

# Nghiên cứu cài đặt chương trình dịch cho ngôn ngữ lập trình Barebone

---

## Mục tiêu

---

1. Trình bày ngôn ngữ lập trình Barebone.
2. Xây dựng chương trình cho phép biên dịch chương trình viết bằng Barebone để có thể thực thi được cho các phép toán cộng trừ nhân chia hai số nguyên, phép toán so sánh giữa hai số nguyên.
3. Giao diện chương trình biên dịch ở dạng đồ họa.

## Nội dung

---

### Ngôn ngữ lập trình Barebone

#### Đặc điểm

Barebone (BB) là một ngôn ngữ lập trình thủ tục với các đặc điểm :

- Không có chương trình con như hàm hay thủ tục.
  - Mã nguồn chỉ bao gồm các *câu lệnh* (statement). Mỗi câu lệnh phải kết thúc bằng dấu chấm phẩy `;`.
  - Kiểu dữ liệu duy nhất là **số nguyên không âm**.
  - Các biến đều là toàn cục (global). Khi được nhắc đến lần đầu trong một câu lệnh, biến được gán bằng không trước khi câu lệnh đó được thực thi.
  - Cấu trúc lặp có thể nằm lồng trong cấu trúc lặp khác.
  - Không có đầu ra và đầu vào chuẩn.
  - *Không* phân biệt chữ hoa và thường (case-insensitive). Chẳng hạn, `x` và `incr` đôi một tương đương với `x` và `INCR`.
  - Các từ khóa là `clear`, `decr`, `do`, `end`, `incr`, `not`, `while`.
  - Tên biến chỉ được bao gồm các chữ cái hoặc chữ số ASCII hoặc dấu underscore `_`. Tên biến không được bắt đầu với chữ số. Tên biến không phân biệt chữ hoa và thường. Tên biến không được giống một từ khóa nào.
-

BB sử dụng ba câu lệnh

- `clear X;` gán biến mang tên `x` bằng 0.
- `incr X;` tăng giá trị của biến `x` lên một.
- `decr X;` giảm giá trị của biến `x` đi một nếu `x` lớn hơn không.

và một cấu trúc lặp

```
while X not 0 do ... end;
```

được thực thi như sau :

1. Nếu `x` bằng không thì thoát vòng lặp, nếu không thì tiếp tục.
2. Thực hiện tuần tự những câu lệnh ở vị trí dấu chấm lửng `...` cho đến câu lệnh `end;` và quay lại bước 1.

---

Mỗi dấu cách đều có thể thay bằng ký tự xuống dòng hay *tab* (bốn space) để dễ đọc, chẳng hạn

`while X not 0 do decr X end;` thì cũng tương đương với

```
while X not 0 do
    decr X;
end;
```

và

```
while X not 0 do
decr X;
end;
```

Khuyến khích viết thành các dòng để code dễ đọc hơn.

Ví dụ về một đoạn mã nguồn hợp lệ :

```
clear Y;
clear T;
while X not 0 do
    incr Y;
    incr T;
    decr X;
end;
while T not 0 do
    incr X;
    decr T;
end;
```

## Nhận xét

Các file được nhắc đến trong mục này có thể tìm thấy ở `app/src/main/resources/snippets/`.

---

Một số phép toán cơ bản :

- Phép gán `Y := X` ở file `assign.txt`.
- Cấu trúc rẽ nhánh `if X != 0 then X := 0 else X := 1` ở file `toggle.txt`, có thể gọi là `invert X`.
- Phép cộng `Z := X + Y` ở file `add.txt`.
- Phép trừ `Z := X - Y` ở file `subtract.txt`.
- Phép nhân `Z := X * Y` ở file `multiply.txt`.
- Phép chia nguyên `Z := floor(X / Y)` ở file `int_divide.txt`.
- Phép so sánh `if X < Y then Z := 0 else Z := 1` ở file `less_than.txt`, có thể viết lại bằng pseudo-code như sau :

```

Z := 0
T_X := X
T_Y := Y
while T_Y != 0 do
    Z := Z + 1
    T_Y := T_Y - 1
while T_X != 0 do
    if Z > 0 then Z := Z - 1
    T_X := T_X - 1
T_X := T_X + 1
while Z != 0 do
    T_X := 0
    Z := 0
while T_X != 0 do
    Z := Z + 1
    T_X := T_X - 1

```

- Các phép so sánh khác có thể tìm thấy ở `not_less_than.txt`, `greater_than.txt`, `not_greater_than.txt`, `equals.txt`, `not_equals.txt`.

