

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN**



**BÁO CÁO BÀI THỰC HÀNH
HỌC PHẦN: KIỂM THỬ XÂM PHẬP
MÃ HỌC PHẦN: INT14107**

BÀI THỰC HÀNH: GDB-CPP

Họ tên sinh viên:

B21DCAT151 Trần Thị Thu Phương

Tên nhóm: Nhóm 04

Tên lớp: Lớp 03

Giảng viên hướng dẫn: Đinh Trường Duy

HÀ NỘI 2025

Mục lục

1.	Mục đích	3
2.	Lý thuyết.....	3
2.1	Khái niệm GDB	3
2.2	Kiến trúc của GDB	3
2.3	Chức năng của GDB.....	3
3.	Thực hành	4
4.	Kết quả.....	8

1. Mục đích

- Sử dụng tiện ích GDB để gỡ lỗi một chương trình C++
- Gỡ lỗi là điều không thể tránh khỏi. Mỗi lập trình viên tại một thời điểm nào đó trong sự nghiệp phải gỡ lỗi một đoạn mã. Có nhiều cách để bắt đầu gỡ lỗi, từ in thông báo ra màn hình, sử dụng trình gỡ lỗi hoặc chỉ nghĩ về những gì chương trình đang làm và đưa ra một phỏng đoán về vấn đề. Trước khi một lỗi có thể được sửa, nguồn của lỗi phải được xác định. Ví dụ, với các lỗi phân đoạn, sẽ rất hữu ích khi biết lỗi đang xảy ra trên dòng mã nào. Khi dòng mã được đề cập đã được tìm thấy, sẽ rất hữu ích nếu biết về các giá trị trong phương thức đó, cái gì đã gọi phương thức và tại sao (cụ thể) lại xảy ra lỗi. Sử dụng trình gỡ lỗi làm cho việc tìm kiếm tất cả thông tin này trở nên đơn giản

2. Lý thuyết

2.1 Khái niệm GDB

GDB (GNU Debugger) là một trình gỡ lỗi mã nguồn mở do GNU phát triển. Nó giúp lập trình viên:

- Chạy chương trình từng bước để quan sát cách nó hoạt động
- Kiểm tra giá trị của biến, thanh ghi, vùng nhớ
- Thay đổi giá trị của biến trong khi chạy chương trình
- Xác định lỗi (như segmentation fault) và tìm nguyên nhân gây lỗi
- Gỡ lỗi chương trình từ xa qua gdbserver

2.2 Kiến trúc của GDB

GDB hoạt động theo mô hình client-server, gồm 4 thành phần chính:

- Debugger (GDB): Giao diện tương tác, nơi nhập lệnh
- Debuggee (Program): Chương trình được debug
- Symbol Table: Lưu thông tin về biến, hàm, dòng lệnh
- Execution Control: Điều khiển chương trình, đặt breakpoint, theo dõi stack

2.3 Chức năng của GDB

GDB (GNU Debugger) cung cấp nhiều chức năng mạnh mẽ hỗ trợ quá trình gỡ lỗi chương trình. Một số chức năng chính bao gồm:

- **Điều khiển thực thi chương trình:** GDB cho phép chạy chương trình từ đầu (run), dừng thực thi (Ctrl+C), tiếp tục chạy (continue), thực hiện từng bước (next, step), chạy đến khi kết thúc hàm (finish), hoặc nhảy đến một dòng cụ thể (jump).

- **Quản lý breakpoint:** Người dùng có thể đặt breakpoint tại dòng hoặc hàm cụ thể (break), liệt kê (info breakpoints), xóa (delete), và tắt/bật breakpoint (disable, enable) để kiểm soát luồng thực thi.
- **Kiểm tra trạng thái chương trình:** GDB hỗ trợ xem giá trị các biến cục bộ (info locals), đối số hàm (info args), thanh ghi (info registers), in giá trị biến (print), xem nội dung bộ nhớ (x/<n> <address>), và theo dõi ngăn xếp lệnh (backtrace).
- **Chỉnh sửa giá trị trong quá trình chạy:** Có thể thay đổi giá trị biến (set variable) hoặc giá trị thanh ghi (set \$<register>), hỗ trợ kiểm thử các tình huống khác nhau.
- **Phân tích lỗi với core dump:** GDB hỗ trợ làm việc với file core dump bằng cách bật ghi core (ulimit -c unlimited), mở file core (gdb ./program core), và xác định nguyên nhân lỗi thông qua lệnh bt (backtrace).

