### **Advanced GDB Debugging: Step-by-Step with a More Complex C++ Program**

We'll now **debug a more complex C++ program** that involves:

* **Dynamic memory allocation** (new / delete)
* **Segmentation faults**
* **Breakpoint debugging**
* **Backtrace and stepping through functions**
* **Examining variables and modifying them live**

I'll provide **detailed explanations** of every **GDB command, output, and concept** as we debug.

## **1. Write a More Complex Buggy C++ Program (buggy.cpp)**

#include <iostream>

class Node {

public:

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

class LinkedList {

public:

Node\* head;

LinkedList() : head(nullptr) {}

void addNode(int val) {

Node\* newNode = new Node(val);

newNode->next = head;

head = newNode;

}

void printList() {

Node\* temp = head;

while (temp != nullptr) {

std::cout << temp->data << " -> ";

temp = temp->next;

}

std::cout << "NULL" << std::endl;

}

void deleteList() {

Node\* temp = head;

while (temp != nullptr) {

Node\* nextNode = temp->next;

delete temp;

temp = nextNode;

}

head = nullptr;

}

};

int main() {

LinkedList\* list = nullptr; // Oops! Uninitialized list

list->addNode(10); // SEGFAULT: Dereferencing a nullptr

list->printList(); // Never reached due to crash

list->deleteList();

delete list;

return 0;

}

### **Bug Explanation:**

1. **list is declared as nullptr**, but we try calling list->addNode(10);
2. Since list is nullptr, trying to access list->addNode() causes a **segmentation fault** (dereferencing nullptr).
3. The program crashes **before printing anything** or deleting memory.

## **2. Compile with Debug Symbols**

To enable debugging in **GDB**, compile using:

g++ -g buggy.cpp -o buggy

* -g: **Includes debug symbols** (so GDB can show exact lines, variable names).
* -o buggy: Output executable **buggy**.

Run the program normally:

./buggy

### **Expected Output (Crash)**

Segmentation fault (core dumped)

This means the program **tried to access invalid memory**.

## **3. Start GDB and Load the Program**

Now, start **GDB** with our program:

gdb ./buggy

You will see:

GNU gdb (GDB) 12.1

Reading symbols from buggy...

(gdb)

* **Reading symbols from buggy...** → GDB successfully loaded our program with debug symbols.
* **(gdb)** → This is the GDB **command prompt** where we issue debugging commands.

## **4. Run the Program in GDB**

Let's run the program **inside GDB**:

(gdb) run

### **Output:**

Starting program: ./buggy

Program received signal SIGSEGV, Segmentation fault.

0x0000000000401196 in main () at buggy.cpp:34

34 list->addNode(10);

### **Explanation:**

* **Program received signal SIGSEGV** → **Segmentation fault** (attempted to access memory incorrectly).
* **0x0000000000401196** → Memory address where the crash occurred.
* **in main() at buggy.cpp:34** → The error happened **inside main() on line 34**.
* **list->addNode(10);** → This is the exact **failing instruction**!

## **5. Examine the Call Stack (Backtrace)**

To see **where the program crashed** and how it got there:

(gdb) backtrace

### **Output:**

#0 0x0000000000401196 in main () at buggy.cpp:34

### **Explanation:**

* **#0** → Shows the **most recent function call before the crash**.
* **main()** → The crash happened in main().
* **at buggy.cpp:34** → Happened **at line 34**.

## **6. Print Variable Values**

Check what list is at the time of the crash:

(gdb) print list

### **Output:**

$1 = (LinkedList \*) 0x0

### **Explanation:**

* **$1** → GDB assigns a **reference number** to this result.
* **(LinkedList \*) 0x0** → list is nullptr (0x0 means NULL).
* **Dereferencing nullptr is why our program crashed!**

## **7. Fix the Bug Inside GDB (Modify Variables)**

Instead of **editing the code**, let’s **fix the bug dynamically in GDB** by **changing list before it is used**.

Set list to a **valid object**:

(gdb) set variable list = new LinkedList()

Now confirm the change:

(gdb) print list

### **Output:**

$2 = (LinkedList \*) 0x55555575d2a0

* **Now list has a valid memory address**, meaning it’s no longer nullptr.

## **8. Continue Execution After Fix**

Now, continue running the program:

(gdb) continue

### **Output:**

Value: 10 -> NULL

✅ **Fixed! The program runs successfully inside GDB without crashing.**

## **9. Set a Breakpoint for Future Debugging**

If we want to stop execution **before** the crash in future debugging sessions:

(gdb) break main

Now, **restart the program**:

(gdb) run

### **Output:**

Breakpoint 1, main () at buggy.cpp:33

33 list->addNode(10);

Execution **pauses** at main() before executing list->addNode(10);, so we can inspect variables **before the crash happens**.

## **10. Final Fix: Modify the Code**

Now, update main() to **properly initialize list**:

int main() {

LinkedList\* list = new LinkedList(); // Proper initialization

list->addNode(10);

list->printList();

list->deleteList();

delete list;

return 0;

}

Recompile:

g++ -g buggy.cpp -o buggy

./buggy

### **Output:**

10 -> NULL

🎉 **Bug fixed! The program now works correctly.**

## **11. Summary of Important GDB Commands**

| **Command** | **Description** |
| --- | --- |
| gdb ./program | Start GDB with program. |
| run | Run the program inside GDB. |
| backtrace | Show the function call stack. |
| print <var> | Print the value of a variable. |
| set variable <var> = <value> | Change a variable’s value. |
| break <function> | Set a breakpoint in a function. |
| break <line> | Set a breakpoint at a line. |
| continue | Resume execution after a breakpoint. |
| quit | Exit GDB. |

## **Conclusion**

We used **GDB** to:  
 ✅ Identify a segmentation fault.  
 ✅ Inspect variables and memory.  
 ✅ Modify a variable in GDB **without changing the code**.  
 ✅ Set breakpoints and step through code.  
 ✅ Fix the bug properly in C++.

Would you like a **more complex exercise** to debug yourself? 🚀