## Agenda

## Using the Debugger
### Introduction

- You learned that there are two types of errors, compilation errors and logic errors, and you learned how to eliminate compilation errors from your code. Logic errors do not prevent a program from compiling successfully, but they can cause the program to produce erroneous results when it runs.
- The debugger is one of the most important program development tools. Many IDEs provide their own debuggers similar to the one included in GNU or provide a graphical user interface to GNU's debugger.

## Using the Debugger
### Breakpoints and the run, start, stop and print Commands

- We begin by investigating breakpoints, which are markers that can be set at any executable line of code. When program execution reaches a breakpoint, execution pauses, allowing you to examine the values of variables to help determine whether a logic error exists.
- Note that attempting to set a breakpoint at a line of code that is not executable will actually set the breakpoint at the next executable line of code in that function.

## Using the Debugger
### Breakpoints and the run, start, stop and print Commands

- To illustrate the features of the debugger, we use the program which finds the maximum of three integers.

```
// Finding the maximum of three integers
#include <stdio.h>

 int maximum( int x, int y, int z ); // function prototype

// function main begins program execution
int main( void )
{
   int number1, number2, number3;
   printf( "%s:", "Enter three integers:" );
   scanf( "%d%d%d", &number1, &number2, &number3 );
   printf( "Maximum is:%d\n", maximum( number1, number2, number3 ) );

   return 0;
} // end main
int maximum( int number1, int number2, int number3 ){
   int max = x;
   if( y > max ){
      max = y;
   }
   if( z > max ){
      max = z;
   }
   return max;
}
```

## Using the Debugger

### Breakpoints and the run, start, stop and print Commands

- **Compiling the program for debugging:** To use the debugger, you must compile your program with the -g option, which generates additional information that the debugger needs to help you debug your programs. To do so, type

```
gcc -g SourceCode.c
```

- **Starting the debugger:** Type gdb ./a.out. The gdb command starts the debugger and displays the gdb prompt at which you can enter commands.
- **Running a program in the debugger:** Run the program through the debugger by typing run. If you do not set any breakpoints before running your program in the debugger, the program will run to completion.

## Using the Debugger

### Breakpoints and the run, start, stop and print Commands

- **Inserting breakpoints using the GNU debugger:** The break command inserts a breakpoint at the line number specified as its argument. You can set as many breakpoints as necessary. When the program runs, it suspends executing at any line that contains a breakpoint and the debugger enters break mode.
- If you do not have a numbered listing for your code, you can use the list command to output your code with line numbers.
- **Running the program and beginning the debugging process:** The debugger enters break mode when executing reaches the breakpoint. At this point, the debugger notifies you that a breakpoint has been reached and displays the source code at that line, which will be the next statement to execute.
- **Using the continue command to resume execution:** The continue command causes the program to continue running until the next breakpoint is reached. The debugger notifies you when execution reaches the second breakpoint.
- **Examining a variable's value:** The print command allows you to peek inside the computer at the value of one of your variables.
- **Using convenience variables:** Convenience variables are temporary variables created by the debugger that are names using a dollar sign followed by an integer. Convenience variables can be used to perform arithmetic and evaluate boolean expressions.

## Using the Debugger

### Breakpoints and the run, start, stop and print Commands

- **Continuing program execution:** Type continue to continue the program's execution. The debugger encounters no additional breakpoints, so it continues executing and eventually terminates.
- **Removing a breakpoint:** You can display a list of all of the breakpoints in the program by typing info break. To remove a breakpoint, type delete, followed by a space and the number of the breakpoint to remove.
- **Using the quit command:** Use the quit command to end the debugging session. This command causes the debugger to terminate.

## Using the Debugger

### Breakpoints and the run, start, stop and print Commands

- Typing break, then a function name will cause the debugger to enter the break mode whenever that function is called.
- If you have any question about the debugger or any of its commands, type help or help followed by the command name for more information.