Ta nhận thấy có thể thay thế `clear X;` bằng đoạn mã sau

```

while X not 0 do
    decr X;
end;

```

Có thể biểu diễn số âm trong BB bằng cách dùng một biến dấu đi chung với số không âm. Biến dấu mang giá trị 0 nếu  $X \geq 0$  và mang giá trị khác 0 trong trường hợp ngược lại.

Ví dụ, số nguyên  $X$  có thể được biểu diễn bởi

- `x` mang giá trị tuyệt đối của  $X$  và
- `X_NEG` là biến dấu.

Từ đó ta có thể đổi dấu của  $X$  bằng đoạn code tương đương với

```

if X_NEG != 0 then X_NEG := 0 else X_NEG := 1

```

## Chương trình biên dịch ở dạng đồ họa

Một chương trình giả lập quá trình biên dịch mã nguồn BB, sử dụng giao diện đồ họa, đã được xây dựng bằng Java với Gradle, được gọi là **Barebonesim** (Barebone + Simulator).

## Yêu cầu

Cần Oracle JDK từ 1.8 trở lên. Kiểm tra phiên bản của Java bằng dòng lệnh

```
java -version
```

Nếu không có sẵn Java, hãy tải về theo hướng dẫn tại [java.com](https://java.com) và chạy lại lệnh trên để kiểm tra phiên bản.