3. Thực hành

- Khởi động bài lab

Khởi động môi trường lab, chạy lệnh

```
labtainer -r gdb-cpp
```

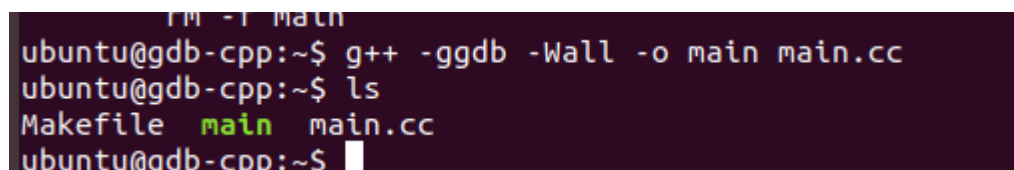
Môi trường lab được khởi động. Để giúp minh họa một số nguyên tắc gỡ lỗi, lab sử dụng một ví dụ đang chạy của một chương trình lỗi. Sinh viên sẽ sử dụng gdb để định vị và sửa lỗi trong mã. Mã và một tệp makefile đơn giản nằm trên máy tính gdb-cpp khởi động khi chạy lab này. Mã chương trình này rất đơn giản và bao gồm hai định nghĩa lớp, một nút và một danh sách liên kết. Ngoài ra còn có một chương trình đơn giản để kiểm tra danh sách. Tất cả mã được đặt vào một tệp duy nhất để minh họa quá trình gỡ lỗi dễ dàng hơn. Chương trình và makefile nằm trên máy tính được tạo khi lab khởi động và có thể được nhìn thấy trong thư mục chính.

- Build và chạy

Hãy tiếp tục tạo chương trình và chạy chương trình. Chương trình sẽ in ra một số thông báo và sau đó nó sẽ in ra rằng nó đã nhận được tín hiệu lỗi phân đoạn, dẫn đến sự cố chương trình. Với thông tin trên màn hình tại thời điểm này, gần như không thể xác định lý do tại sao chương trình bị lỗi. Hãy bắt đầu gỡ lỗi chương trình này.

Biên dịch chương trình sử dụng câu lệnh sau

```
g++ -ggdb -Wall -o main main.cc
```



```
ubuntu@gdb-cpp:~$ g++ -ggdb -Wall -o main main.cc
ubuntu@gdb-cpp:~$ ls
Makefile  main  main.cc
ubuntu@gdb-cpp:~$
```

- Nạp một chương trình trong gdb

Trước tiên phải khởi chạy trình gỡ lỗi. Trình gỡ lỗi được gọi là gdb và ta có thể cho nó biết tệp nào cần gỡ lỗi tại dấu nhắc shell. Vì vậy, để gỡ lỗi main, ta chạy tệp thực thi main trong thư mục hiện tại và gỡ gdb main.

```
ubuntu@gdb-cpp:~$ ./main
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1

Segmentation fault (core dumped)
ubuntu@gdb-cpp:~$
```

```
ubuntu@gdb-cpp:~$ gdb main
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
(gdb)
```

Gdb sẽ hiện ra và đợi người dùng gõ lệnh. Ta cần chạy chương trình để trình gỡ lỗi có thể giúp xem điều gì sẽ xảy ra khi chương trình bị treo. Nhập run tại dấu nhắc (gdb)

```
ubuntu@gdb-cpp: ~
File Edit View Search Terminal Help
(gdb) run
Starting program: /home/ubuntu/main
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1

Program received signal SIGSEGV, Segmentation fault.
0x000055555555586c in Node<int>::next (this=0x0) at main.cc:28
28      Node<T>* next () const { return next_; }
(gdb)
```

Chương trình bị lỗi. Vì vậy hãy xem loại thông tin nào có thể thu thập. Kiểm tra sự cố, ta có thể thấy chương trình ở dòng 28 của main.cc, điểm này chỉ đến 0 và chúng ta có thể thấy dòng mã đã được thực thi. Nhưng ta cũng muốn biết cái gì đã gọi phương thức này và muốn có thể kiểm tra các giá trị trong các phương thức gọi. Vì vậy, tại dấu nhắc gdb, nhập backtrace cho kết quả sau

```
Program received signal SIGSEGV, Segmentation fault.
0x000055555555586c in Node<int>::next (this=0x0) at main.cc:28
28      Node<T>* next () const { return next_; }
(gdb) backtrace
#0  0x000055555555586c in Node<int>::next (this=0x0) at main.cc:28
#1  0x0000555555555763 in LinkedList<int>::remove (this=0x55555556aeb0,
      item_to_remove=@0x7fffffffe43c: 1) at main.cc:77
#2  0x00005555555553b1 in main (argc=1, argv=0x7fffffffe558) at main.cc:120
(gdb)
```