## Using the Debugger
### print and set Commands

- The print command can be used to examine the value of more complex expressions. The set command allows you assign new values to variables.
- **Starting debugging:** Type gdb ./a.out to start the GNU debugger.
- **Inserting a breakpoint:** Set a breakpoint at line “printf( ”Maximum is:%d\n”, maximum( number1, number2, number3 ) );” in the source code by typing break 12.
- **Running the program and reaching a breakpoint:** Type run to begin the debugging process. This will cause main to execute until the breakpoint at line 12 is reached.
- **Evaluating arithmetic and boolean expressions:** You can use print to evaluate arithmetic and boolean expressions. Type print number1 - 2. This expression retunrs the value 20, but does not actually change the value of number1. Type print number1 == 20. Expression containing the == symbol return 0 if the statement is false and 1 if the statement is true.
- **Modifying values:** You can change the values of variables during the program’s executing in the debugger. This can be value for experimenting different values and for locating logic errors. You can use the debugger’s set command to change a variable’s value. Type set number1 = 90 to change the value of number1, then type print number1 to display its new value.
- **Viewing the program result:** Type continue to continue program execution.
- **Using the quit command:** Use the quit command to end the debugging session.

## Using the Debugger
### Controlling Execution Using the step, finish and next Commands

- Sometimes you’ll need to execute a program line by line to find and fix errors. Walking through a portion of your program this way can help you verify that a function’s code executes correctly. The commands in this section allow you to execute a function line by line, execute all the statements of a function at once or execute only the remaining statements of a function.
- **Starting the debugger:** Start the debugger by typing gdb ./a.out.
- **Setting a breakpoint:** Set a breakpoint at line “printf( ”Maximum is:%d\n”, maximum( number1, number2, number3 ) );” in the source code by typing break 12.
- **Running the program:** Run the program by typing run, then enter numbers at the prompt. The debugger then indicates that the breakpoint has been reached and displays the code at line “printf( ”Maximum is:%d\n”, maximum( number1, number2, number3 ) );”.
- **Using the step command:** The step command executes the next statement in the program. If the next statement to execute is a function call, control transfers to the called function.
- **Using the finish command:** After you’ve stepped into the member function, type finish. This command executes the remaining statements in the function and returns control to the place where the function was called. The finish command executes the remaining statements in function, then pauses at line “printf( ”Maximum is:%d\n”, maximum( number1, number2, number3 ) );”.
- **Using the continue command to continue execution:** Enter the continue command to continue execution until the program terminates.

## Using the Debugger
### Controlling Execution Using the step, finish and next Commands

- **Running the program again:** Breakpoints persist until the end of the debugging session in which they are set. Type the run to execute the program and enter numbers at the prompt.
- **Using the next command:** The next command behaves like the step command, except when the next statement to excute contains a function call. In that case, the called function executes in its entirety and the program advances to the next executable line after the function call.

## Using the Debugger
### watch Command

- The watch command tells the debugger to watch a data member. When that data member is about to change, the debugger will notify you.
- **Starting the debugger:** Start the debugger by typing gdb ./a.out.
- **Setting a breakpoint and running the program:** Insert a breakpoint at line “printf( ”Enter three integers:” );”. Then, run the program with the command run. The debugger and program will pause at the breakpoint at line “printf( ”Enter three integers:” )”.
- **Watching a class’s data member:** Set a watch on number1 by typing watch number1. This watch is labeled as watchpoint 2 because watchpoints are labeled with the same sequence of numbers as breakpoints. Whenever the value of a watched variable changes, the debugger enters break mode and notifies you that the value has changed.
- **Continuing execution:** The debugger removes the watch on number1 because number1 goes out of scope when function main ends. Removing the watchpoint causes the debugger to enter break mode. Type continue again to finish execution of the program.
- **Restarting the debugger and resetting the watch on the variable:** Type run to restart the debugger. Once again, set a watch on number1 by typing watch number1. This watchpoint is labeled as watchpoint 3. Type continue to continue execution.
- **Removing the watch on the data member:** Suppose you want to watch a data member for only part of a program’s execution. You can remove the debugger’s watch on variable number1 by typing delete 3.