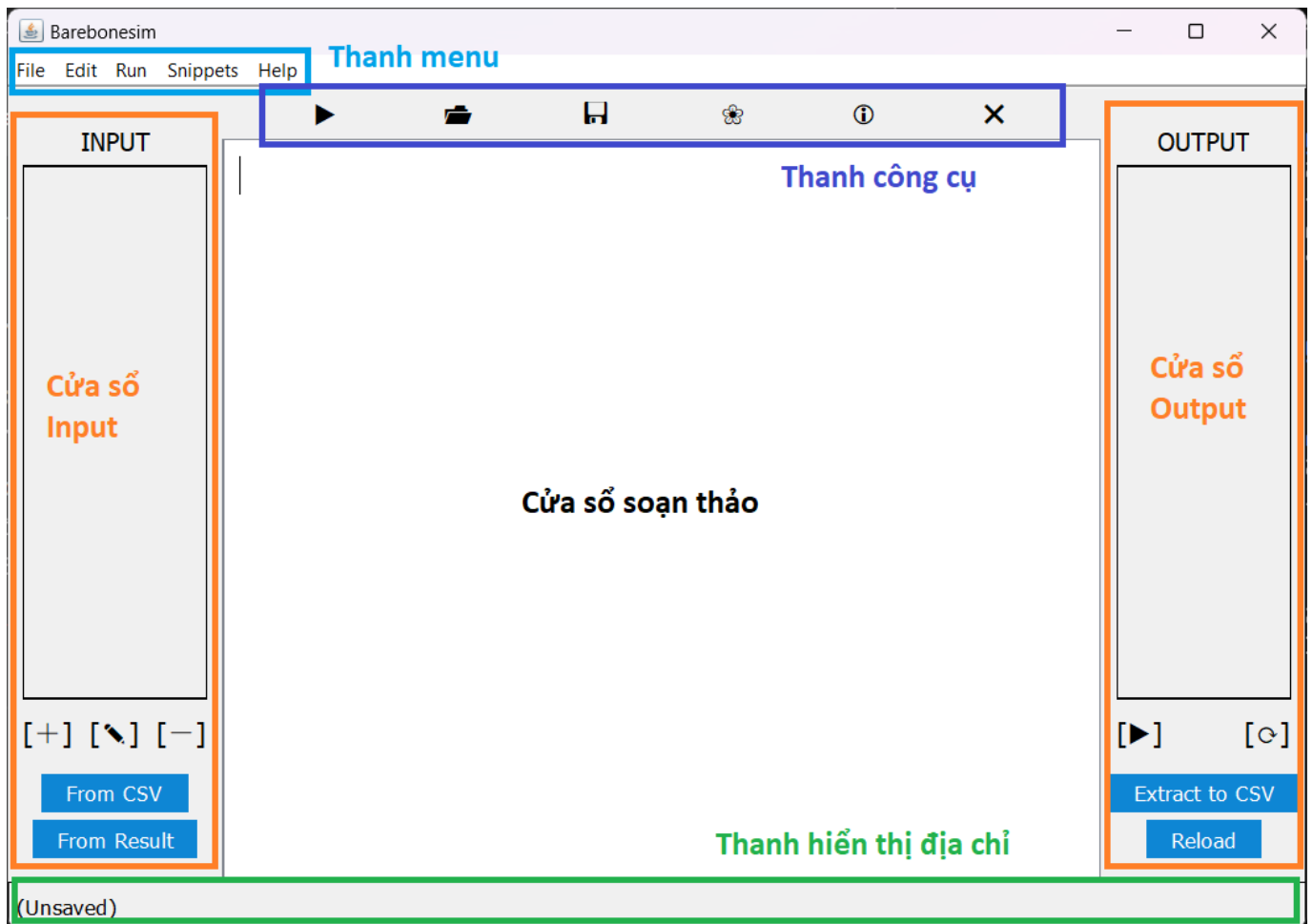
## Hướng dẫn sử dụng

### Khởi động

Ở thư mục gốc (chứa file `gradlew`), chạy dòng lệnh sau

```
./gradlew run
```



Cửa sổ sau sẽ hiện ra với các thành phần như hình dưới :




trong đó

- Thanh menu bao gồm các lệnh cơ bản để thao tác với file và code.
- Thanh công cụ bao gồm các nút, từ trái sang phải, là chạy code, mở file, lưu file, kiểm tra lỗi đồng thời "làm đẹp" code, tạo snippet (các đoạn code có sẵn) và thoát file.
- Cửa sổ soạn thảo soạn thảo mã nguồn (code).
- Cửa sổ Input thao tác với các biến đầu vào.
- Cửa sổ Output thao tác với kết quả sau khi thực thi code.
- Thanh hiển thị địa chỉ hiển thị địa chỉ tuyệt đối của file code đang mở.

### Soạn thảo

Có thể viết code trực tiếp vào cửa sổ soạn thảo, hoặc mở file code bằng cách nhấn vào  trên thanh công cụ và chọn file cần tìm. Sau khi soạn thảo, lưu bằng cách nhấn vào  trên thanh công cụ và nhập tên rồi lưu file.




Trong quá trình soạn thảo, có thể kiểm tra lỗi và "làm đẹp" code bằng cách nhấn vào  (biểu tượng bông hoa) trên thanh công cụ.

### Sử dụng code có sẵn (snippet)

Để tiết kiệm thời gian soạn thảo, có thể sử dụng các đoạn code có sẵn giúp thực hiện các phép tính cơ bản (cộng, trừ, nhân, chia, gán giá trị, so sánh, chuyển từ không thành khác không và ngược lại, đổi dấu).

### Các biến đầu vào

Sử dụng cửa sổ Input (bên trái cửa sổ soạn thảo), ta có thể gán giá trị ban đầu cho một số biến.

- Để thêm biến, chọn . Trong cửa sổ vừa hiện ra, nhập tên biến và giá trị ban đầu của nó rồi chọn OK.
- Để thay đổi giá trị của biến, chọn . Chọn biến cần chỉnh sửa, chọn OK rồi nhập giá trị mới, chọn OK.
- Để xóa một biến khỏi danh sách, chọn . Chọn biến cần xóa và chọn OK.


Đây là giá trị ban đầu của các biến trước khi code được thực thi.

Ngoài ra, các biến này có thể được thêm từ file CSV. Nhấn nút  và chọn file cần tìm. File CSV tuân theo [RFC 4180](#) và *không chứa dòng header*.

Một ví dụ về file CSV hợp lệ :

```
X,12
Y,20
Z,0
```

### Thực thi và xem kết quả

Sau khi đã có code và danh sách biến với giá trị ban đầu, có thể thực thi (chạy code) bằng cách chọn  trên thanh công cụ.

Nếu code có lỗi (tên biến không phù hợp, vòng lặp vô hạn v.v.) hoặc chạy quá thời gian giới hạn (mặc định là 1000 ms), chương trình sẽ báo lỗi và ngừng chạy.

