

Ngôn ngữ lập trình C

Bài 1. Tổng quan

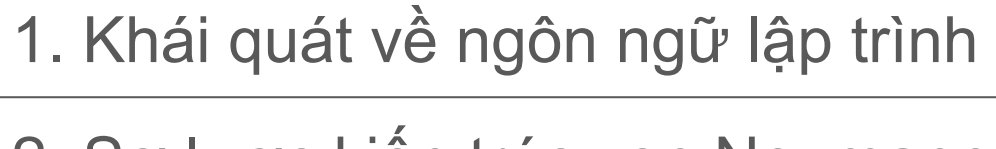
Soạn bởi: TS. Nguyễn Bá Ngọc

2021

Nội dung

1. Khái quát về ngôn ngữ lập trình
2. Sơ lược kiến trúc von Neumann
3. Lịch sử phát triển NNLT C
4. Môi trường lập trình
5. Tiến trình biên dịch với gcc

Nội dung

- 
1. Khái quát về ngôn ngữ lập trình
 2. Sơ lược kiến trúc von Neumann
 3. Lịch sử phát triển NNLT C
 4. Môi trường lập trình
 5. Tiến trình biên dịch với gcc

Ngôn ngữ lập trình

- Ứng dụng chính của ngôn ngữ lập trình là gì?
 - Công cụ ra lệnh điều khiển thiết bị tính toán;
 - Phương tiện biểu diễn giải thuật;
 - Công cụ để tiến hành thực nghiệm và phân tích dữ liệu;
 - Công cụ để tự động hóa các thao tác với máy tính;
 - Công cụ để tạo trang Web; v.v..
 - *Có nhiều quan điểm, không có 1 câu trả lời thống nhất.*
- Ngoài các mục đích sử dụng được đặt ra khi phát triển ngôn ngữ lập trình, thiết kế cơ bản của ngôn ngữ lập trình còn chịu ảnh hưởng của các yếu tố môi trường điển hình như: Nguyên lý hoạt động của môi trường thực thi (phần cứng, máy ảo, v.v..) và phương pháp phát triển phần mềm.

Ngôn ngữ lập trình₍₂₎

- Sự ảnh hưởng qua lại giữa ngôn ngữ lập trình và tư duy lập trình diễn ra theo cả 2 chiều:
 - Ngôn ngữ lập trình giới hạn hệ thống khái niệm và cấu trúc có thể được sử dụng để lập trình.
 - Tuy nhiên từ nhu cầu diễn đạt những ý tưởng lập trình có thể dẫn đến việc sáng tạo ra những thành phần ngôn ngữ mới hoặc cả ngôn ngữ lập trình mới.
- Một vấn đề lập trình nếu có thể được giải quyết bằng 1 ngôn ngữ lập trình thì cũng có thể được giải quyết bằng các ngôn ngữ lập trình khác (*tuy nhiên các lời giải có thể khác nhau đáng kể về độ khó và tính hiệu quả*).
- Người lập trình nên học nhiều ngôn ngữ lập trình để biết nhiều khái niệm hay, để sử dụng sáng tạo và hiệu quả các công cụ để giải quyết các vấn đề lập trình.

Phân lớp ngôn ngữ lập trình

- Có thể phân lớp các ngôn ngữ lập trình theo 3 lớp chính:
 - **Mệnh lệnh/Imperative** (ví dụ C) - Có thiết kế tương thích với kiến trúc von Neumann
 - Biến mô phỏng ô nhớ, phép gán đưa dữ liệu vào ô nhớ, chương trình mô tả chính xác trình tự thực hiện lệnh.
 - **Hàm/Functional** (ví dụ Lisp) - Áp dụng hàm cho các tham số khác nhau (không cần biến, phép gán, vòng lặp).
 - **Dựa trên luật/Logic** (ví dụ Prolog) - Công cụ thực hiện chương trình tự xác định trình tự thực hiện các mệnh đề lô-gic.
- Ngoài ra còn có các quan điểm phân lớp khác:
 - Hướng đối tượng/OOP (ví dụ Java)
 - Script (ví dụ Python, Javascript)
 - ... Tuy nhiên các ngôn ngữ Java, Python và Javascript cũng có đầy đủ các đặc trưng của lớp Mệnh lệnh.

Phân lớp ngôn ngữ lập trình có giới hạn mềm, 1 ngôn ngữ có thể thuộc nhiều hơn 1 lớp.

Ví dụ 1.1. Lập trình mệnh lệnh (với C)

```
#include <stdio.h>
int main() {
    int x = 3, y = 7;
    printf("%d\n", x + y);
}
```

Ví dụ 1.2. Lập trình hàm (với LISP)

```
(defun average3 (n1 n2 n3)
  (/ (+ n1 n2 n3) 3)
)
(write(average3 1 2 6))
```


Ví dụ 1.3. Lập trình Lô-gic (với Prolog)

```
parent(jhon,bob).
```

```
parent(lili,bob).
```

```
male(jhon).
```

```
female(lili).
```

```
% Lô-gic và/hội
```

```
father(X,Y) :- parent(X,Y),male(X).
```

```
mother(X,Y) :- parent(X,Y),female(X).
```

```
% Lô-gic hoặc/tuyển
```

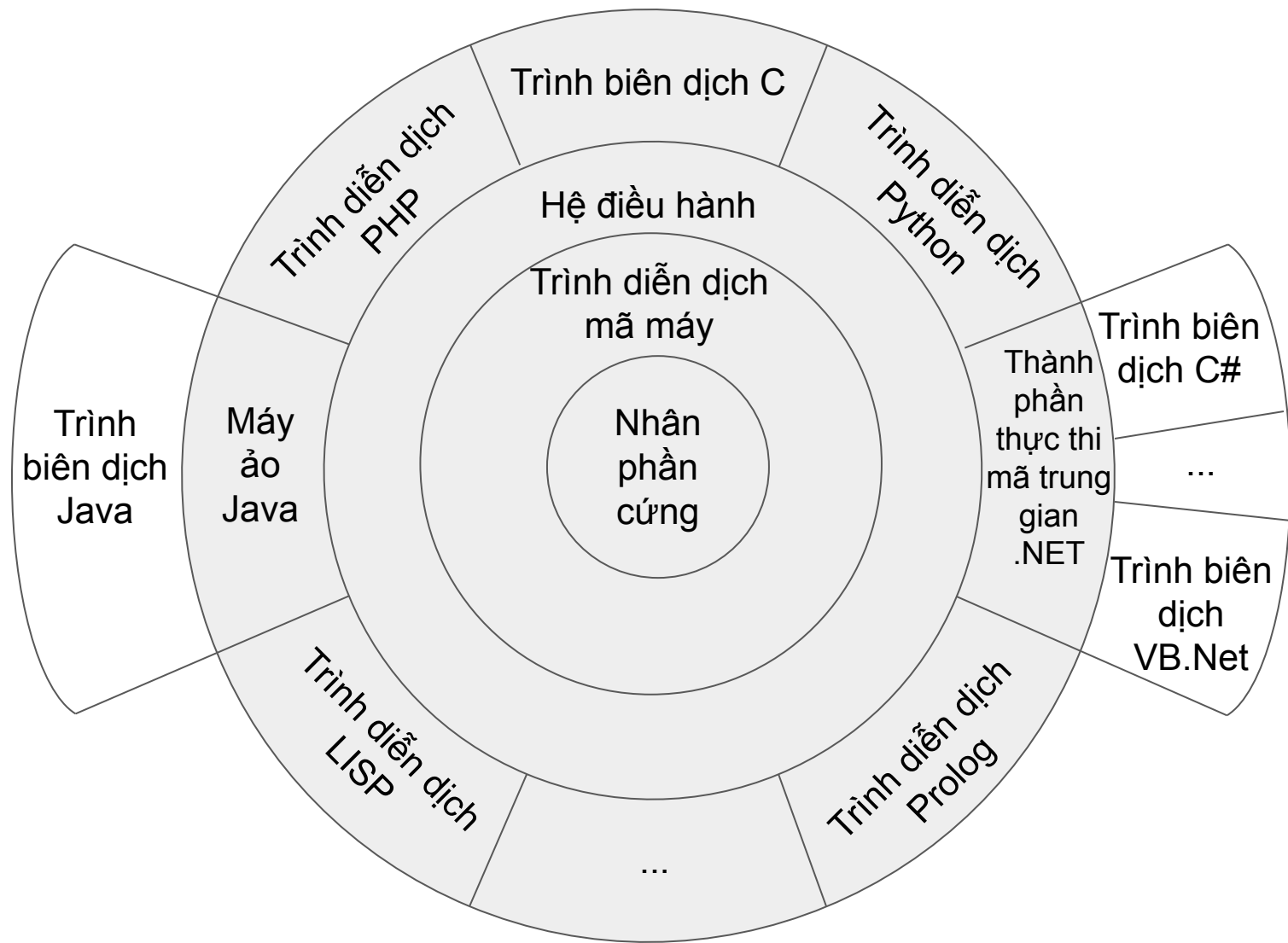
```
child_of(X,Y) :- father(X,Y);mother(X,Y).
```

Các phương pháp triển khai

- Biên dịch/Compile: Chuyển đổi mã nguồn thành mã máy.
 - Ví dụ: Assembly, C, C++
- Diễn dịch/Interpret: Thực hiện bằng phần mềm
 - Ví dụ: Python, PHP
- Kết hợp biên dịch và diễn dịch:
 - Biên dịch sang mã trung gian sau đó diễn dịch bằng máy ảo
 - Ví dụ: Java, nền tảng .Net.
 - Biên dịch khi cần (Just-In-Time compilation, JIT) - Ví dụ: V8 (được sử dụng trong Chrome và Node.js) có cơ chế chọn lọc để biên dịch mã Javascript thành mã máy khi cần thiết nhằm đạt hiệu năng thực thi cao hơn.

