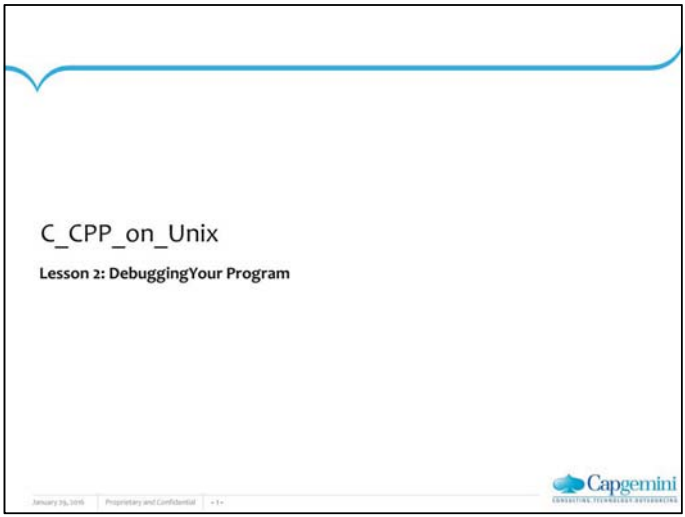
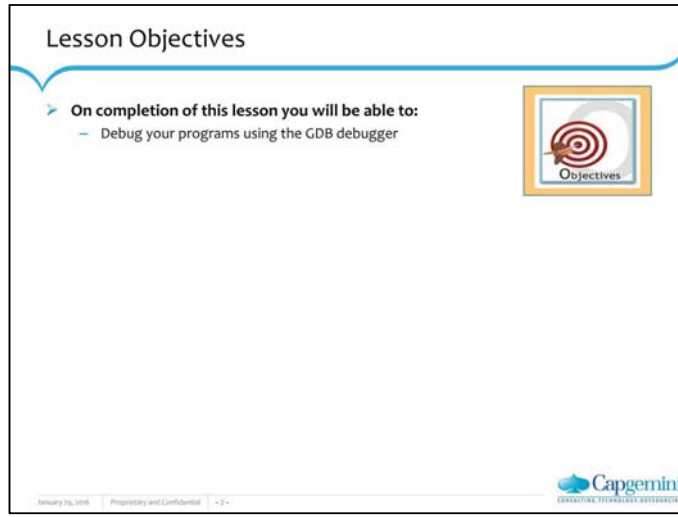
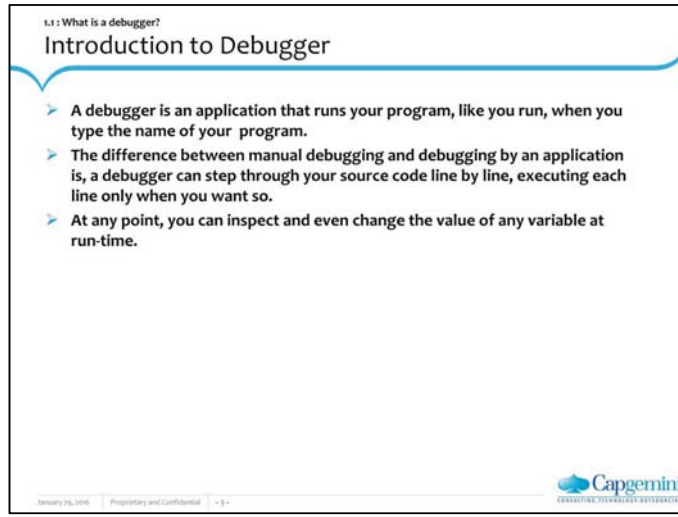
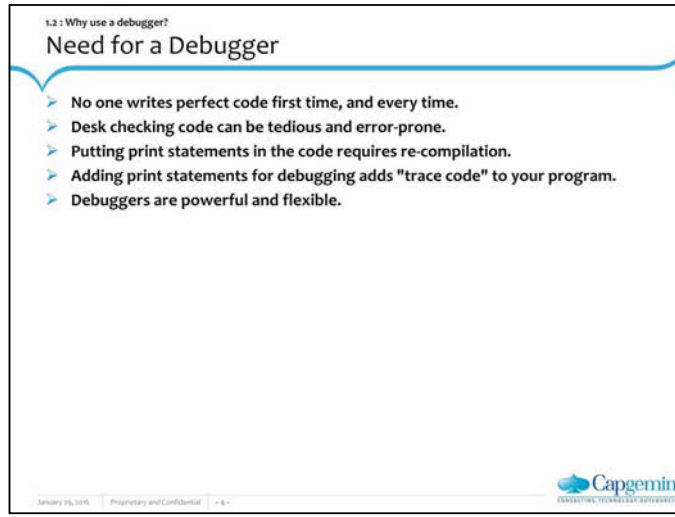

C_CPP_on_Unix
Lesson 2: DebuggingYour Program
Capgemini



Lesson Objectives:
This lesson introduces you to functions of a debugger tool.