Nếu chạy thành công, các giá trị *sau khi thực thi* của biến sẽ hiển thị trên cửa sổ Output (bên phải cửa sổ soạn thảo).

Lưu kết quả vào file CSV bằng nút **Extract to CSV** trên cửa sổ Output.

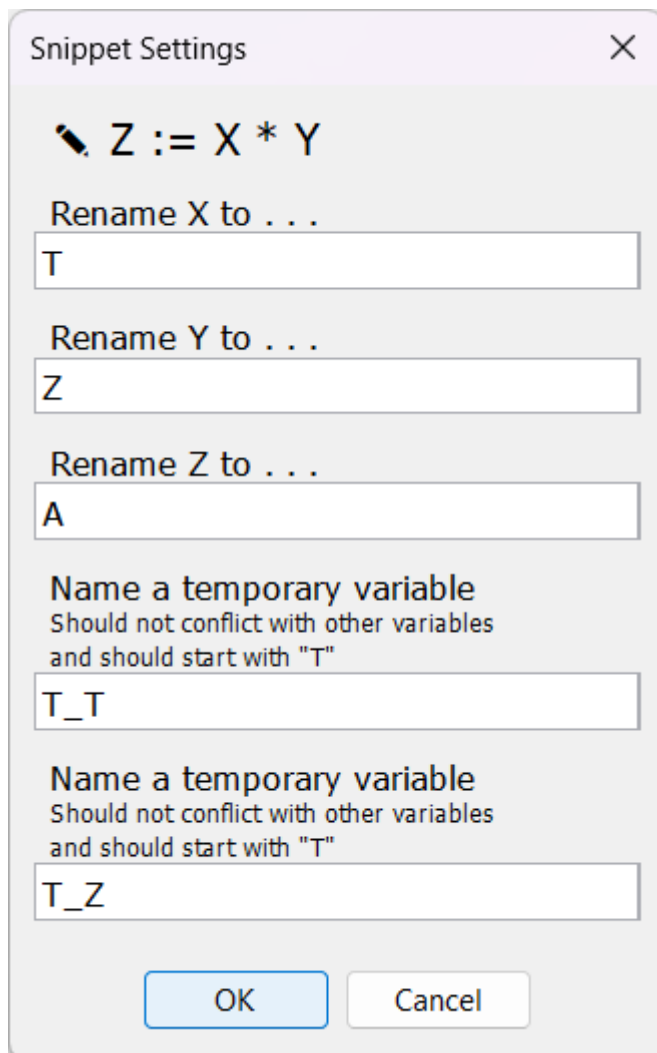
Đem các giá trị của cửa sổ Output sang cửa sổ Input bằng nút **From Result** trên cửa sổ Input.

Nếu code và giá trị ban đầu của các biến (Input) có thay đổi sau khi đã thực thi, có thể cập nhật giá trị Output mới bằng cách chạy lại.


#### Ví dụ

Ví dụ, để tạo đoạn code thực hiện `A := Z * T` với giá trị của `Z` và `T` được cho trước, ta làm như sau :

1. Trên thanh menu, chọn File → New File rồi tạo file mới.
2. Sau khi tạo và mở file mới, trên thanh menu, chọn Snippets → Snippets.
3. Trên cửa sổ "Snippets" mới hiện ra, chọn `Z := X * Y` rồi chọn OK.
4. Trên cửa sổ "Snippet Settings", thay đổi tên các biến như bên dưới rồi chọn OK.



Snippet Settings

 `Z := X * Y`

Rename X to ...

T

Rename Y to ...

Z

Rename Z to ...