Một ngôn ngữ có thể được triển khai theo nhiều cách, ví dụ phát triển cả Trình biên dịch và Trình diễn dịch.

Các tầng trừu tượng hóa



Nội dung

1. Khái quát về ngôn ngữ lập trình

2. Sơ lược kiến trúc von Neumann

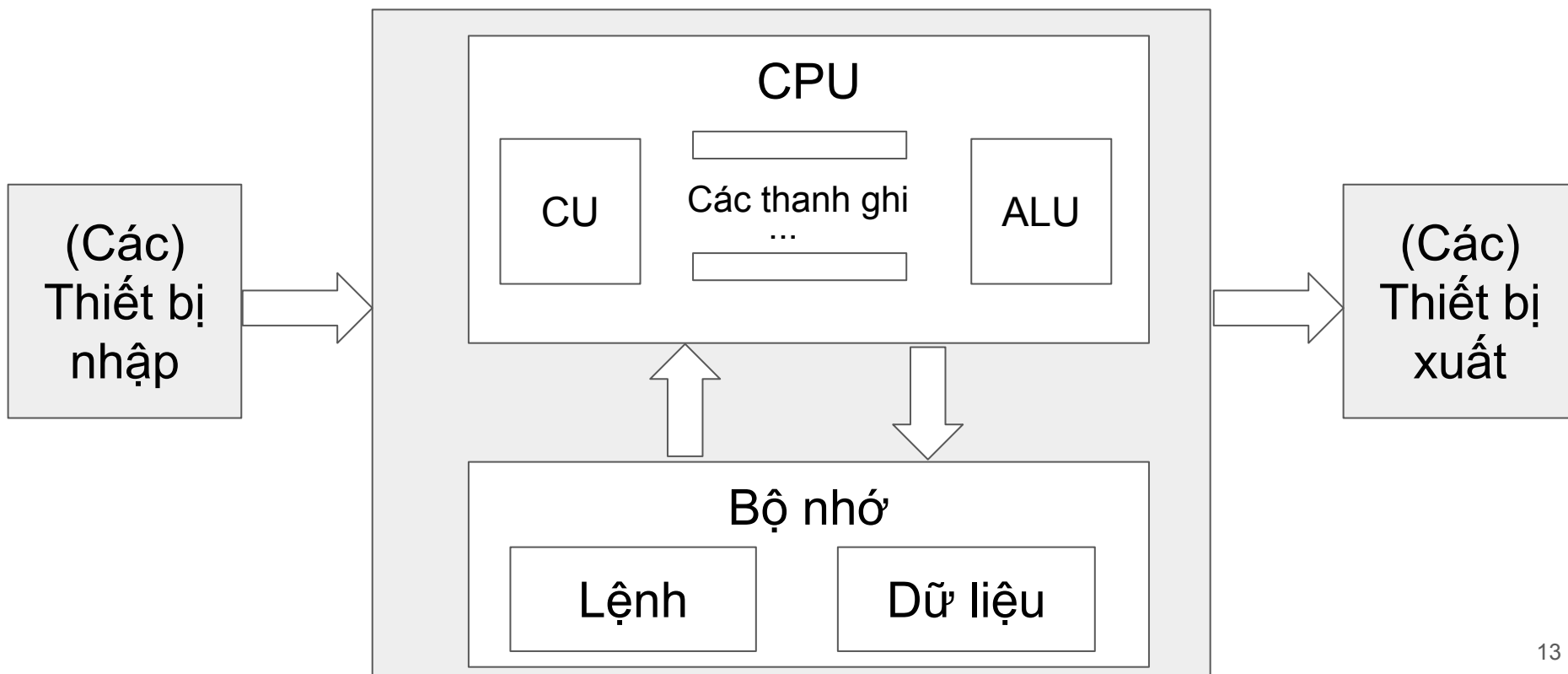
3. Lịch sử phát triển NNLT C

4. Môi trường lập trình

5. Tiến trình biên dịch với gcc

Kiến trúc von Neumann

- Lệnh và dữ liệu có cùng bản chất biểu diễn và được lưu trong cùng 1 bộ nhớ.
- Bộ nhớ và CPU được tách rời nhau.
- Về bản chất các lệnh được xử lý tuần tự.



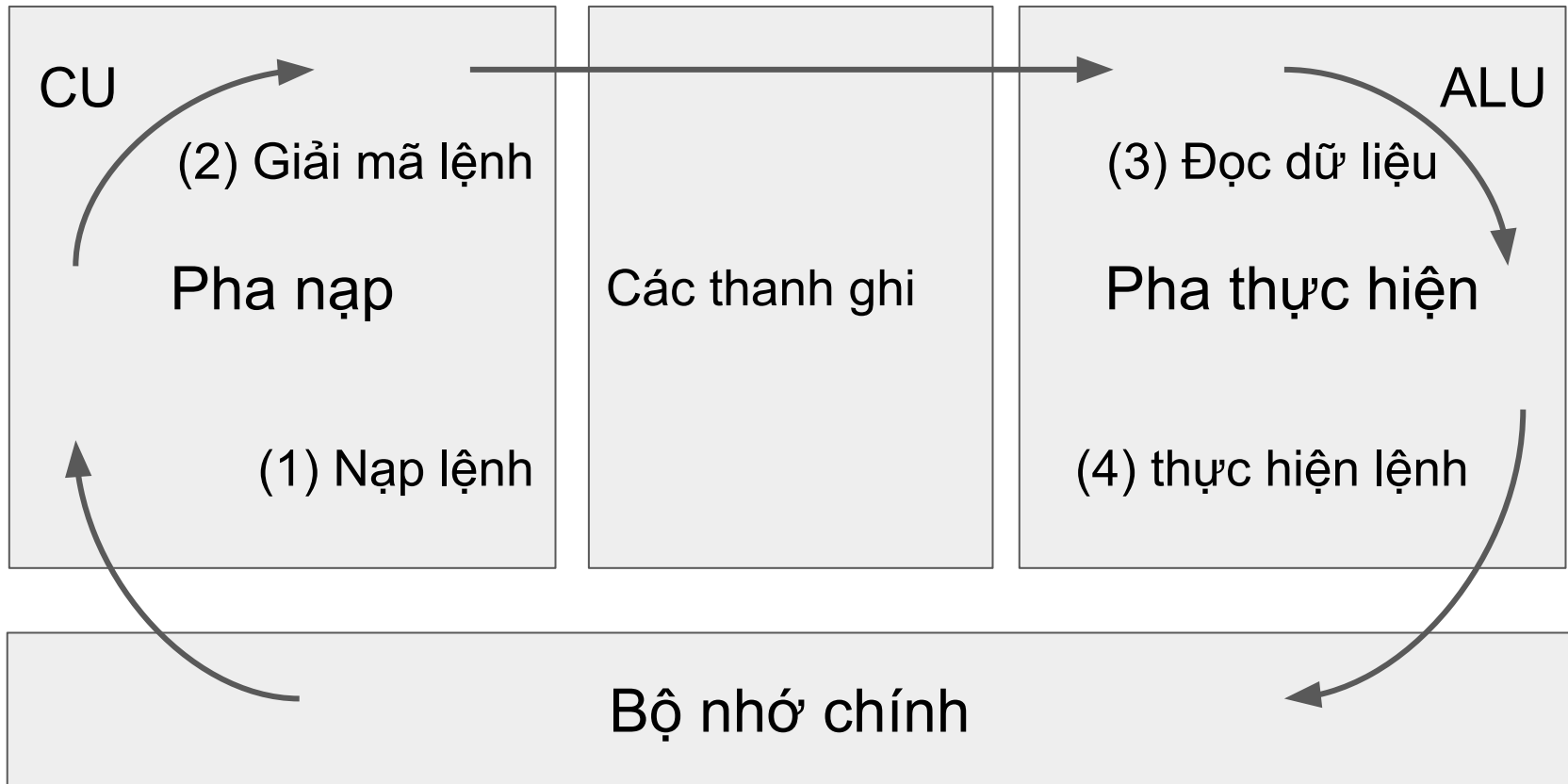
Vòng lặp Nạp-Thực hiện lệnh

Các bước trong 1 chu trình lệnh:

- 1) CU sao chép lệnh trong bộ nhớ có địa chỉ đang được lưu ở thanh ghi PC (**P**rogram **C**ounter) vào thanh ghi IR (**I**nstruction **R**egister);
- 2) CU chuyển đổi mã lệnh đang được lưu trong IR thành tín hiệu điều khiển;
- 3) Trong trường hợp lệnh đang được thực hiện có tham số thì CU đọc các tham số từ bộ nhớ vào các thanh ghi;
- 4) Các tín hiệu được truyền tới ALU để xử lý, kết quả thực hiện lệnh được sao chép sang bộ nhớ.