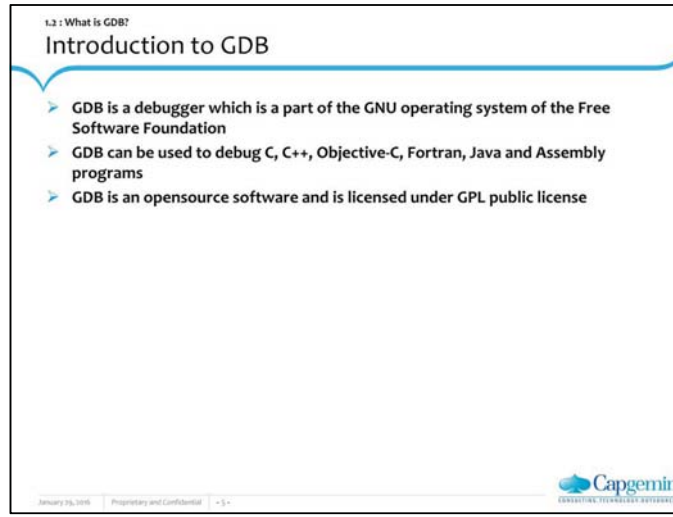


What is a debugger?
A symbolic debugger is an application that enables you to step through your program executing one machine instruction at a time. Following are its advantages:
At any point, you can inspect and even change the value of any variable at run-time.
If your program crashes, a symbolic debugger shows you where and why the program has crashed so that you can deduce the error.
You can go through the program and observe which lines of source code got executed and in which order.
You can use a debugger to step through an infinite loop and observe why your conditions fail to function as per your requirements and specifications.
If your program crashes on a variable access, the debugger shows you all the information about the variable you tried to access and the value you assigned (or perhaps didn't assign) to it.
If a line in your code does not get executed, you can use the debugger to observe what gets executed, in which order, and why a particular line is not reached.
Thus, other than a compiler, the debugger is the most useful tool a programmer can use.

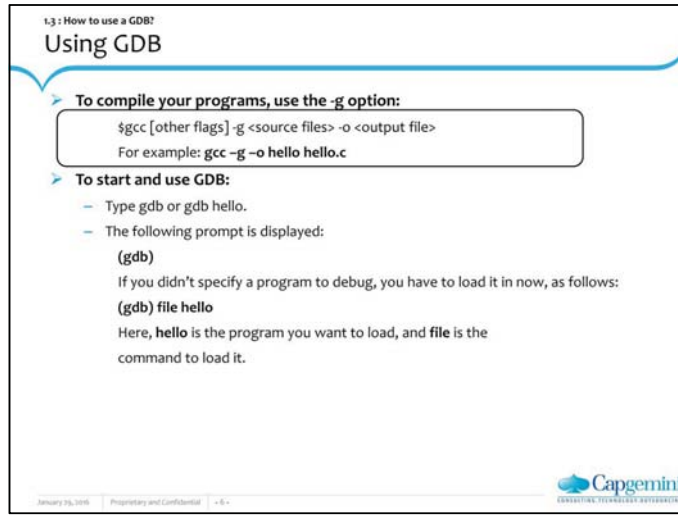


Why a Debugger?
Most people use the printf() debugging method. This is called adding "trace code" to your program. Simply put, they include printf() in their code to view the value of variables at certain strategic points and to examine the order of execution of lines of source code.

There are a few reasons why a debugger is better than the printf() command:
Sometimes, you need many printf() commands, and putting them in and taking them out is tedious. Inserting and deleting superfluous code all the time is distracting.
A symbolic debugger can perform some functions that printf() can't. For example, you can change the value of variables at run-time, halt the program temporarily, list source code, print the datatype of a variable or struct that you don't recognize, jump to an arbitrary line of code, and much more.
You can use a symbolic debugger on a running process (without ending the process). You can use the printf() command for the same.
You can use a symbolic debugger on a process that has already crashed and ended. For that, you do not have to re-run the program. You can view the state in which the program was at the time of its crash and can inspect all the variables.
Knowledge of GDB increases your knowledge of programs, processes, memory and your language of choice.
You can find and fix your bugs faster using a symbolic debugger such as GDB. However, printf() is still useful in debugging.