A

Name a temporary variable  
Should not conflict with other variables  
and should start with "T"

T\_T

Name a temporary variable  
Should not conflict with other variables  
and should start with "T"

T\_Z

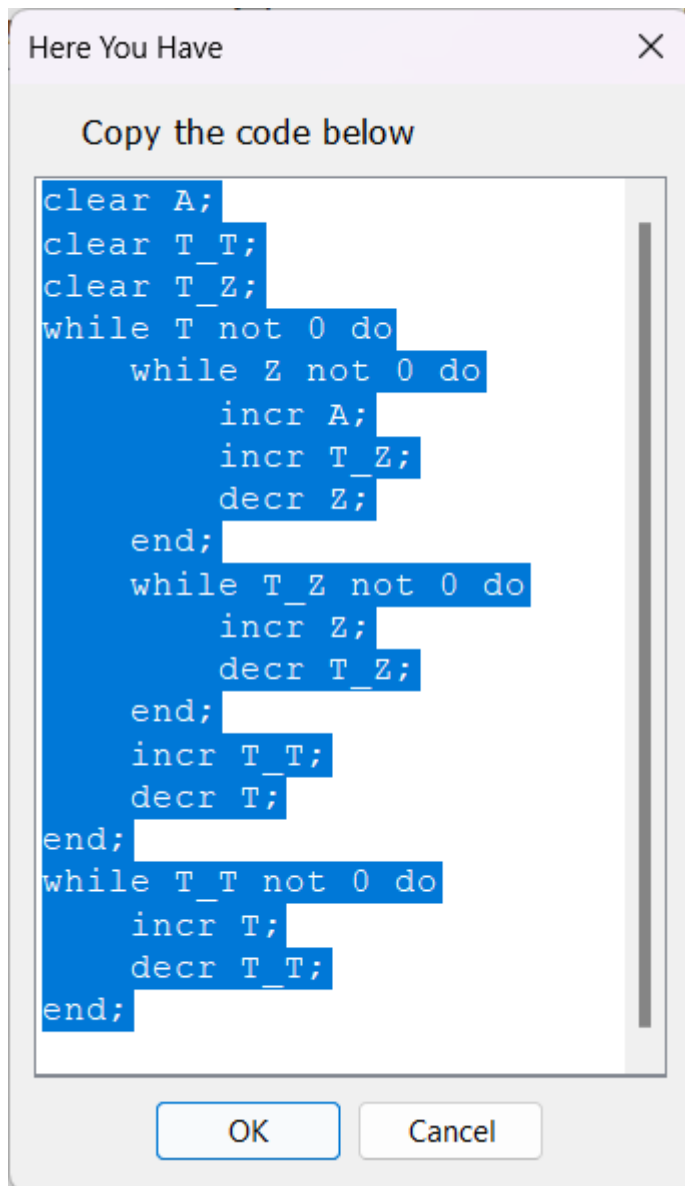
OK Cancel



Chú ý đặt tên các biến tạm thời (temporary variables) sao cho không trùng với biến nào đang dùng. Các biến tạm thời trong một phép toán không được trùng tên. Tên biến tạm thời nên bắt đầu với "T" hoặc "T\_".

Chúng có thể được tái sử dụng nhiều lần. Sau mỗi phép tính cung cấp bởi snippet, các biến tạm đều mang giá trị không.

- Trong cửa sổ mới hiện ra, tôi chọn tất cả đoạn code trong khung và nhấn tổ hợp Ctrl + C để sao chép, rồi chọn OK.



- Vào cửa sổ soạn thảo, rồi nhấn tổ hợp Ctrl + V để dán đoạn code vừa sao chép.
- Bấm nút `[+]` trên cửa sổ Input (bên trái cửa sổ soạn thảo) rồi điền tên và giá trị ban đầu của biến `Z` (ví dụ, bằng 12).

Add New Input

×

?

Variable Name

Z

Variable names **MUST** follow these rules

1. A name cannot be empty.
2. A name starts with an character from A to Z, (uppercase or lowercase), or an underscore.
3. A name contains only characters from A to Z (uppercase or lowercase), or digits, or underscore.
4. If a name changes to lowercase, it will not be the same as any keyword.

These are all keywords: clear, decr, do, end, incr, not, while.

A name **MUST** have a length less than 32 characters.

Initial Value

12

Each variable can only hold an non-negative integer value.

OK

Cancel

Làm tương tự với biến `T` (ví dụ, bằng 15).

Add New Input

×

?

Variable Name

T

Variable names **MUST** follow these rules

1. A name cannot be empty.
2. A name starts with an character from A to Z, (uppercase or lowercase), or an underscore.
3. A name contains only characters from A to Z (uppercase or lowercase), or digits, or underscore.
4. If a name changes to lowercase, it will not be the same as any keyword.

These are all keywords: clear, decr, do, end, incr, not, while.