Giá trị của PC phải được cập nhật trước khi chuyển sang chu kỳ lệnh tiếp theo: Có thể được cộng thêm một hằng số thành địa chỉ của lệnh tiếp theo trong khối nhớ hoặc được cập nhật bởi lệnh vừa được thực hiện.

Sơ đồ vòng lặp Nạp-Thực hiện lệnh

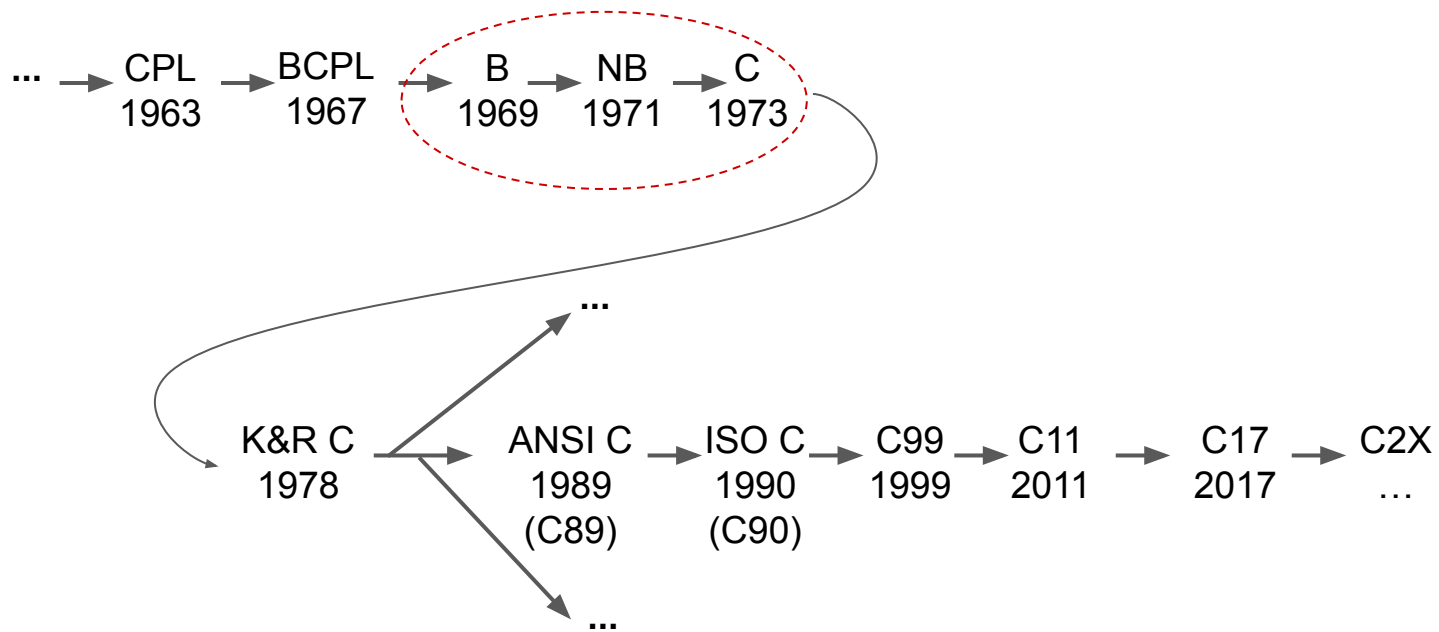


Nội dung

1. Khái quát về ngôn ngữ lập trình
2. Sơ lược kiến trúc von Neumann
3. Lịch sử phát triển NNLT C
4. Môi trường lập trình
5. Tiến trình biên dịch với gcc



Sơ lược lịch sử phát triển NNLT C



Lịch sử phát triển NNLT C

- Ban đầu (từ những năm 1970) C được phát triển để viết Hệ Điều Hành (HĐH) Unix
 - HĐH Unix ban đầu được viết bởi Ken Thompson bằng hợp ngữ ở phòng thí nghiệm của Bell. Sau đó ông đã quyết định viết lại Unix bằng ngôn ngữ bậc cao hơn, và ông cũng đã tạo ra ngôn ngữ B cho mục đích này.
 - Năm 1971 ngôn ngữ B bắt đầu bộc lộ các hạn chế, không đáp ứng được các nhu cầu trên PDP-11, vì vậy Dennis Ritchie sau khi gia nhập dự án Unix đã bắt đầu mở rộng B.
 - Ban đầu Ritchie gọi ngôn ngữ do mình phát triển là NB (*New B*/ B mới), theo thời gian phần khác biệt so với B ngày càng lớn, khi phần khác biệt đã đủ lớn Ritchie đã đổi tên ngôn ngữ thành C.
 - Đến năm 1973 ngôn ngữ C đã có đủ các tính năng cần thiết để có thể viết lại toàn bộ HĐH Unix bằng C.

Lịch sử phát triển NNLT C₍₂₎

- Sau đó phạm vi sử dụng C được mở rộng sang nhiều môi trường cho nhiều mục đích khác nhau và có nhiều trình biên dịch được phát triển độc lập
 - Sách *The C Programming Language* được, Brian Kernighan và Dennis Ritchie (phiên bản *K&R*), 1978, là tài liệu đầu tiên mô tả chi tiết về C.
 - Bước sang những năm 1980, nhiều trình biên dịch C cho các môi trường khác nhau được phát triển, C được sử dụng ngày càng phổ biến cả bên ngoài cộng đồng Unix:
 - Các lập trình viên sử dụng *K&R* làm tham chiếu để viết trình biên dịch, nhưng một số tính năng được mô tả khá nhập nhằng và các lập trình viên đã triển khai những tính năng đó theo cách riêng.
 - C vẫn tiếp tục được phát triển sau khi *K&R* được xuất bản.
 - ... dẫn đến nguy cơ bất tương thích giữa các triển khai trình biên dịch và làm mất tính khả chuyển của mã nguồn C. Vì vậy vấn đề chuẩn hóa đã được đặt ra.

Lịch sử phát triển NNLT C₍₃₎

- Để duy trì tính tương thích giữa các trình biên dịch khi NNLT C ngày càng được sử dụng rộng rãi hơn, các quy chuẩn cho NNLT C đã được biên soạn.
 - Quy chuẩn C của Mỹ bắt đầu được phát triển từ 1983 dưới sự điều hành của viện ANSI.
 - Viện ANSI thông qua quy chuẩn đầu tiên X3.159-1989 năm 1989, và không lâu sau đó được tổ chức ISO công nhận như quy chuẩn quốc tế ISO/IEC 9899:1990 năm 1990. Phiên bản này của C thường được gọi là C89 hoặc C90.
 - NNLT C vẫn liên tục được hiệu chỉnh, cập nhật và phát triển trong các phiên bản mới hơn:
 - ISO/IEC 9899:1999 - được gọi là C99
 - ISO/IEC 9899:2011 - C11
 - ISO/IEC 9899:2018 - C17 hoặc C18
 - ... - C2x đang trong giai đoạn hoàn thiện

Các quy chuẩn mở rộng

- Ngoài các tính năng ngôn ngữ được mô tả trong quy chuẩn ISO, trình biên dịch còn có thể bổ xung thêm các tính năng chưa/không có trong quy chuẩn ISO. Các tính năng được thêm vào được gọi là các mở rộng.
 - GCC và Clang có thể biên dịch mã nguồn C theo các quy chuẩn ISO, và các quy chuẩn gnu:
 - gnu90 = C90 + các mở rộng
 - gnu11 = C11 + các mở rộng
 - gnu17 = C17 + các mở rộng
 - Các mở rộng, điển hình như các mở rộng của GNU và CLang, thường có thể được thay thế bởi các tính năng có trong quy chuẩn (nhưng điển đạt có thể phức tạp hơn).

Vì sao học lập trình C?

- Để minh họa những vấn đề lập trình đơn giản: C có thiết kế nhỏ gọn và các cấu trúc lập trình cơ bản trong C cũng rất đơn giản và dễ học.
- Để sử dụng chuyên sâu: C là ngôn ngữ chính để lập trình ở tầng cơ bản:
 - Có thể biên dịch thành mã máy tối ưu, hoạt động nhanh.
 - Có khả năng mạnh mẽ để làm việc với phần cứng, truy cập trực tiếp vào bộ nhớ của hệ thống máy tính.
 - Có thể tạo chương trình kích thước nhỏ, sử dụng hiệu quả các tài nguyên tính toán, kể cả các hệ thống máy tính có nguồn tài nguyên hạn chế.
- Sau khi học tốt 1 ngôn ngữ lập trình việc học các ngôn ngữ lập trình khác sẽ đơn giản hơn.

