# Compiling and Debugging C Programs

gcc & gdb

# What Is **gcc ?**

- **gcc** is an open compilation tool similar to cc.
- **gcc** stands for GNU Compiler Collections
- **gcc** supports C, C++, Pascal and Fortran77.

# Basics of gcc

- To compile a C Program
  - gcc filename.c
  - It will generate an object code – a.out (default object file name)
- To run C object code
  - ./a.out

# Basics of gcc

- To generate user defined object code
  - gcc filename.c –o filename.out
- To run user defined object code
  - ./filename.out
- To include math library (math.h)
  - gcc filename.c –o filename.out -lm

# Using gdb

The GNU C/C++ debugger

# gdb, the GNU Debugger

- A debugger is a application which allows you to look inside your own program as it executes.
- Debugging is used to locate and correct logic errors (eg.segmentation fault) in a C program after all syntactic errors have been eliminated

# gdb

- The key features of gdb are that it allows you to start and stop your program at any line of your program.
- Whenever the program is stopped, you can look at the contents of any variable (identifier) in your program.

# The debugging process using gdb.

# gdb

1. Compile your Program
2. Invoking gdb
3. Execution State
4. Breakpoints
5. Running your program
6. Examining Data
7. Debugging
8. Quit

# Compile your Program

- Use gcc to eliminate all syntatic errors in your program. It should compile before gdb is used. After your program already compiles, run the compiler again.
- This time include the –g gdb option:
  - gcc -ggdb your_program.c
  - gcc -ggdb your_program.c –lm
    - /* if you need the math library */

# Invoking gdb

- gdb a.out /* or whatever your executable file is called */
  - Or
- gdb
  - Some copyright information will print and you will get the **gdb** prompt:
- (gdb)

To open an executable

- (gdb) file a.out
  - Some comments – Reading symbols from a.out

# Execution State

- There are two possible executing states for your program:
  - running
    - While your program is running, it executes normally
  - stopped
    - While stopped, you can enter gdb commands. Your program begins in the stopped state, waiting for the first gdb command.

# Breakpoints

- Breakpoints are places in your program where execution will stop and gdb will await your command.
  - (gdb) break 37
    - This sets a breakpoint at line 37 of your source code. Any valid line number can be used.
  - (gdb) break swap
    - This sets a breakpoint at the beginning of the function swap(). Any valid function name can be used.

# Clear Breakpoints

- clear 37
- clear swap
  - These commands clear (delete) breakpoints. Their syntax is the same as the break command.

# Watchpoints

- The argument to the watch command is an expression that is evaluated.
- To set a watch point on a non-global variable, you must have set a breakpoint that will stop your program when the variable is in scope.
- You set the watchpoint after the program breaks.
  - (gdb) watch x

```
Hardware watchpoint 4: x
(gdb) c
Continuing.
Hardware watchpoint 4: x

Old value = -1073743192
New value = 11
main (argc=1, argv=0xbffffaf4) at test.c:10
10      return 0;
```

# Read Watchpoints

- Use the rwatch command. Usage is identical to the watch command.
  - (gdb) rwatch y

# read/write Watchpoints

- Use the awatch command. Usage is identical to the watch command.
  - (gdb) awatch y

# disable Watchpoints

- Active watchpoints show up the breakpoint list.
- Use the info breakpoints command to get this list.
- Then use the disable command to turn off a watchpoint, just like disabling a breakpoint.
  - gdb) info breakpoints

```
Num     Type           Disp  Enb Address    What
1       breakpoint     keep y   0x080483c6 in main at test.c:5
                              breakpoint already hit 1 time
4       hw watchpoint  keep y   x
                              breakpoint already hit 1 time
(gdb) disable 4
```

# Running your program

- First, be sure that breakpoints are set where you want them.
  - Once your program is executing you cannot gracefully stop it to insert breakpoints -- you have to wait for it to stop (or crash) or you can use C-c (Control c) to stop execution of your program and return to gdb.
- First, be sure that breakpoints are set where you want them.