A name **MUST** have a length less than 32 characters.

Initial Value

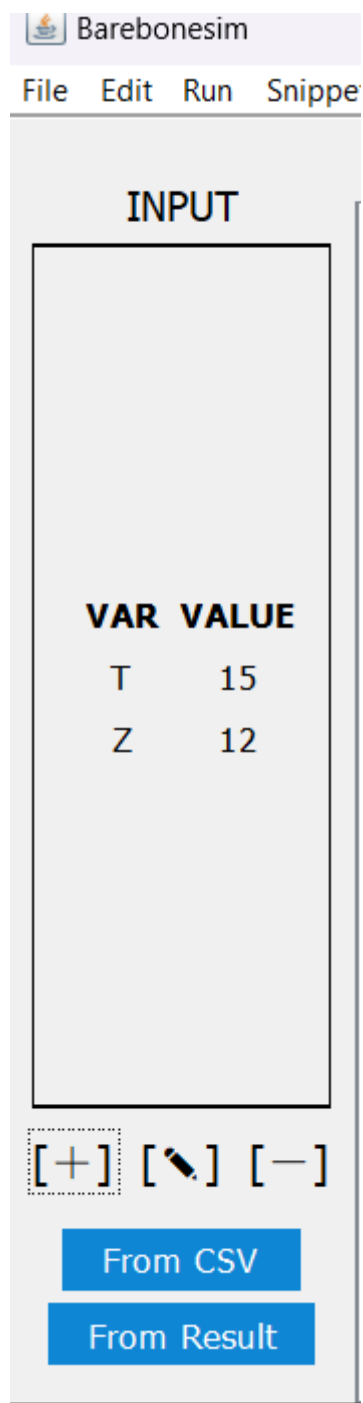
15


Each variable can only hold an non-negative integer value.

