1. 控制流如何修改状态 (Control Flow Modifies State)

- **语句的执行**：程序中的语句按顺序执行，某些语句（如操作符、表达式等）会直接修改程序的状态。
- **逻辑控制语句的执行顺序**：程序逻辑决定了哪些语句会被执行，以及它们的执行顺序。操作符优先级等因素会进一步影响执行的顺序。
  - **例子**：后缀 `i++` 和前缀 `++i` 的行为差异，操作符的优先级会影响状态的修改顺序。
- **调试建议**：可以使用调试器（如 `gdb`）来跟踪执行顺序，监视状态值的变化。

2. 为什么需要调试程序 (Why Debug a Program?)

调试是一种用于检查程序行为的过程，主要用于：

- **当程序崩溃时**：通常由于段错误或浮点运算错误。调试可以帮助我们查看程序崩溃时的状态，如通过 core 文件检查内存状态。
- **当程序中出现 Bug 时**：调试可以帮助我们观察变量的状态，确定问题发生的原因和位置。
- **通过跟踪程序行为**：调试可以帮助我们学习程序的实际行为，验证程序逻辑是否符合预期。
  - **例子**：当遇到意外的行为时，通过 `gdb` 观察变量值，确定异常的原因。

3. 如何调试程序 (How to Debug a Program)

- **调试器的作用**：调试器（如 `gdb`）可以帮助我们观察程序的行为，逐步执行代码，监视函数调用栈和变量值。
  - **逐步执行**：通过单步调试，观察程序每一步的执行结果。
  - **监视变量值**：观察变量值的变化，检查它们是否符合预期。
- **调试前的思考**：
  - 在调试之前，应该先明确预期的程序行为、可能的失败点，并做出假设和预测，这将有助于更高效地进行调试。

GDB 调试示例

Q2：运行程序的示例

```
[l.tingjun@shell studio3]$ ./studio3 + 3 + 4 3
The value calculated is 10
[l.tingjun@shell studio3]$ echo $?
0
[l.tingjun@shell studio3]$ ./studio3 + 3 +  300
caught exception for unexpected end of expression.
[l.tingjun@shell studio3]$ echo $?
2
```

- 程序运行成功，返回值为 0。
- 当遇到异常时，捕获到表达式错误并返回错误码 2。

Q3：启动 GDB

```
[l.tingjun@shell studio3]$ gdb studio3
GNU gdb (GDB) Rocky Linux 8.2-20.el8.0.1
...
Reading symbols from studio3...done.
(gdb)
```

- 使用 `gdb studio3` 启动调试，准备进行断点和逐步调试。

Q4：设置断点

```
(gdb) break parse_and_compute
Breakpoint 1 at 0x401b71: file studio3.cpp, line 103.
```

- 设置断点，在函数 `parse_and_compute` 的第 103 行处暂停程序执行。

Q5：运行程序并触发断点

```
(gdb) run + 1 + 2 + 3 4
Starting program: /home/warehouse/l.tingjun/cse428_fall24/studios/studio3/studio3 + 1 + 2 + 3 4

Breakpoint 1, parse_and_compute (current_index=@0x7fffffffc974: 1, last_index=7, argv=0x7fffffff
103         if (current_index > last_index)
```

- 程序运行到断点处暂停，显示当前代码行。

Q6：检查调用栈

```
(gdb) where
#0  parse_and_compute (current_index=@0x7fffffffc974: 2, last_index=7, argv=0x7fffffffca78) at s
#1  0x0000000000401c30 in parse_and_compute (current_index=@0x7fffffffc974: 2, last_index=7, arg
#2  0x00000000004019d4 in main (argc=8, argv=0x7fffffffca78) at studio3.cpp:53
```

- 使用 `where` 命令查看调用栈，当前处于 `parse_and_compute` 函数内。

Q7：打印变量的值

```
(gdb) print current_index
$7 = (int &) @0x7fffffffc974: 2
(gdb) print argv[2]
$8 = 0x7fffffffcf1c "1"
```

- 使用 `print` 命令打印当前变量的值， `current_index` 为 2， `argv[2]` 为 "1"。

GDB 常用命令总结

- **启动调试**： `gdb studio3`
- **设置断点**： `break [function_name]`
  - 示例： `break parse_and_compute` ，在函数 `parse_and_compute` 处设置断点。
- **运行程序**： `run [arguments]`
  - 示例： `run + 1 + 2 + 3 4` ，带参数运行程序。
- **继续执行**： `continue` ，继续执行程序，直到下一个断点。
- **显示调用栈**： `where` ，查看当前调用栈。
- **逐步执行**：
  - `step` ：进入函数逐步执行。
  - `next` ：执行下一行，跳过函数调用。
- **打印变量**： `print [variable]` ，显示变量的当前值。
  - 示例： `print current_index` 打印 `current_index` 的值。

考题示例

考题示例 1:
"What command in GDB would you use to set a breakpoint at the start of the `parse_and_compute` function?"

答案: `break parse_and_compute`

考题示例 2:
"How do you continue program execution after hitting a breakpoint in GDB?"

答案: `continue`

考题示例 3:
"What does the command `print current_index` do in GDB?"

答案：该命令会打印变量 `current_index` 的当前值。