# Running your program (contd )

- After breakpoints are set, you can use the following commands:
  - (gdb) run
    - This is used to begin execution at the beginning. Normally it is the first command issued after breakpoints are set.
  - (gdb) kill
    - This is the opposite of run. It immediately halts execution of your (stopped) program.

# Running your program (contd )

- (gdb) continue/c
  - When your program has stopped at a breakpoint, this command is used to continue program execution. Execution will continue until it stops at the next breakpoint or until your program quits (or crashes).
- (gdb) next/n
  - The same as continue except that only the next line of code will be executed and then gdb will await further commands. All commands in the line will be executed, including function calls.

# Running your program (contd )

- (gdb) step/s
  - This is the same as next except that if the line contains a function call, execution will stop at the first line of that function. This command is useful if you want to observe what is happening both in the main program you are debugging and in the functions it calls.

# Running your program (contd )

- (gdb) stepi/si
- (gdb) nexti/ni
  - These commands are the same as next and step except that one instruction is executed, instead of one line.

# Examining Data

- While execution is stopped at a breakpoint, gdb allows you to examine and change any value in any memory location.
  - (gdb) print ans/p ans
    - This command prints the contents of memory associated with the C identifier (variable) "ans". Any valid identifier can be used.

# Debugging

- A useful debugging command is
  - (gdb) backtrace
    - This command tells you what happened just before you typed the backtrace command. That can sometimes be helpful if your program crashes while running within gdb.

# Debugging

- Another way to tell where you are in a program is to use the command
  - (gdb) list
  - (gdb) list 15
  - (gdb) list swap
    - This command lists the 10 lines around the "current" line. When a number is used, the 10 lines listed are those around the line number entered. When a function name is used, the 10 lines are those surrounding the function definition. Any valid line number or function name can be used.

# Quit

- You can quit gdb at any time by using quit command
  - (gdb) quit

# Help

- The gdb command help can be used to obtain help about gdb, any class of gdb commands, or any specific gdb command. You can use the help command any time your program is stopped -- any time you see the (gdb) prompt.
- E.g:
  - (gdb) help run
  - (gdb) help data - To know more about Examining data
  - (gdb) help running – To know more about running a program
  - (gdb) help breakpoints

# Miscellaneous Commands

- (gdb) shell – Temporarily quits the gdb and opens a shell prompt

# Miscellaneous Commands

- (gdb) x - To examine memory
  - If we have char *s = "Hello World\n", some uses of the x command could be:
    - Examine the variable as a string:
      - **(gdb) x/s s**
      - **0x8048434 <_IO_stdin_used+4>:  "Hello World\n"**
    - Examine the variable as a character:
      - **(gdb) x/c s**
      - **0x8048434 <_IO_stdin_used+4>: 72 'H'**
    - Examine the variable as 4 characters:
      - **(gdb) x/4c s**
      - **0x8048434 <_IO_stdin_used+4>: 72 'H' 101 'e' 108 'l' 108 'l'**

# Miscellaneous Commands

- (gdb) info registers - To see the contents in processor registers
  - (gdb) info registers

```
eax            0x40123460       1074934880
ecx            0x1              1
edx            0x80483c0        134513600
ebx            0x40124bf4       1074940916
esp            0xbffffa74       0xbffffa74
ebp            0xbffffa8c       0xbffffa8c
esi            0x400165e4       1073833444
```

# Miscellaneous Commands

- (gdb) disassemble- To see the assembly code of running program
  - (gdb) disassemble main

```
Dump of assembler code for function main:
0x80483c0 <main>:      push   %ebp
0x80483c1 <main+1>:    mov    %esp,%ebp
0x80483c3 <main+3>:    sub    $0x18,%esp
0x80483c6 <main+6>:    movl   $0x0,0xfffffffc(%ebp)
0x80483cd <main+13>:   mov    0xfffffffc(%ebp),%eax
0x80483d0 <main+16>:   movb   $0x7,(%eax)
0x80483d3 <main+19>:   xor    %eax,%eax
0x80483d5 <main+21>:   jmp    0x80483d7 <main+23>
0x80483d7 <main+23>:   leave
0x80483d8 <main+24>:   ret
End of assembler dump.
```

# Thank You