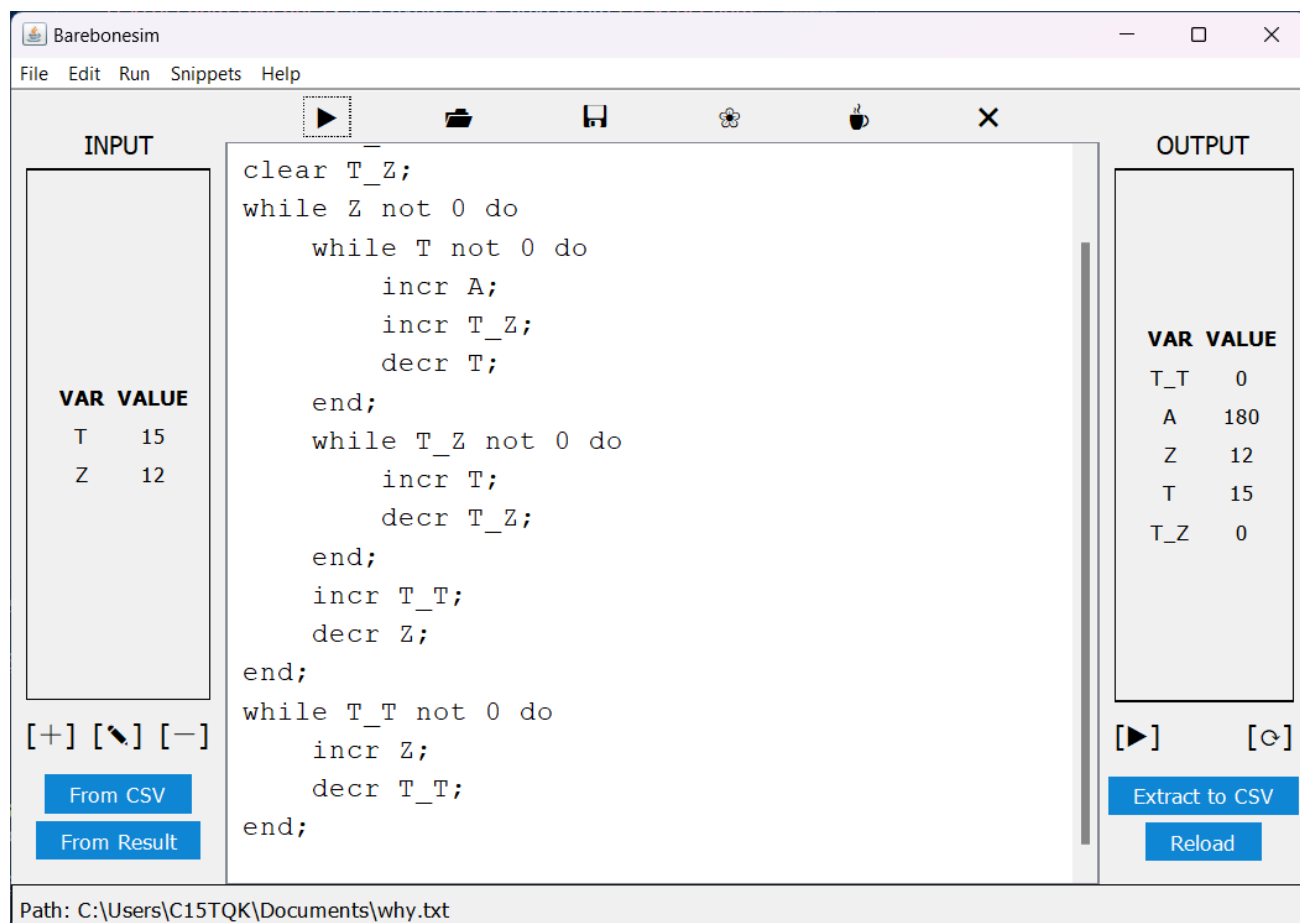
OK

Cancel

Cửa sổ Input sẽ hiển thị như sau :



8. Bấm nút  trên thanh công cụ để thực thi đoạn code. Có thể thấy kết quả gần giống như hình dưới :



Nhìn vào cửa sổ Output (bên phải cửa sổ soạn thảo), ta thấy **A** bằng 180 là kết quả cần tìm.

## Đặc điểm

Barebonesim dịch BB bằng cách nào ?

Code được biên dịch và thực thi theo các bước sau :

1. Các giá trị ban đầu của biến (trên cửa sổ Input, bên trái cửa sổ soạn thảo) được lưu vào một đối tượng *VariableContainer*, có tác dụng lưu trữ tất cả biến theo tên và giá trị của chúng.
2. *Parser* chuyển code thành một danh sách các *cây cú pháp trừu tượng* (Abstract Syntax Tree), mỗi cây biểu diễn một câu lệnh (statement). Mỗi vòng lặp (while) cũng là một câu lệnh.
3. Trong quá trình *parse* của Parser, nếu phát hiện ra lỗi cú pháp, Parser sẽ ngừng lại và báo lỗi.
4. Danh sách các cây cú pháp trừu tượng được lưu trong *Compiler* và được thực thi tuần tự trong một tiểu trình khác. Các thay đổi về giá trị biến sẽ được áp dụng lên đối tượng *VariableContainer* ở bước đầu tiên.

5. Nếu Compiler thực thi code quá thời gian giới hạn (timeout), nó sẽ ngừng lại và báo lỗi. Thời gian giới hạn là 1000 ms theo mặc định và có thể tùy chỉnh.
6. Nếu thực thi thành công, các giá trị cuối cùng lưu trong VariableContainer sẽ được hiển thị trên cửa sổ Output (bên phải cửa sổ soạn thảo).

Ta có thể nhận thấy :

- Tất cả các bước được thực hiện trong thời gian chạy (runtime) của Barebonesim.
- **Không có file executable hay bytecode nào được tạo ra**, thay vào đó là danh sách các kết quả.

**Barebonesim có hỗ trợ nhập và xuất giá trị không ?**

Theo khái niệm, BB chỉ hỗ trợ các câu lệnh `incr`, `decr`, `clear` và vòng lặp `while` với điều kiện duy nhất là biến điều kiện khác 0, ngoài ra không có câu lệnh nào khác.

Một số compiler cho BB hỗ trợ thêm `read` và `print` để nhập và xuất giá trị ; tuy nhiên để nhấn mạnh tính đơn giản của BB, Barebonesim không hỗ trợ đầu vào hay đầu ra chuẩn. Thay vào đó, có thể xem giá trị của các biến sau khi thực thi ở cửa sổ Output.

**Barebonesim có hỗ trợ comment trong code không ?**

Barebonesim chưa thể dịch các đoạn code có chứa comment.