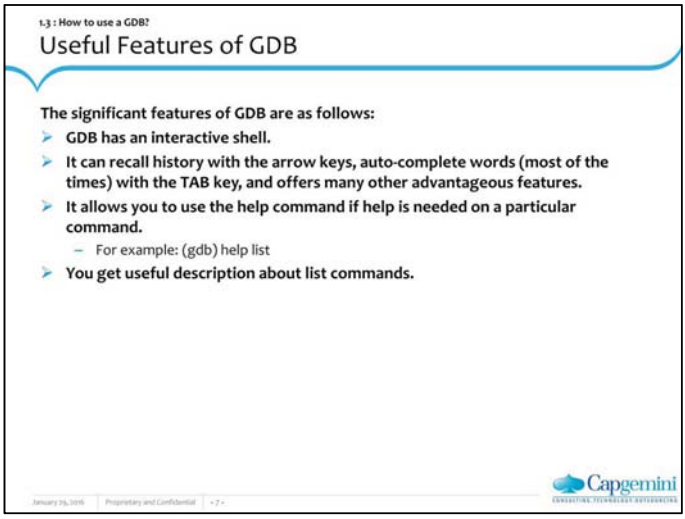


What is GDB?
GDB is a debugger which is a part of the Free Software Foundation's GNU operating system. Its original author is Richard M. Stallman. GDB can be used to debug C, C++, Objective-C, Fortran, Java and Assembly programs. GDB provides partial support for Modula-2 and Pascal.



Using GDB:
If you plan to debug an executable, a corefile resulting from an executable, or a running process, you must compile the executable with an enhanced symbol table. To generate an enhanced symbol table for an executable, you must compile it with gcc's -g option as follows:
gcc -g -o filename filename.c

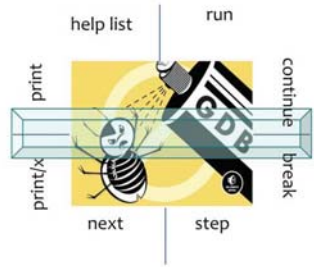
As previously discussed, there are many different debugging formats. The actual function of -g is to produce debugging information in the native format for your system.


1.3 : How to use a GDB?
Useful Features of GDB
The significant features of GDB are as follows:
GDB has an interactive shell.
It can recall history with the arrow keys, auto-complete words (most of the times) with the TAB key, and offers many other advantageous features.
It allows you to use the help command if help is needed on a particular command.
For example: (gdb) help list
You get useful description about list commands.
Capgemini

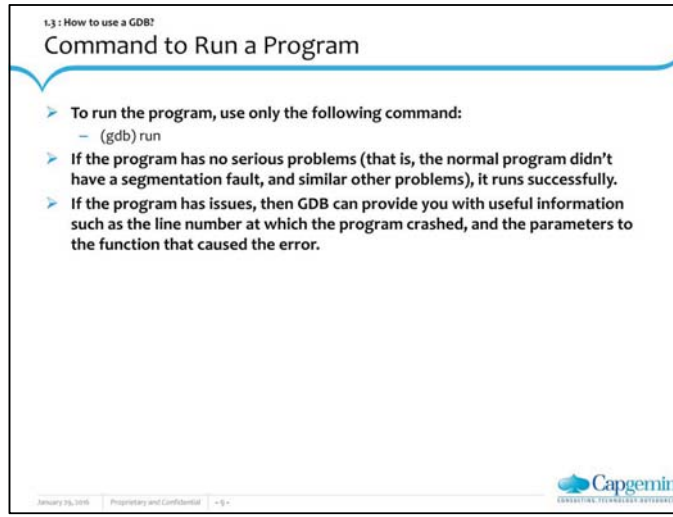
1.3 : How to use a GDB?