Vì sao sử dụng các quy chuẩn ISO?

Có nhiều phiên bản C: Quy chuẩn, ngoài quy chuẩn, các mở rộng, v.v..

Các tính năng thuộc quy chuẩn ISO có tính ổn định cao, được hỗ trợ tốt bởi nhiều trình biên dịch, giúp làm tăng tính *khả chuyển* của chương trình trong nhiều môi trường khác nhau.

Nội dung

1. Khái quát về ngôn ngữ lập trình
2. Sơ lược kiến trúc von Neumann
3. Lịch sử phát triển NNLT C
4. Môi trường lập trình
5. Tiến trình biên dịch với gcc



Lựa chọn hệ điều hành

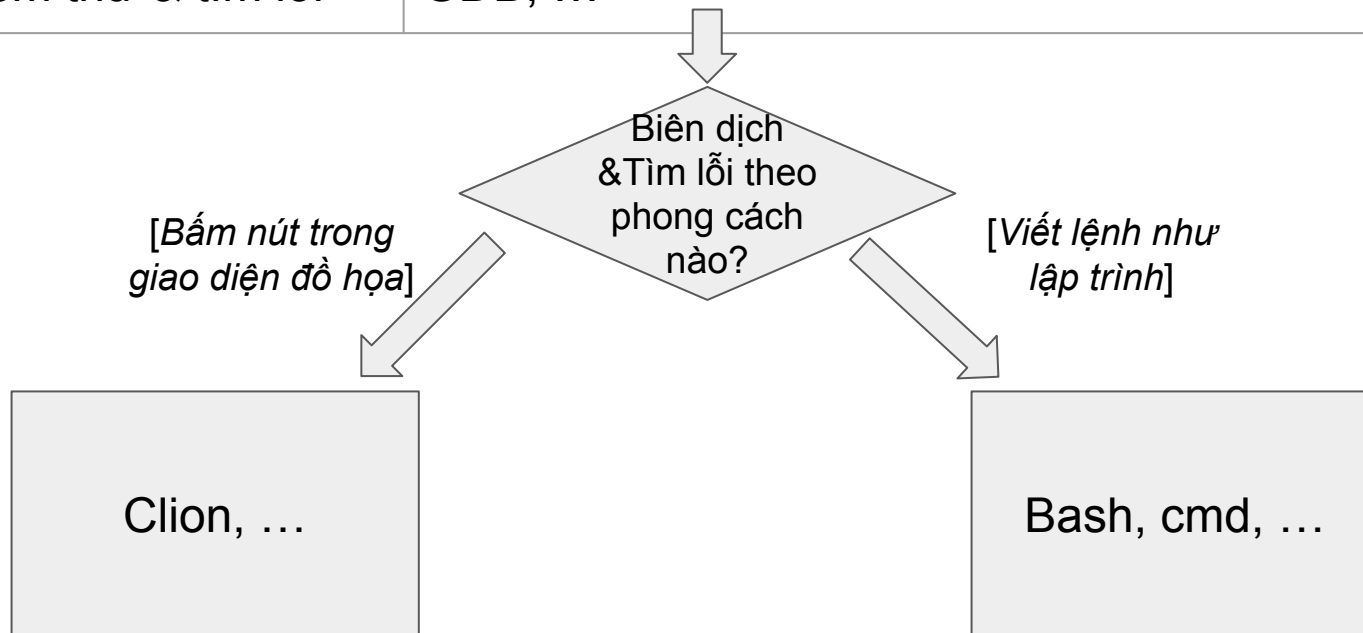
Có thể học NNLT C trên bất kỳ HĐH nào

- Khuyến khích sử dụng hệ điều hành (dựa trên) Linux
 - Linux được sử dụng phổ biến cho các máy chủ, miễn phí
 - Môi trường thuận lợi để lập trình, hỗ trợ C rất tốt
 - Môi trường dòng lệnh thuận tiện, có thể sử dụng UTF-8.
- Các trường hợp khác
 - macOS:
 - Như Linux, cũng hỗ trợ C rất tốt
 - ... Nhưng thường đắt hơn
 - Windows:
 - Thiên về phát triển C++, ít đầu tư phát triển C.
 - Môi trường dòng lệnh kém phát triển, chưa hỗ trợ tốt UTF-8 (chcp 65001), hạn chế khả năng hiển thị tiếng Việt.
 - ... Có thể cài đặt máy ảo Linux và chia sẻ thư mục với máy chủ

Lựa chọn các công cụ

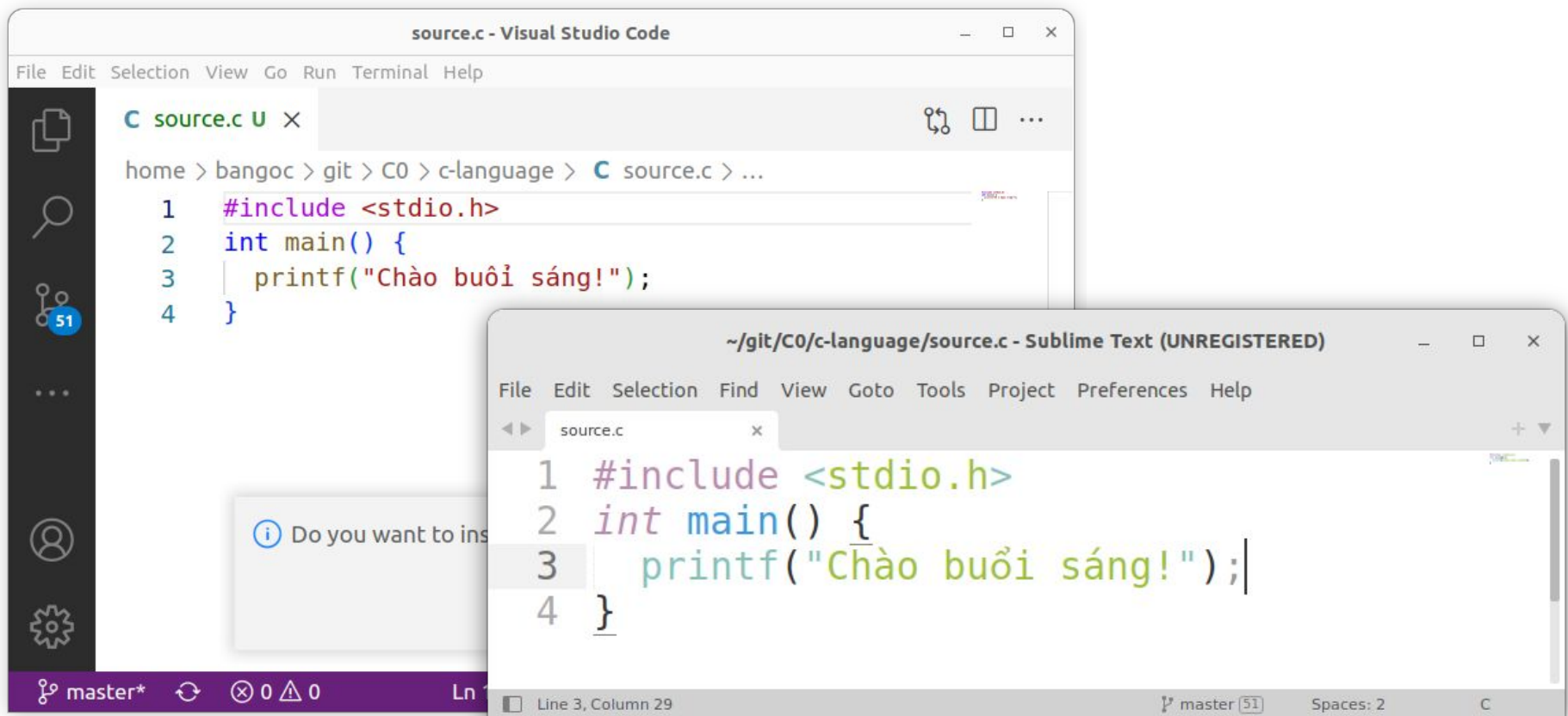
Một số hoạt động lập trình C thường gặp và công cụ:

Hoạt động	Công cụ
Viết mã nguồn	Sublime Text, Visual Studio Code, Emacs, Vim, ...
Biên dịch	GCC, Clang, ...
Kiểm thử & tìm lỗi	GDB, ...