Vì vậy, ngoài những gì đã biết về phương thức hiện tại và các biến cục bộ, bây giờ ta cũng có thể xem những phương thức nào được gọi và tham số của chúng là gì. Ví dụ, có thể thấy rằng chương trình đã được gọi bởi LinkedList :: remove () trong đó mục tham số cần loại bỏ ở địa chỉ 0x7fffffffe43c. Nó có thể giúp hiểu lỗi nếu biết giá trị của mục cần xóa, vì vậy cần xem tiếp giá trị tại địa chỉ của mục cần xóa. Điều này có thể được thực hiện bằng lệnh x sử dụng địa chỉ làm tham số. ("X" có thể được coi là viết tắt của "exam"). Kết quả sau khi chạy lệnh (gdb) x 0x7fffffffe43c như sau:

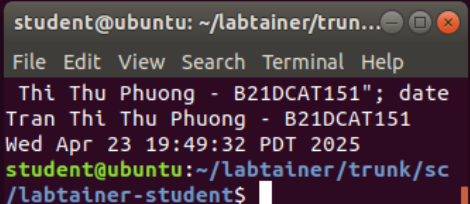
```
(gdb) backtrace
#0  0x000055555555586c in Node<int>::next (this=0x0) at main.cc:28
#1  0x0000555555555763 in LinkedList<int>::remove (this=0x55555556aeb0,
      item_to_remove=@0x7fffffffe43c: 1) at main.cc:77
#2  0x00005555555553b1 in main (argc=1, argv=0x7fffffffe558) at main.cc:120
(gdb) x @0x7fffffffe43c
A syntax error in expression, near `0x7fffffffe43c'.
(gdb) x 0x7fffffffe43c
A syntax error in expression, near `0x7fffffffe43c'.
(gdb) x 0x7fffffffe43c
0x7fffffffe43c: 0x00000001
(gdb)
```

Vì vậy, chương trình gặp sự cố trong khi cố gắng chạy `LinkedList :: remove` với tham số là 1. Bây giờ ta đã thu hẹp vấn đề xuống một hàm cụ thể và một giá trị cụ thể cho tham số.

- Các điểm ngắt có điều kiện

Bây giờ ta biết segfault đang xảy ra ở đâu và khi nào, chúng ta muốn xem chương trình đang làm gì trước khi nó bị treo. Một cách để làm điều này là thực hiện lần lượt từng câu lệnh của chương trình cho đến khi đi đến điểm thực thi mà ta muốn xem điều gì đang xảy ra. Cách làm này đúng, nhưng đôi khi ta có thể muốn chỉ chạy đến một phần mã cụ thể và dừng thực thi tại thời điểm đó để có thể kiểm tra dữ liệu tại vị trí đó. Nếu đã từng sử dụng trình gỡ lỗi, bạn có thể quen thuộc với khái niệm điểm ngắt (breakpoint). Về cơ bản, điểm ngắt là một dòng trong mã nguồn nơi trình gỡ lỗi sẽ ngắt thực thi. Trong ví dụ này, để xem mã trong `LinkedList :: remove ()`, ta đặt một điểm ngắt ở dòng 52 của `main.cc`. Gõ lệnh (gdb) `break 52` sẽ thu được kết quả như sau:

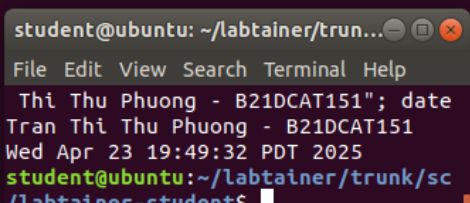
```
(gdb) backtrace
#0  0x000055555555586c in Node<int>::next (this=0x0) at main.cc:28
#1  0x0000555555555763 in LinkedList<int>::remove (this=0x55555556aeb0,
    item_to_remove=@0x7fffffffe43c: 1) at main.cc:77
#2  0x00005555555553b1 in main (argc=1, argv=0x7fffffffe558) at main.cc:120
(gdb) x @0x7fffffffe43c
A syntax error in expression, near `0x7fffffffe43c'.
(gdb) x @0x7fffffffe43c
A syntax error in expression, near `0x7fffffffe43c'.
(gdb) x 0x7fffffffe43c
0x7fffffffe43c: 0x00000001
(gdb) break 52
Breakpoint 1 at 0x555555555f9: file main.cc, line 52.
(gdb)
```



