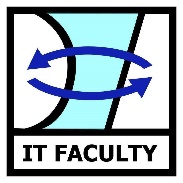
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN LẬP TRÌNH TÍNH TOÁN**

**Xây dựng ứng dụng chuyển đổi giữa các hệ đếm.**

Người hướng dẫn**: GVC.ThS. Trần Hồ Thủy Tiên**

Sinh viên thực hiện**:**

**Siphanthong Xanakone LỚP: 22T\_DT5 NHÓM: 14**

**Souvannaphoum Athit LỚP: 22T\_DT5 NHÓM: 14**

**Xaysongkham Phoutthasinh LỚP: 22T\_DT5 NHÓM: 14**

**Đà Nẵng, 06/2023**

MỤC LỤC

[MỤC LỤC i](#_Toc138089051)

[DANH MỤC HÌNH VẼ ii](#_Toc138089052)

[MỞ ĐẦU i](#_Toc138089053)

[1. TỔNG QUAN ĐỀ TÀI 1](#_Toc138089054)

[1.1. Ý tưởng 1](#_Toc138089055)

[1.2. Ngôn ngữ cài đặt 2](#_Toc138089056)

[2. CƠ SỞ LÝ THUYẾT 2](#_Toc138089057)

[3. TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN 5](#_Toc138089058)

[3.1. Phát biểu bài toán 5](#_Toc138089059)

[3.2. Cấu trúc dữ liệu 5](#_Toc138089060)

[3.3. Thuật toán 6](#_Toc138089061)

[4. CHƯƠNG TRÌNH VÀ KẾT QUẢ 26](#_Toc138089062)

[4.1. Tổ chức chương trình 26](#_Toc138089063)

[4.2. Kết quả 27](#_Toc138089064)

[4.2.1. Giao diện chính của chương trình 27](#_Toc138089065)

[4.2.2. Kết quả thực thi của chương trình 28](#_Toc138089066)

[4.2.3. Nhận xét đánh giá 29](#_Toc138089067)

[5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 30](#_Toc138089068)

[5.1. Kết luận 30](#_Toc138089069)

[5.2. Hướng phát triển 30](#_Toc138089070)

[TÀI LIỆU THAM KHẢO 31](#_Toc138089071)

[PHỤ LỤC 32](#_Toc138089072)

DANH MỤC HÌNH VẼ

[Hình 1: Flowchart Hàm main. 7](file:///D:\Documents\Bachelor%20at%20Vietname\Năm%201\Năm%201%20kỳ%202\PBL1\BAO%20CAO%20DO%20AN%20-%204.docx#_Toc138089073)

[Hình 2: Flowchart Hàm menu. 9](#_Toc138089074)

[Hình 3: Flowchart Hàm chuyển đổi từ hệ thập phân sang nhị phân. 11](file:///D:\Documents\Bachelor%20at%20Vietname\Năm%201\Năm%201%20kỳ%202\PBL1\BAO%20CAO%20DO%20AN%20-%204.docx#_Toc138089075)

[Hình 4: Flowchart Hàm chuyển đổi từ hệ thập phân sang bát phân. 13](file:///D:\Documents\Bachelor%20at%20Vietname\Năm%201\Năm%201%20kỳ%202\PBL1\BAO%20CAO%20DO%20AN%20-%204.docx#_Toc138089076)

[Hình 5: Flowchart Hàm chuyển đổi từ hệ thập phân sang thập lục phân. 16](#_Toc138089077)

[Hình 6 : Flowchart Hàm chuyển đổi từ nhị phân sang hệ thập phân. 18](#_Toc138089078)

[Hình 7: Flowchart Hàm chuyển đổi từ bát phân sang hệ thập phân. 20](#_Toc138089079)

[Hình 8: Flowchart Hàm chuyển đổi từ thập lục phân sang hệ thập phân. 22](file:///D:\Documents\Bachelor%20at%20Vietname\Năm%201\Năm%201%20kỳ%202\PBL1\BAO%20CAO%20DO%20AN%20-%204.docx#_Toc138089080)

[Hình 9: Flowchart Hàm chuyển đổi từ nhi phân sang thập lúc phân. 23](#_Toc138089081)

[Hình 10: Flowchart Hàm chuyển đổi từ nhi phân sang bát phân. 24](#_Toc138089082)

[Hình 11: Flowchart Hàm chuyển đổi từ bát phân sang nhi phân. 25](#_Toc138089083)

[Hình 12: Flowchart Hàm chuyển đổi từ thập lúc phân sang nhi phân. 26](#_Toc138089084)