Soạn thảo mã nguồn C

- Sử dụng trình soạn thảo mã nguồn hiểu cú pháp C và ưu tiên công cụ đa nền tảng
 - Ví dụ: Sublime Text, Visual Studio Code, v.v..
- Đặt tên tệp với phần mở rộng .c



Trình biên dịch C

- Sử dụng trình biên dịch hỗ trợ tốt các quy chuẩn ngôn ngữ C và ưu tiên công cụ đa nền tảng
 - Ví dụ: GCC, Clang
- Một số phiên bản phổ biến của GCC:
 - Windows: MinGW (được phân phối qua TDM GCC).
 - Linux: Thường có trong kho phần mềm tiêu chuẩn
 - `sudo apt-get install gcc`
 - `sudo apt-get install build-essential`
 - macOS: Thường có trong kho phần mềm tiêu chuẩn
 - `brew install gcc`

Phần lớn các minh họa trong bộ trình chiếu này được thực hiện với gcc trong môi trường dòng lệnh của Ubuntu (Bash)

Biên dịch mã nguồn C

- Mở môi trường dòng lệnh
 - *Lưu ý thư mục hiện hành và đường dẫn tệp.*
- Chạy lệnh biên dịch:
 - Định dạng: gcc -o <tên chương trình đầu ra> <mã nguồn>
 - Ví dụ: gcc -o hello xin-chao.c

```
bangoc:$gcc -o prog source.c
bangoc:$./prog
Chào buổi sáng!bangoc:$
```

Các câu lệnh biên dịch thường dùng

`gcc -o prog source.c`

Các chi tiết:

`-o`: Cờ biên dịch

`prog`: Tham số của cờ `-o`, tên tệp đầu ra (tệp thực thi)

`source.c`: Tệp mã nguồn đầu vào

Dịch mã nguồn `source.c` thành chương trình `prog`

Cài đặt GCC: Môi trường GNU/Linux

** Có nhiều hệ điều hành khác nhau*

Giả sử Ubuntu:

```
sudo apt install build-essential
```

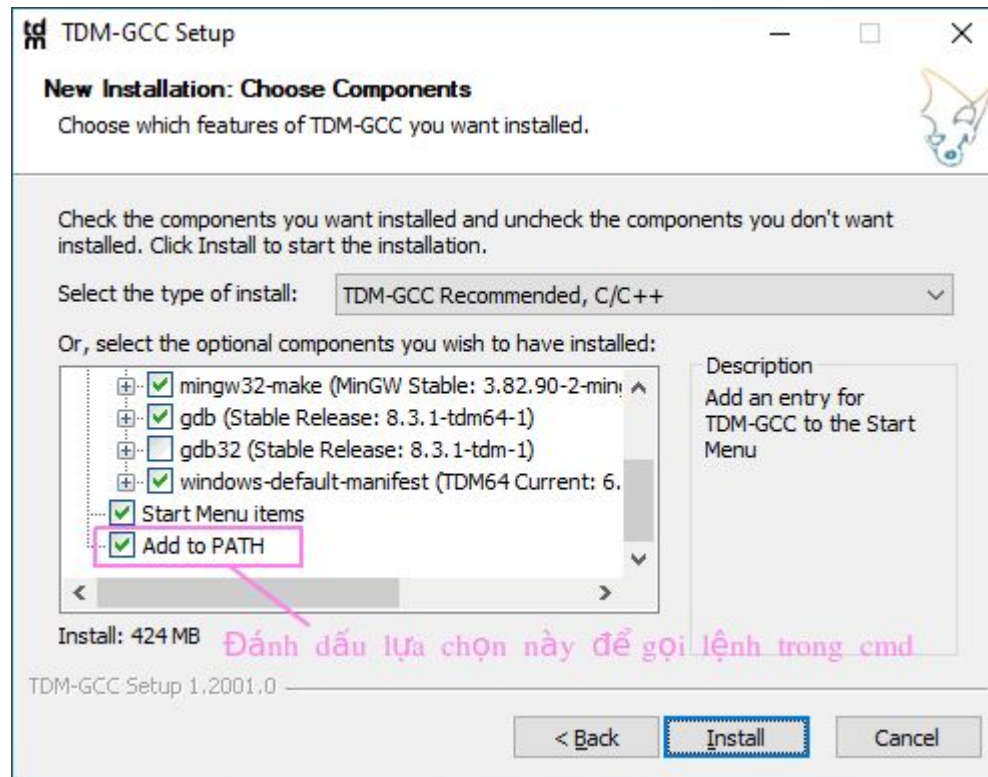
Kiểm tra: Thử thực hiện các lệnh trong môi trường dòng lệnh

```
gcc --version
```

...

Cài đặt GCC: Môi trường Windows

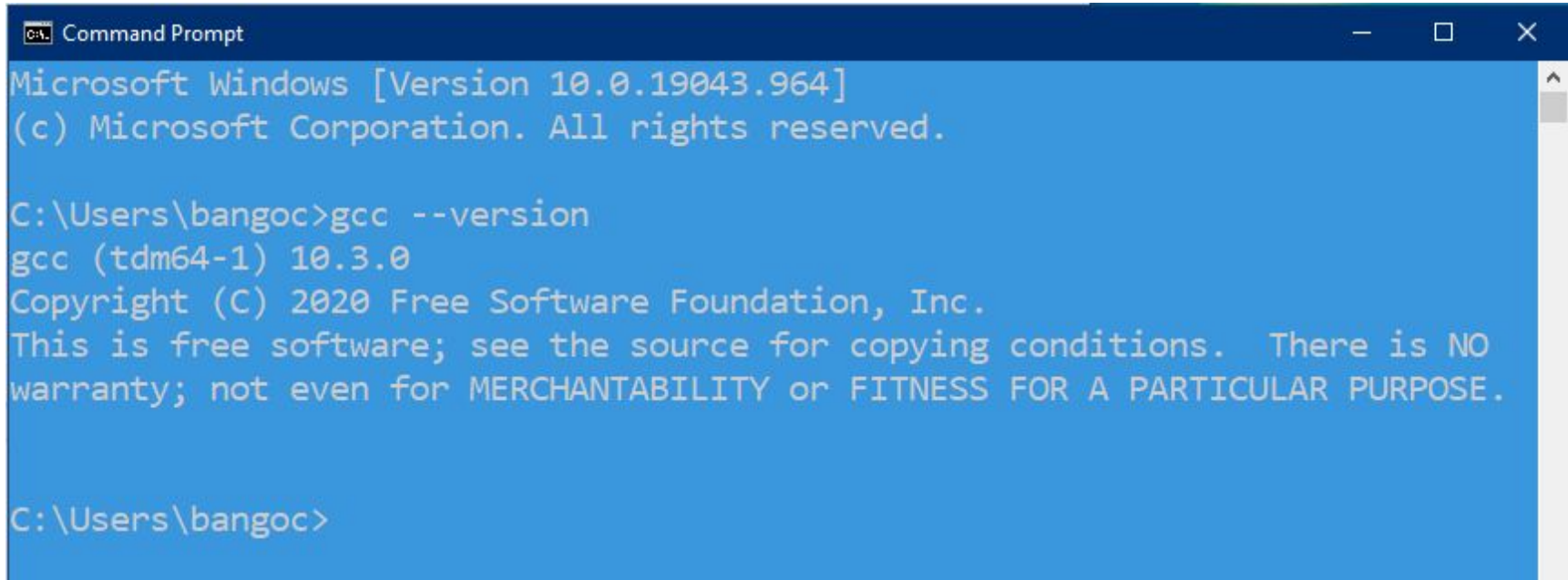
Sử dụng TDM-GCC



<https://jmeubank.github.io/tdm-gcc/>

Cài đặt GCC: Môi trường Windows₍₂₎

Kiểm tra cài đặt:

A screenshot of a Windows Command Prompt window. The title bar is dark blue with the text 'C:\> Command Prompt' and standard window controls. The main area is blue with white text. It shows the Windows version '10.0.19043.964' and copyright information for Microsoft Corporation. The user 'bangoc' has entered the command 'gcc --version'. The output shows 'gcc (tdm64-1) 10.3.0' and copyright information for the Free Software Foundation, Inc. The prompt is now 'C:\Users\bangoc>'.

```
C:\> Command Prompt
Microsoft Windows [Version 10.0.19043.964]
(c) Microsoft Corporation. All rights reserved.

C:\Users\bangoc>gcc --version
gcc (tdm64-1) 10.3.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

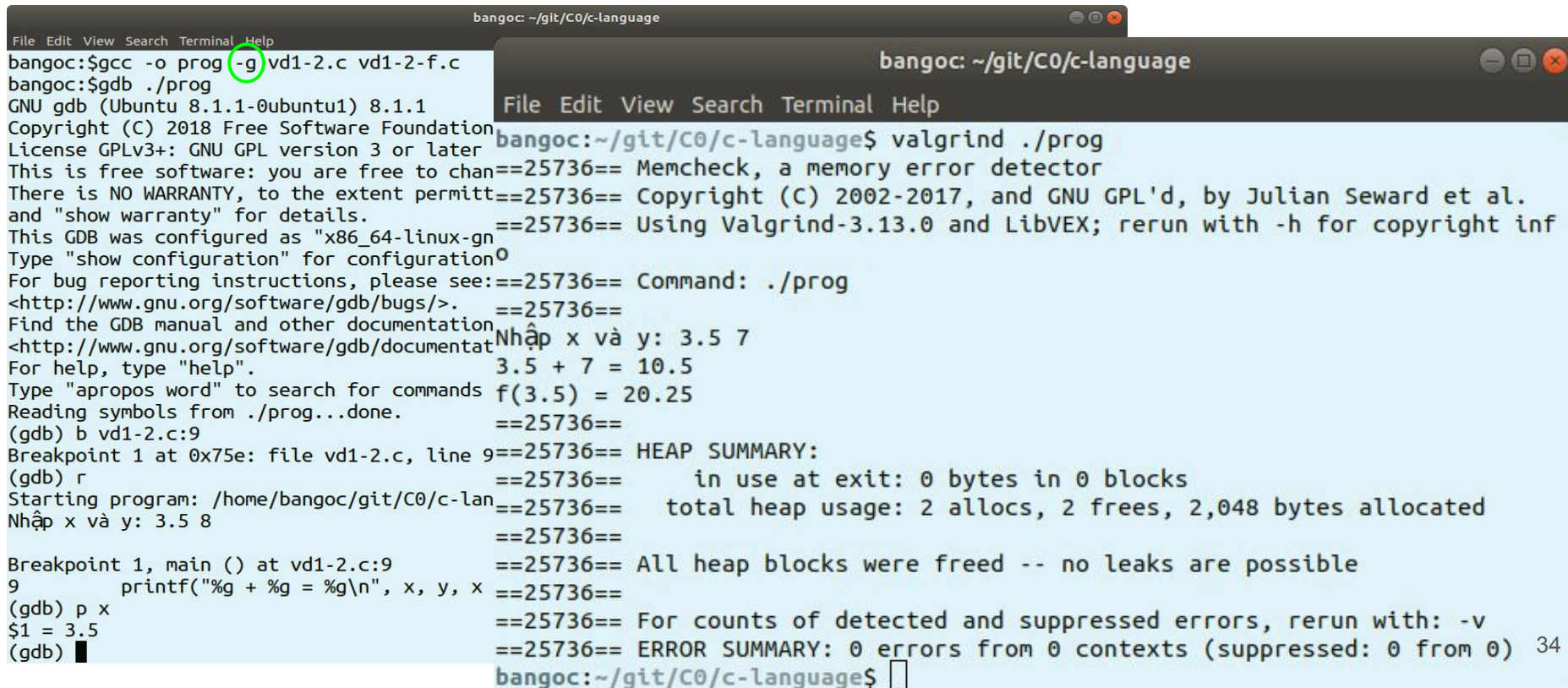
C:\Users\bangoc>
```

Trong môi trường dòng lệnh (cmd):

- + Thử chạy lệnh gcc
- + Thử biên dịch chương trình Hello World!
- + V.V..

Công cụ gỡ rối

- Hỗ trợ người lập trình tìm lỗi trong chương trình:
 - Có thể đặt điểm dừng, thực hiện lệnh từng bước, kiểm tra giá trị của biến, tìm lỗi sử dụng bộ nhớ động v.v.
 - Được sử dụng kết hợp với trình biên dịch
 - gcc: gdb, valgrind



```
bangoc: ~/git/C0/c-language
File Edit View Search Terminal Help
bangoc:$gcc -o prog -g vd1-2.c vd1-2-f.c
bangoc:$gdb ./prog
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation
License GPLv3+: GNU GPL version 3 or later
This is free software: you are free to change the code and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show configuration" for configuration 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
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands.
Reading symbols from ./prog...done.
(gdb) b vd1-2.c:9
Breakpoint 1 at 0x75e: file vd1-2.c, line 9
(gdb) r
Starting program: /home/bangoc/git/C0/c-language/./prog
Nhập x và y: 3.5 8
3.5 + 7 = 10.5
f(3.5) = 20.25
Breakpoint 1, main () at vd1-2.c:9
9      printf("%g + %g = %g\n", x, y, x + y);
(gdb) p x
$1 = 3.5
(gdb)

bangoc: ~/git/C0/c-language$ valgrind ./prog
==25736== Memcheck, a memory error detector
==25736== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==25736== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==25736== Command: ./prog
==25736==
Nhập x và y: 3.5 7
3.5 + 7 = 10.5
f(3.5) = 20.25
==25736==
==25736== HEAP SUMMARY:
==25736==    in use at exit: 0 bytes in 0 blocks
==25736==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==25736==
==25736== All heap blocks were freed -- no leaks are possible
==25736==
==25736== For counts of detected and suppressed errors, rerun with: -v
==25736== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
bangoc: ~/git/C0/c-language$
```

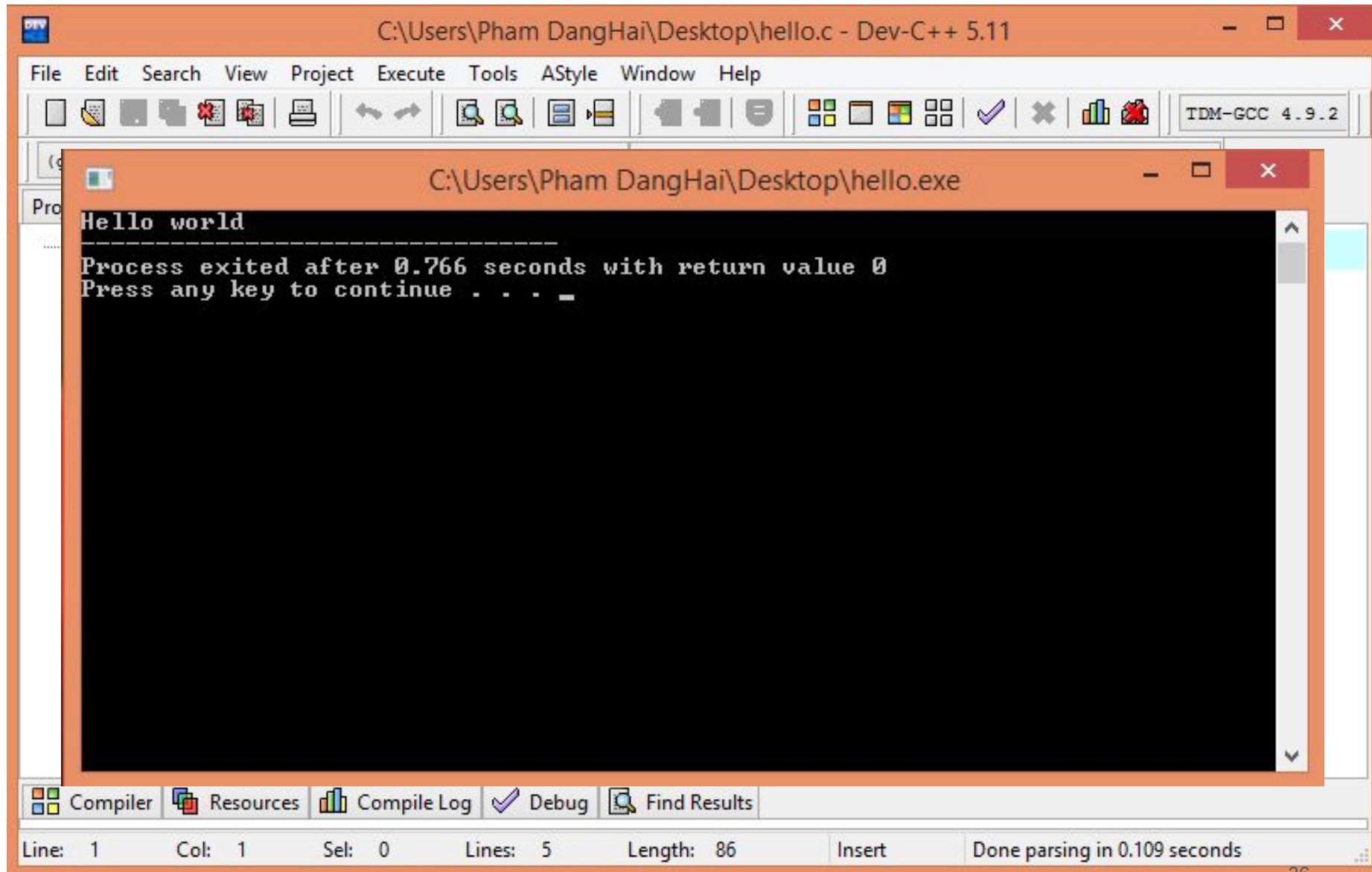
Môi trường tích hợp: Dev-C++

- Tải Dev-C++
 - (*Tìm kiếm Dev-C++*)
 - <https://www.bloodshed.net/>
 - Lựa chọn phiên bản để tải về
 - Ví dụ phiên bản mới nhất