Lúc này Breakpoint 1 được đặt tại `main.cc`, dòng 52. (Lý do điểm ngắt nhận một số là vì ta có thể tham khảo điểm ngắt sau này, chẳng hạn như nếu muốn xóa nó.) Vì vậy, khi chương trình được chạy, nó sẽ trả lại quyền điều khiển cho trình gỡ lỗi mỗi khi nó đến dòng 52. Điều này có thể không mong muốn nếu phương thức được gọi nhiều lần nhưng chỉ có vấn đề với một số giá trị nhất định được truyền. Các điểm ngắt có điều kiện có thể giúp ta lúc này. Ví dụ: ta biết rằng chương trình bị treo khi `LinkedList :: remove ()` được gọi với giá trị là 1. Vì vậy, ta có thể yêu cầu trình gỡ lỗi chỉ ngắt ở dòng 52 nếu mục cần xóa bằng 1. Điều này có thể được thực hiện bằng cách ra lệnh sau:

(gdb) `condition 1 item_to_remove==1`

```
(gdb) x @0x7fffffffe43c
A syntax error in expression, near `0x7fffffffe43c'.
(gdb) x @0x7fffffffe43c
A syntax error in expression, near `0x7fffffffe43c'.
(gdb) x 0x7fffffffe43c
0x7fffffffe43c: 0x00000001
(gdb) break 52
Breakpoint 1 at 0x555555555f9: file main.cc, line 52.
(gdb) condition 1 item_to_remove==1
(gdb)
```



- Lệnh Step

Như vậy ta đã đặt một điểm ngắt có điều kiện và bây giờ muốn xem qua hàm này từng dòng một và xem liệu có thể xác định được nguồn gốc của lỗi hay không. Điều này được

thực hiện bằng cách sử dụng lệnh step. gdb có một tính năng hay là khi nhấn enter mà không cần gõ lệnh thì lệnh cuối cùng sẽ tự động được sử dụng. Bằng cách đó, chúng ta có thể thực hiện một cách đơn giản bằng cách nhấn vào phím enter sau khi bước đầu tiên đã được nhập (gdb) run

```
(gdb) condition 1 item_to_remove==1
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ubuntu/main
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1

Breakpoint 1, LinkedList<int>::remove (this=0x55555556aeb0,
    item_to_remove=@0x7fffffffe43c: 1) at main.cc:52
52      Node<T> *marker = head_;
(gdb)
```

Nếu nhìn vào kết quả đầu ra từ việc chạy chương trình, trước hết ta sẽ thấy rằng chương trình chạy mà không bị treo, nhưng có một chỗ rò rỉ bộ nhớ ở đâu đó trong chương trình. Nó nằm trong hàm LinkedList :: remove (). Một trong những trường hợp remove không hoạt động bình thường.

```
Breakpoint 1, LinkedList<int>::remove (this=0x55555556aeb0,
    item_to_remove=@0x7fffffffe43c: 1) at main.cc:52
52      Node<T> *marker = head_;
(gdb) step
53      Node<T> *temp = 0; // temp points to one behind as we iterate
(gdb)
55      while (marker != 0) {
(gdb)
56          if (marker->value() == item_to_remove) {
(gdb)
Node<int>::value (this=0x7ffff7f1344e <std::ostream::put(char)+94>)
    at main.cc:30
30      const T& value () const { return value_; }
(gdb)
LinkedList<int>::remove (this=0x55555556aeb0,
    item_to_remove=@0x7fffffffe43c: 1) at main.cc:75
75      marker = 0; // reset the marker
(gdb)
76      temp = marker;
(gdb)
77      marker = marker->next();
(gdb)
Node<int>::next (this=0x55555556b360) at main.cc:28
28      Node<T>* next () const { return next_; }
(gdb)
```

Sau khi hoàn thành việc kiểm tra có thể thoát gdb bằng cách gõ quit

4. Kết quả

- Tìm hiểu kiến thức cơ bản về gdb
- Tìm hiểu về các lệnh thông dụng trong gdb
- Trả lời các câu hỏi và hoàn thành checkwork

Checkwork:

```
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/gdb-cpp
Labname gdb-cpp

Student          | gdb_commands |
=====|=====|
B21DCAT151      |          22  |
What is automatically assessed for this lab:

      gdb_commands: How many gdb commands issued by the student
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$ echo "Tran Thi Thu Phuong - B21DCAT151"; date
Tran Thi Thu Phuong - B21DCAT151
Wed Apr 23 19:58:12 PDT 2025
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$
```

Trần Thị Thu Phương (B21DCAT151) 1:36:47

Lịch sử nộp bài			
ID	Thời gian	Bài tập	Kết quả
20379	2025-04-24 10:04:05	Gỡ rối chương trình với GDB	1/1 (AC)