[Hình 13: Giao diện chính của chương trình. 28](file:///D:\Documents\Bachelor%20at%20Vietname\Năm%201\Năm%201%20kỳ%202\PBL1\BAO%20CAO%20DO%20AN%20-%204.docx#_Toc138089085)

[Hình 14: Kết của sau chương trình đã chuyển đổi hệ của số. 28](file:///D:\Documents\Bachelor%20at%20Vietname\Năm%201\Năm%201%20kỳ%202\PBL1\BAO%20CAO%20DO%20AN%20-%204.docx#_Toc138089086)

MỞ ĐẦU

Trong khoa học máy tính và toán học, hệ đếm là một phương pháp để biểu diễn các số và tính toán. Hệ thống đếm phổ biến nhất là hệ thập phân, trong đó các số được biểu diễn bằng cách sử dụng 10 ký hiệu số từ 0 đến 9. Tuy nhiên, trong một số trường hợp khác, chúng ta có thể cần phải sử dụng các hệ đếm khác, ví dụ như hệ nhị phân (2 ký hiệu số 0 và 1), hệ bát phân (8 ký hiệu số từ 0 đến 7), hoặc hệ thập lục phân (16 ký hiệu số từ 0 đến 9 và các chữ cái A đến F).

Để chuyển đổi các số giữa các hệ đếm, chúng ta có thể sử dụng các công thức toán học đơn giản. Tuy nhiên, việc tính toán thủ công có thể mất nhiều thời gian và dễ dẫn đến lỗi. Vì vậy, việc xây dựng một ứng dụng chuyển đổi giữa các hệ đếm có thể giúp cho người dùng tiết kiệm thời gian và đảm bảo tính chính xác của kết quả.

Ứng dụng chuyển đổi giữa các hệ đếm có thể được thiết kế để cho phép người dùng chuyển đổi các số nguyên dương hoặc số thực. Người dùng chỉ cần nhập số cần chuyển đổi và lựa chọn hệ đếm muốn chuyển đổi thành. Ứng dụng sẽ thực hiện các phép tính toán tương ứng và cho ra kết quả đúng theo yêu cầu của người dùng.

Tóm lại, xây dựng một ứng dụng chuyển đổi giữa các hệ đếm là một ứng dụng hữu ích giúp cho người dùng tiết kiệm thời gian và đảm bảo tính chính xác của các phép tính toán.

1. TỔNG QUAN ĐỀ TÀI

Đề tài: XÂY DỰNG ỨNG DỤNG CHUYỂN ĐỔI GIỮA CÁC HỆ ĐẾM

* Mô tả chung đề tài:
* Ứng dụng nhằm cung cấp chức năng chuyển đổi giữa các hệ đếm số, ví dụ như hệ thập phân, hệ nhị phân, hệ thập lục phân, v.v.
* Ứng dụng sẽ cung cấp các chức năng chuyển đổi số từ hệ này sang hệ khác, cũng như hiển thị kết quả chuyển đổi cho người dùng.
* Ứng dụng cho phép người dùng sẽ nhập một số nguyên dương và âm hay số thập phân dương và âm trong một hệ đếm cụ thể và ứng dụng sẽ chuyển đổi số đó sang các hệ đếm khác theo yêu cầu.

Yêu cầu: Viết chương trình xây dựng ứng dụng chuyển đổi giữa các hệ đếm và xây dựng một giao diện đơn giản để người dùng có thể chọn loại chuyển đổi mong muốn.

* 1. Ý tưởng

Xây dựng một ứng dụng chuyển đổi giữa các hệ đếm số để người dùng có thể dễ dàng chuyển đổi giữa các hệ đếm số khác nhau, bao gồm hệ thập phân, hệ nhị phân, hệ thập lục phân và hệ thập phân tám phân.

* Ý tưởng mở rộng chương trình:

1. Thiết kế giao diện người dùng:

* Tạo giao diện đơn giản và thân thiện với người dùng.
* Hiển thị kết quả chuyển đổi một cách rõ ràng và dễ hiểu.

1. Xây dựng chức năng chuyển đổi:

* Xây dựng các hàm hoặc phương thức để chuyển đổi giữa các hệ đếm số. Ví dụ: từ hệ thập phân sang hệ nhị phân, từ hệ nhị phân sang hệ thập lục phân, và ngược lại.
* Sử dụng thuật toán phù hợp để thực hiện chuyển đổi giữa các hệ đếm số.

1. Xử lý lỗi và kiểm tra đầu vào:

* Đảm bảo rằng ứng dụng kiểm tra và xử lý các lỗi đầu vào từ người dùng. Ví dụ: kiểm tra xem số nhập vào có hợp lệ với tùy chọn trong menu hay không.
* Hiển thị thông báo lỗi hoặc cung cấp gợi ý cho người dùng khi có lỗi xảy ra.