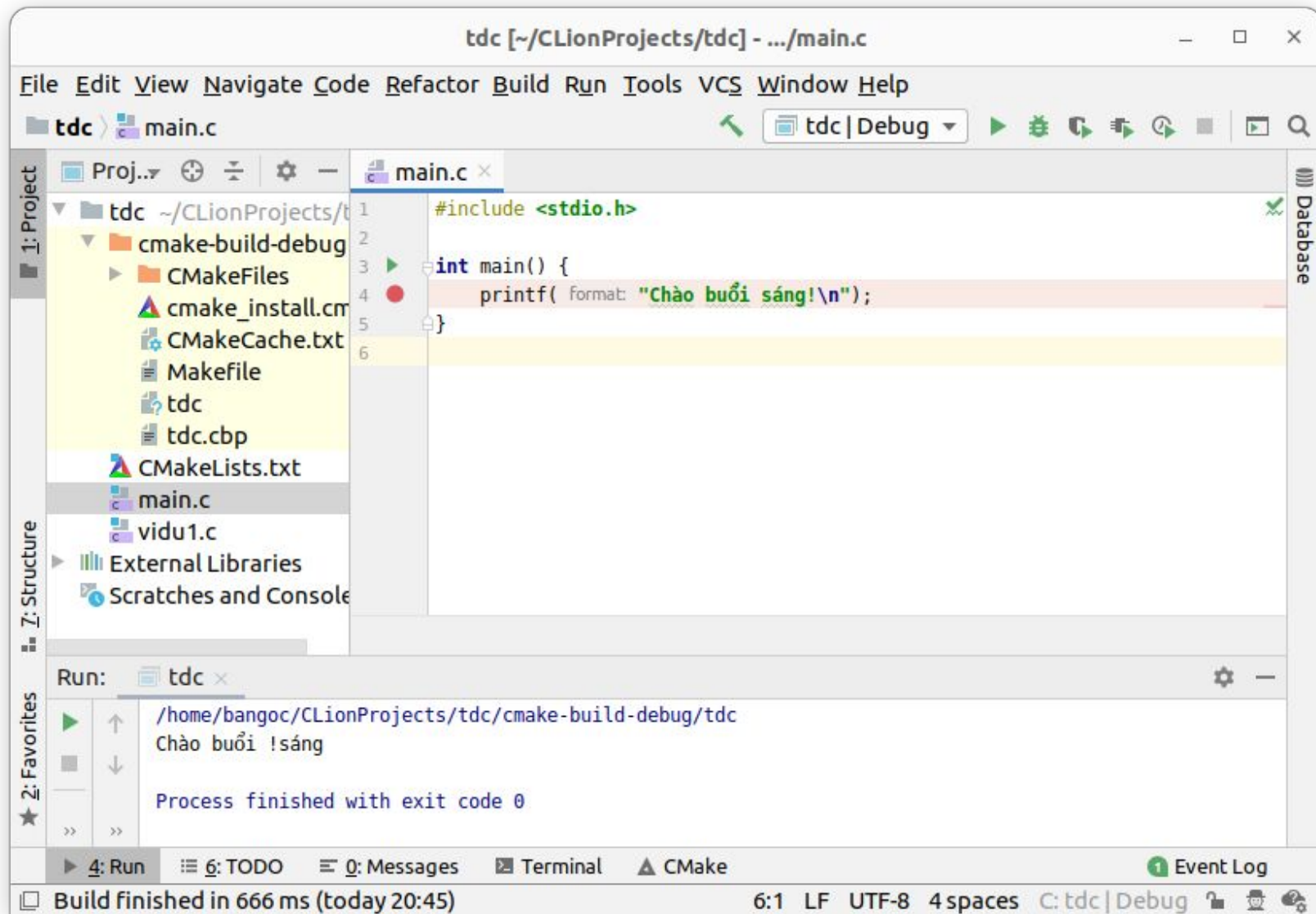


- Thực thi file tải về, cài đặt theo hướng dẫn

Màn hình giao diện DEV-C++



Môi trường tích hợp: CLion



<https://www.jetbrains.com/clion/>

Môi trường trực tuyến

Không yêu cầu cài đặt

The image displays a web-based C development environment. The top window shows a code editor with the following C code:

```
1 #include <stdio.h>
2
3 int main(void) {
4     printf("Ngôn ngữ lập trình C\n");
5 }
```

The console output shows the execution of the code:

```
> make -s
> ./main
Ngôn ngữ lập trình C
gcc (GCC) 10.3.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for c
```

The bottom window shows the GDB online Debugger interface. The code is loaded, and the console output shows the program finished with exit code 0.

OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.
IDE
My Projects
Classroom **new**
Learn Programming
Programming Questions
Sign Up
Login
About • FAQ • Blog • Terms of Use •
Contact Us • GDB Tutorial • Credits •
Privacy
© 2016 - 2022 GDB Online

Call Stack
Function File:Line
Local Variables
Variable Value
Registers
Register Value
Display Expressions
Expression Value
Enter expression to watch
Breakpoints and Watchpoints
Description

input
Ngôn ngữ lập trình C
...Program finished with exit code 0
Press ENTER to exit console.

Nếu như không có kết nối mạng?

Một số liên kết hữu ích

<https://www.sublimetext.com/>

<https://code.visualstudio.com/>

<https://www.gnu.org/software/emacs/>

<https://gcc.gnu.org/install/>

<https://jmeubank.github.io/tdm-gcc/>

<https://www.jetbrains.com/clion/>

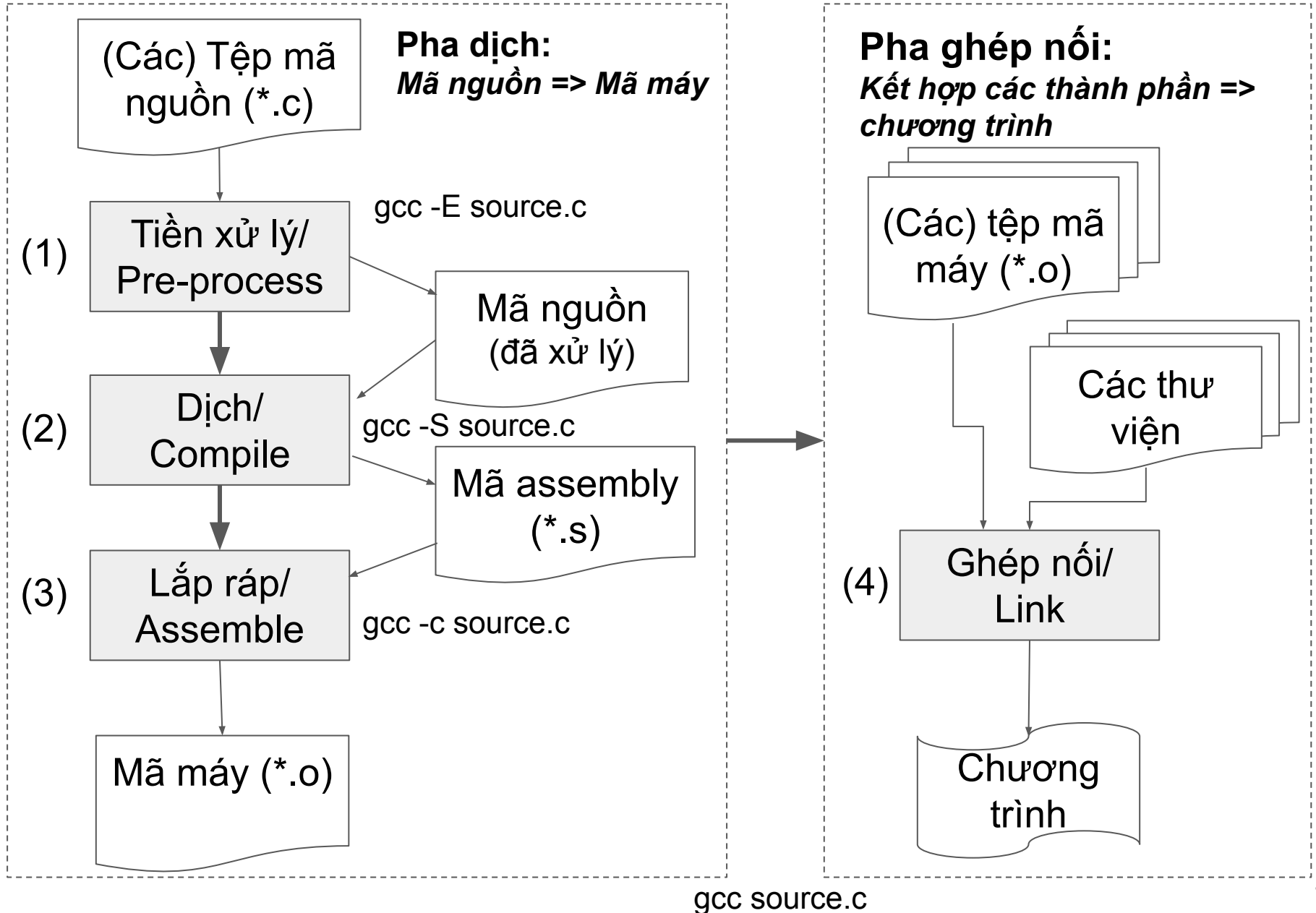
<https://replit.com/>

<https://www.onlinegdb.com/>

Nội dung

1. Khái quát về ngôn ngữ lập trình
2. Sơ lược kiến trúc von Neumann
3. Lịch sử phát triển NNLT C
4. Môi trường lập trình
5. Tiến trình biên dịch với gcc

Tiến trình biên dịch chương trình C với GCC



Ví dụ 1.4. Tiền xử lý

```
vd1-4.c x
1 #include "vd1-4.h"
2
3 #define N 100
4
5 int a[N];
6
7 #ifdef PRINT
8 for (int i = 0; i < N; ++i) {
9     printf("a[%d] = %d\n", i, a[i]);
10 }
11 #endif

vd1-4.h x
1 static inline mmax(int x, int y) {
2     return x > y? x: y;
3 }
4
```

Các lệnh tiền xử lý thường được bắt đầu với #:

#include - Chèn nội dung tệp vào mã nguồn

#define - Định nghĩa Macro

#ifdef ... #endif - Cấu trúc rẽ nhánh

...

