again

```
g++ -g swap.cpp -o swap
```

```
 gdb  swap
b  main      {set breakpoint
start        {start program
```

Debugging with gdb

10. watch variablename { to set a watch point on the variable a so that gdb notify when the value of a changes }

Now set Watch point on a and b variables. So that when ever values of a and b are changed, you will be notified

```
watch a
watch b
```

11. display a

12. c to continue