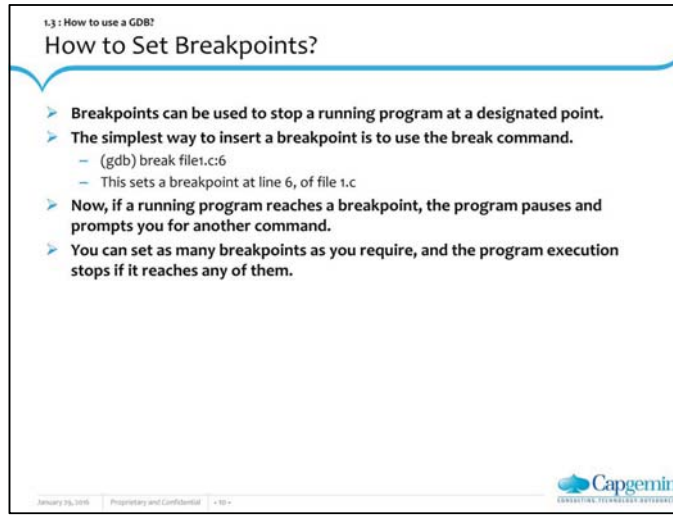
Useful Commands in GDB

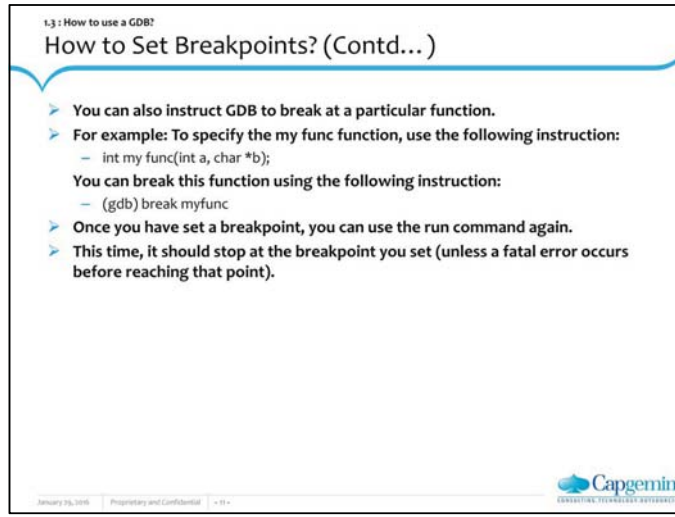
- **The significant commands are as follows:**

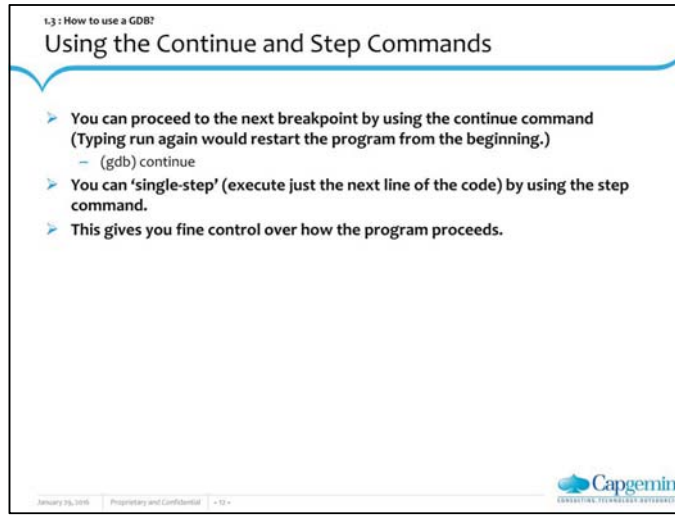


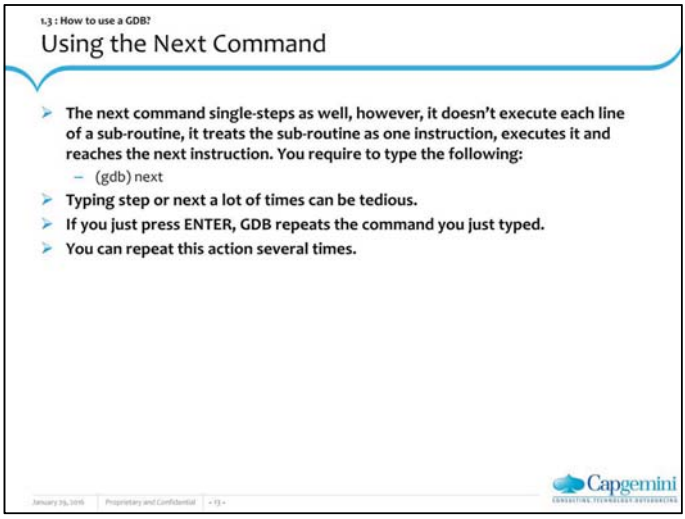
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