Các tham số gcc:

-E Chỉ tiền xử lý mã nguồn

-o <tệp> Lưu kết quả biên dịch vào tệp

-D Định nghĩa Macro

```
bangoc:$gcc -E vd1-4.c
# 0 "vd1-4.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "vd1-4.c"
# 1 "vd1-4.h" 1
static inline mmax(int x, int y) {
    return x > y? x: y;
}
# 2 "vd1-4.c" 2
```

```
int a[100];
bangoc:$gcc -E vd1-4.c -DPRINT
# 0 "vd1-4.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "vd1-4.c"
# 1 "vd1-4.h" 1
static inline mmax(int x, int y) {
    return x > y? x: y;
}
# 2 "vd1-4.c" 2
```

```
int a[100];

for (int i = 0; i < 100; ++i) {
    printf("a[%d] = %d\n", i, a[i]);
}
```

Ví dụ 1.5.a. Dịch thành mã Assembly

```
1 #include <stdio.h>
2
3 #define SECRET 1984
4
5 int main() {
6     int x;
7     printf("Đoán số: ");
8     scanf("%d", &x);
9     if (x == SECRET) {
10         printf("Đúng\n");
11     } else {
12         printf("Sai\n");
13     }
14 }
```

Bạn thử đọc và đoán xem chương trình này làm gì?

gcc -o vd1-5.s -S vd1-5.c Hoặc gcc -S vd1-5.c
Các tham số biên dịch:

- S Tiền xử lý và dịch thành mã Assembly.
- o <tệp> Lưu kết quả vào tệp

```
bangoc:$gcc -S vd1-5.c
bangoc:$cat vd1-5.s
.file "vd1-5.c"
.text
.section .rodata
.LC0:
.string "\304\220o\303\241n s\341\273\221: "
.LC1:
.string "%d"
.LC2:
.string "\304\220\303\272ng"
.LC3:
.string "Sai"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
```

```
movq %fs:40, %rax
movq %rax, -8(%rbp)
xorl %eax, %eax
leaq .LC0(%rip), %rax
movq %rax, %rdi
movl $0, %eax
call printf@PLT
leaq -12(%rbp), %rax
movq %rax, %rsi
leaq .LC1(%rip), %rax
movq %rax, %rdi
movl $0, %eax
call __isoc99_scanf@PLT
movl -12(%rbp), %eax
cmpl $1984, %eax
jne .L2
leaq .LC2(%rip), %rax
movq %rax, %rdi
call puts@PLT
jmp .L3
.L2:
leaq .LC3(%rip), %rax
movq %rax, %rdi
call puts@PLT
```

```
.L2:
leaq .LC3(%rip), %rax
movq %rax, %rdi
call puts@PLT
.L3:
movl $0, %eax
movq -8(%rbp), %rdx
subq %fs:40, %rdx
je .L5
call __stack_chk_fail@PLT
.L5:
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
```

```
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
.section .note.GNU-stack,"",@progbits
.section .note.gnu.property,"a"
.align 8
.long 1f - 0f
.long 4f - 1f
.long 5
0:
.string "GNU"
1:
.align 8
.long 0xc0000002
.long 3f - 2f
2:
.long 0x3
3:
.align 8
4:
```

Ví dụ 1.5.b. Dịch thành mã máy (lắp ráp)

```
vd1-5.c x bangoc:$gcc -c vd1-5.c
1 #include <stdio.h> bangoc:$cat vd1-5.o
2
3 #define SECRET 1984
4
5 int main() {
6     int x;
7     printf("Đoán số: ")
8     scanf("%d", &x);
9     if (x == SECRET) {
10         printf("Đúng\n");
11     } else {
12         printf("Sai\n");
13     }
14 }
```

`gcc -o vd1-5.o -c vd1-5.c` Hoặc

`gcc -c vd1-5.c`

Các tham số biên dịch:

-c Tiền xử lý, dịch thành mã Assembly và lắp ráp thành mã máy.

-o <tệp> Lưu kết quả vào tệp

Tệp vd1-5.o chứa mã máy, chúng ta không đọc được như tệp văn bản.

Sử dụng objdump để xem mã máy:

`objdump --disassemble --reloc vd1-5.o`

```
bangoc:$objdump --disassemble --reloc vd1-5.o
```

```
vd1-5.o: file format elf64-x86-64
```

Disassembly of section .text:

```
0000000000000000 <main>:
 0: f3 0f 1e fa          endbr64
 4: 55                   push    %rbp
 5: 48 89 e5             mov     %rsp,%rbp
 8: 48 83 ec 10         sub     $0x10,%rsp
 c: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
13: 00 00
15: 48 89 45 f8         mov     %rax,-0x8(%rbp)
19: 31 c0               xor     %eax,%eax
1b: 48 8d 05 00 00 00 00 lea     0x0(%rip),%rax      # 22 <main+0x22>
>
22: 48 89 c7             mov     %rax,%rdi
25: b8 00 00 00 00      mov     $0x0,%eax
2a: e8 00 00 00 00      call    2f <main+0x2f>
2f: 48 8d 45 f4         lea     -0xc(%rbp),%rax
33: 48 89 c6             mov     %rax,%rsi
36: 48 8d 05 00 00 00 00 lea     0x0(%rip),%rax      # 3d <main+0x3d>
>
39: R_X86_64_PC32      .rodata+0xa
3d: 48 89 c7             mov     %rax,%rdi
40: b8 00 00 00 00      mov     $0x0,%eax
45: e8 00 00 00 00      call    4a <main+0x4a>
46: R_X86_64_PLT32     isoc99_scanf-0x4
```


Ví dụ 1.5.c. Ghép nối

```
vd1-5.c x
1 #include <stdio.h>
2
3 #define SECRET 1984
4 -----
5 int main() {
6     int x;
7     printf("Đoán số: ");
8     scanf("%d", &x);
9     if (x == SECRET) {
10         printf("Đúng\n");
11     } else {
12         printf("Sai\n");
13     }
14 }
```

Chương trình này hỏi người dùng đoán 1 số. Nếu người dùng nhập số 1984 thì được coi là đúng. Nếu ngược lại thì được coi là sai.

gcc -o prog vd1-5.o Hoặc

gcc -o prog vd1-5.c

Các tham số biên dịch:

-o <tệp> Lưu kết quả vào tệp

prog là tên chương trình thực thi thu được.

Sử dụng `objdump` để xem mã máy:

`objdump --disassemble --reloc prog`

```
bangoc:$gcc -o prog vd1-5.c
bangoc:$./prog
Đoán số: 1984
Đúng
bangoc:$objdump --disassemble --reloc prog
```

prog: file format elf64-x86-64

Disassembly of section .init:

```
0000000000001000 <_init>:
1000: f3 0f 1e fa      endbr64
1004: 48 83 ec 08      sub    $0x8,%rsp
1008: 48 8b 05 d9 2f 00 00 mov    0x2fd9(%rip),%rax
# 3fe8 <__gmon_start__@Base>
100f: 48 85 c0          test   %rax,%rax
1012: 74 02            je     1016 <_init+0x16>
1014: ff d0            call   %rax
1016: 48 83 c4 08      add    $0x8,%rsp
101a: c3              ret
```

Disassembly of section .plt:

```
0000000000001020 <.plt>:
1020: ff 35 82 2f 00 00 push   0x2f82(%rip) # 3f
a8 <_GLOBAL_OFFSET_TABLE_+0x8>
1026: f2 ff 25 83 2f 00 00 bnd jmp *0x2f83(%rip) #
3fb0 <_GLOBAL_OFFSET_TABLE_+0x10>
102d: 0f 1f 00          nopl   (%rax)
1030: f3 0f 1e fa      endbr64
1034: 68 00 00 00 00 00 push   $0x0
1039: f2 e9 e1 ff ff ff bnd jmp 1020 <_init+0x20>
103f: 90              nop
```

Hàm printf được định nghĩa trong thư viện

```
00000000000010a0 <printf@plt>:
10a0: f3 0f 1e fa      endbr64
10a4: f2 ff 25 1d 2f 00 00 bnd jmp *0x2f1d(%rip) # 3fc8 <printf@GLIBC_2.2.5>
10ab: 0f 1f 44 00 00    nopl   0x0(%rax,%rax,1)
```