1. Tích hợp chức năng mở rộng:

* Xây dựng chức năng cho phép người dùng có thể lựa chọn sẽ tiếp tục hoặc thoát.
  1. Ngôn ngữ cài đặt

Sử dụng ngôn ngữ C++ để cài đặt tất cả các thuật toán trên.

C++ là một loại ngôn ngữ lập trình bậc trung. Đây là ngôn ngữ lập trình đa năng được tạo ra bởi Bjarne Stroustrup như một phần mở rộng của ngôn ngữ lập trình C, hoặc "C với các lớp Class", Ngôn ngữ đã được mở rộng đáng kể theo thời gian và C ++ hiện đại có các tính năng: lập trình tổng quát, lập trình hướng đối tượng, lập trình thủ tục, ngôn ngữ đa mẫu hình tự do có kiểu tĩnh, dữ liệu trừu tượng, và lập trình đa hình, ngoài ra còn có thêm các tính năng, công cụ để thao tác với bộ nhớ cấp thấp. Từ thập niên 1990, C++ đã trở thành một trong những ngôn ngữ thương mại ưa thích và phổ biến của lập trình viên.

1. CƠ SỞ LÝ THUYẾT

Cơ sở lý thuyết của chương trình này liên quan đến hệ số cơ số và các phép chuyển đổi giữa các hệ số cơ số khác nhau, bao gồm:

1. Hệ thập phân (decimal): Đây là hệ số cơ số 10, được sử dụng phổ biến trong các hệ thống tính toán hàng ngày. Trong hệ thập phân, mỗi chữ số có thể có giá trị từ 0 đến 9. Số 1234 trong hệ thập phân biểu diễn giá trị: 1 \* 10^3 + 2 \* 10^2 + 3 \* 10^1 + 4 \* 10^0.

2. Hệ nhị phân (binary): Đây là hệ số cơ số 2, chỉ sử dụng các chữ số 0 và 1. Trong hệ nhị phân, mỗi chữ số biểu diễn một lũy thừa của 2. Ví dụ: số 1010 trong hệ nhị phân biểu diễn giá trị: 1 \* 2^3 + 0 \* 2^2 + 1 \* 2^1 + 0 \* 2^0.

3. Hệ bát phân (octal): Đây là hệ số cơ số 8, sử dụng các chữ số từ 0 đến 7. Trong hệ bát phân, mỗi chữ số biểu diễn một lũy thừa của 8. Ví dụ: số 765 trong hệ bát phân biểu diễn giá trị: 7 \* 8^2 + 6 \* 8^1 + 5 \* 8^0.

4. Hệ thập lục phân (hexadecimal): Đây là hệ số cơ số 16, sử dụng các chữ số từ 0 đến 9 và các ký tự chữ cái từ A đến F (hoặc a đến f) để biểu diễn các giá trị từ 10 đến 15. Trong hệ thập lục phân, mỗi chữ số biểu diễn một lũy thừa của 16. Ví dụ: số 1F trong hệ thập lục phân biểu diễn giá trị: 1 \* 16^1 + 15 \* 16^0.

Chương trình này cung cấp các hàm để thực hiện các phép chuyển đổi giữa các hệ số cơ số trên. Ví dụ, hàm `decimalToBinary` chuyển đổi một số thập phân sang dạng nhị phân, hàm `binaryToDecimal` chuyển đổi một số nhị phân sang dạng thập phân, và tương tự cho các hệ số cơ số khác.

Các phép chuyển đổi giữa các hệ số cơ số thường dựa trên cơ sở lũy thừa của hệ số cơ số đó. Dưới đây là một số phép chuyển đổi cơ bản:

1. Chuyển đổi từ hệ thập phân sang hệ nhị phân:

* Bước 1: Chia số thập phân cho 2 và lưu lại phần dư.
* Bước 2: Tiếp tục chia kết quả trước đó cho 2 và lưu phần dư mới.
* Bước 3: Lặp lại bước 2 cho đến khi kết quả là 0.
* Bước 4: Sắp xếp các phần dư đã lưu theo thứ tự từ dưới lên để tạo thành số nhị phân.

1. Chuyển đổi từ hệ nhị phân sang hệ thập phân:

* Bước 1: Phân tích số nhị phân thành từng chữ số.
* Bước 2: Nhân mỗi chữ số với 2 mũ số hạng tương ứng (2^0, 2^1, 2^2, ...) từ phải sang trái.
* Bước 3: Tính tổng các kết quả đã nhân.