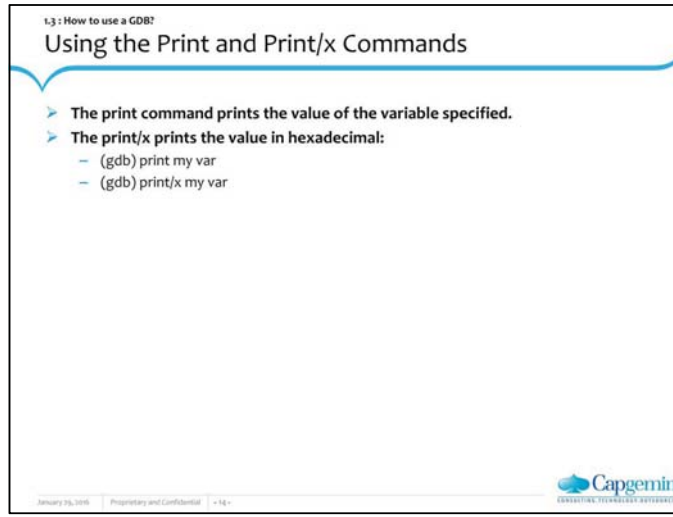

1.3 : How to use a GDB?
Command to Run a Program
To run the program, use only the following command:
(gdb) run
If the program has no serious problems (that is, the normal program didn't have a segmentation fault, and similar other problems), it runs successfully.
If the program has issues, then GDB can provide you with useful information such as the line number at which the program crashed, and the parameters to the function that caused the error.
Capgemini

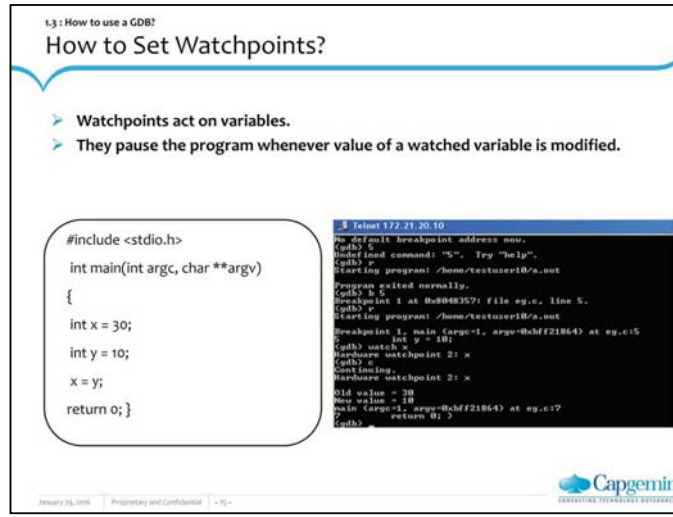

1.3 : How to use a GDB?
How to Set Breakpoints?
Breakpoints can be used to stop a running program at a designated point.
The simplest way to insert a breakpoint is to use the break command.
(gdb) break file1.c:6
This sets a breakpoint at line 6, of file 1.c
Now, if a running program reaches a breakpoint, the program pauses and prompts you for another command.
You can set as many breakpoints as you require, and the program execution stops if it reaches any of them.
Capgemini


1.3 : How to use a GDB?
How to Set Breakpoints? (Contd…)
You can also instruct GDB to break at a particular function.
For example: To specify the my func function, use the following instruction:
int my func(int a, char *b);
You can break this function using the following instruction:
(gdb) break myfunc
Once you have set a breakpoint, you can use the run command again.
This time, it should stop at the breakpoint you set (unless a fatal error occurs before reaching that point).
Capgemini


1.3 : How to use a GDB?
Using the Continue and Step Commands
You can proceed to the next breakpoint by using the continue command (Typing run again would restart the program from the beginning.)
(gdb) continue
You can 'single-step' (execute just the next line of the code) by using the step command.
This gives you fine control over how the program proceeds.
Capgemini


1.3 : How to use a GDB?
Using the Next Command
The next command single-steps as well, however, it doesn't execute each line of a sub-routine, it treats the sub-routine as one instruction, executes it and reaches the next instruction. You require to type the following:
(gdb) next
Typing step or next a lot of times can be tedious.
If you just press ENTER, GDB repeats the command you just typed.
You can repeat this action several times.
Capgemini


1.3 : How to use a GDB?
Using the Print and Print/x Commands
The print command prints the value of the variable specified.
The print/x prints the value in hexadecimal:
(gdb) print my var
(gdb) print/x my var
Capgemini



How to Set Watchpoints?:
Watchpoints are similar to breakpoints. However, watchpoints are not set for functions or lines of code. Watchpoints are set on variables. When those variables are read or written, the watchpoint is triggered and the program execution stops.

Setting a write watchpoint for a variable:

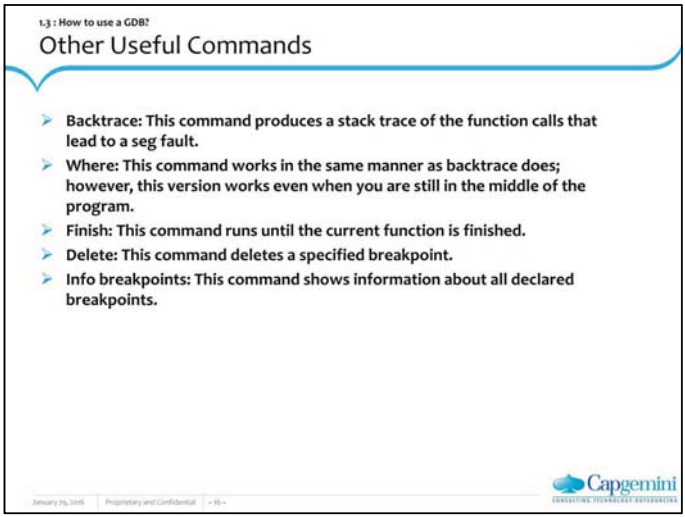
For this, use the watch command. The argument to the watch command is an expression that is evaluated. This implies that the variable on which you want to set a watchpoint must be in the current scope. So, to set a watchpoint on a non-global variable, you must set a breakpoint that stops your program when the variable is in scope. You set the watchpoint after the program breaks.

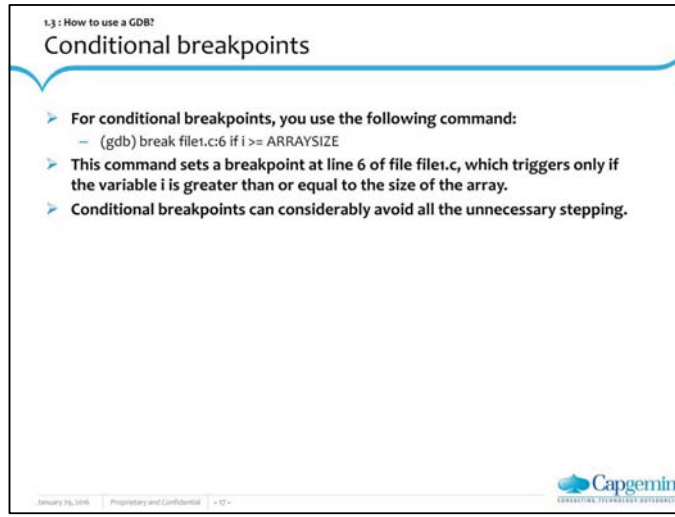
Setting a read watchpoint for a variable:

For this, use the rwatch command. Usage is identical to the watch command.

Setting a read/write watchpoint for a variable:

For this, use the awatch command. Usage is identical to the watch command.


1.3 : How to use a GDB?
Other Useful Commands
Backtrace: This command produces a stack trace of the function calls that lead to a seg fault.
Where: This command works in the same manner as backtrace does; however, this version works even when you are still in the middle of the program.
Finish: This command runs until the current function is finished.
Delete: This command deletes a specified breakpoint.
Info breakpoints: This command shows information about all declared breakpoints.
Capgemini


1.3 : How to use a GDB?
Conditional breakpoints
For conditional breakpoints, you use the following command:
(gdb) break file1.c:6 if i >= ARRAYSIZE
This command sets a breakpoint at line 6 of file file1.c, which triggers only if the variable i is greater than or equal to the size of the array.
Conditional breakpoints can considerably avoid all the unnecessary stepping.
Capgemini

Summary

- **In this lesson, you learnt the following:**
  - The various stages of Compilation
  - gcc compiler and its various options
  - g++ compiler and its various options

Review Question

- **Question 1: Why is the gcc –c option used?**
  - To produce an executable file
  - To produce an object file
  - To produce warnings
- **Question 2: Which of the following is true regarding the Wall option in the gcc compiler?**
  - It is used to display warnings.
  - It is used to display message to all users.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING.

Review Question

- **Question 3: Which compiler is used to compile C++ programs?**
  - gcc
  - g++
  - Both Option1 and Option2 can be used