1. Chuyển đổi từ hệ thập phân sang hệ bát phân:

* Bước 1: Tương tự như chuyển đổi sang hệ nhị phân, nhưng chia cho 8 thay vì 2.
* Bước 2: Sắp xếp các phần dư đã lưu theo thứ tự từ dưới lên để tạo thành số bát phân.

1. Chuyển đổi từ hệ bát phân sang hệ thập phân:

* Bước 1: Phân tích số bát phân thành từng chữ số.
* Bước 2: Nhân mỗi chữ số với 8 mũ số hạng tương ứng (8^0, 8^1, 8^2, ...) từ phải sang trái.
* Bước 3: Tính tổng các kết quả đã nhân.

1. Chuyển đổi từ hệ thập phân sang hệ thập lục phân:

* Bước 1: Tương tự như chuyển đổi sang hệ nhị phân, nhưng chia cho 16 thay vì 2.
* Bước 2: Sắp xếp các phần dư đã lưu theo thứ tự từ dưới lên và sử dụng ký tự chữ cái A-F để biểu diễn các giá trị từ 10 đến 15 (nếu cần).

1. Chuyển đổi từ hệ thập lục phân sang hệ thập phân:

* Bước 1: Phân tích số thập lục phân thành từng chữ số.
* Bước 2: Nhân mỗi chữ số với 16 mũ số hạng tương ứng (16^0, 16^1, 16^2, ...) từ phải sang trái.
* Bước 3: Tính tổng các kết quả đã nhân.

1. Đổi từ hệ nhị phân sang hệ thập lục phân.

* Bước 1: Đổi từ hệ nhị phân sang hệ thập phân bằng cách gọi hàm ‘binaryToDecimal’.
* Bươc 2: Đổi từ hệ thập phân sang hệ thập lục phân bằng cách gọi hàm ‘decimalToHexadecimal’.

1. Đổi từ hệ nhị phân sang hệ bát phân.

* Bước 1: Đổi từ hệ nhị phân sang hệ thập phân bằng cách gọi hàm ‘binaryToDecimal’.
* Bươc 2: Đổi từ hệ thập phân sang hệ bát phân bằng cách gọi hàm ‘decimalToOctal’.

1. Đổi từ hệ bát phân sang hệ nhi phân.

* Bước 1: Đổi từ hệ bát phân sang hệ thập phân bằng cách gọi hàm ‘octalToDecimal’.
* Bươc 2: Đổi từ hệ thập phân sang hệ nhị phân bằng cách gọi hàm ‘decimalToBinary’.

1. Đổi từ hệ thập lục phân sang hệ nhị phân.

* Bước 1: Đổi từ hệ thập lục phân sang hệ thập phân bằng cách gọi hàm ‘hexadecimalToDecimal’.
* Bươc 2: Đổi từ hệ thập phân sang hệ nhị phân bằng cách gọi hàm ‘decimalToBinary’.

1. Thoát
2. TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN
   1. Phát biểu bài toán

* Đầu vào (Input): Người dùng sẽ nhập lựa chọn từ menu để thực hiện các chuyển đổi hoặc thoát chương trình. Người dùng cũng sẽ cung cấp các số thập phân, nhị phân, bát phân hoặc thập lục phân cần được chuyển đổi.
* Đầu ra (Output): Chương trình sẽ hiển thị kết quả chuyển đổi tương ứng cho số được cung cấp bởi người dùng. Kết quả sẽ được hiển thị trên màn hình console.
  1. Cấu trúc dữ liệu

Dưới đây là một tổng quan về các cấu trúc dữ liệu được sử dụng trong chương trình:

1. Hàm `main`:

* Không sử dụng các cấu trúc dữ liệu cụ thể trong hàm `main`.
* Hàm này khởi tạo các biến và gọi hàm `menu` để hiển thị menu chính.

1. Hàm `menu`:

* Biến:
* `int choice`: Lưu trữ sự lựa chọn của người dùng từ menu.
* `double decimal`: Lưu trữ số thập phân được nhập bởi người dùng.
* `string binary`, `string octal`, `string hexadecimal`: Lưu trữ số nhị phân, bát phân và thập lục phân được nhập bởi người dùng.
* Luồng điều khiển:
* Hàm này sử dụng một vòng lặp do-while để hiển thị các tùy chọn menu và xử lý sự lựa chọn của người dùng.
* Sự lựa chọn của người dùng được đọc từ đầu vào và sau đó được xử lý trong một câu lệnh switch.
* Dựa trên sự lựa chọn của người dùng, các hàm chuyển đổi khác nhau được gọi.
* Vòng lặp tiếp tục cho đến khi người dùng chọn tùy chọn "Thoát".

1. Các hàm chuyển đổi:

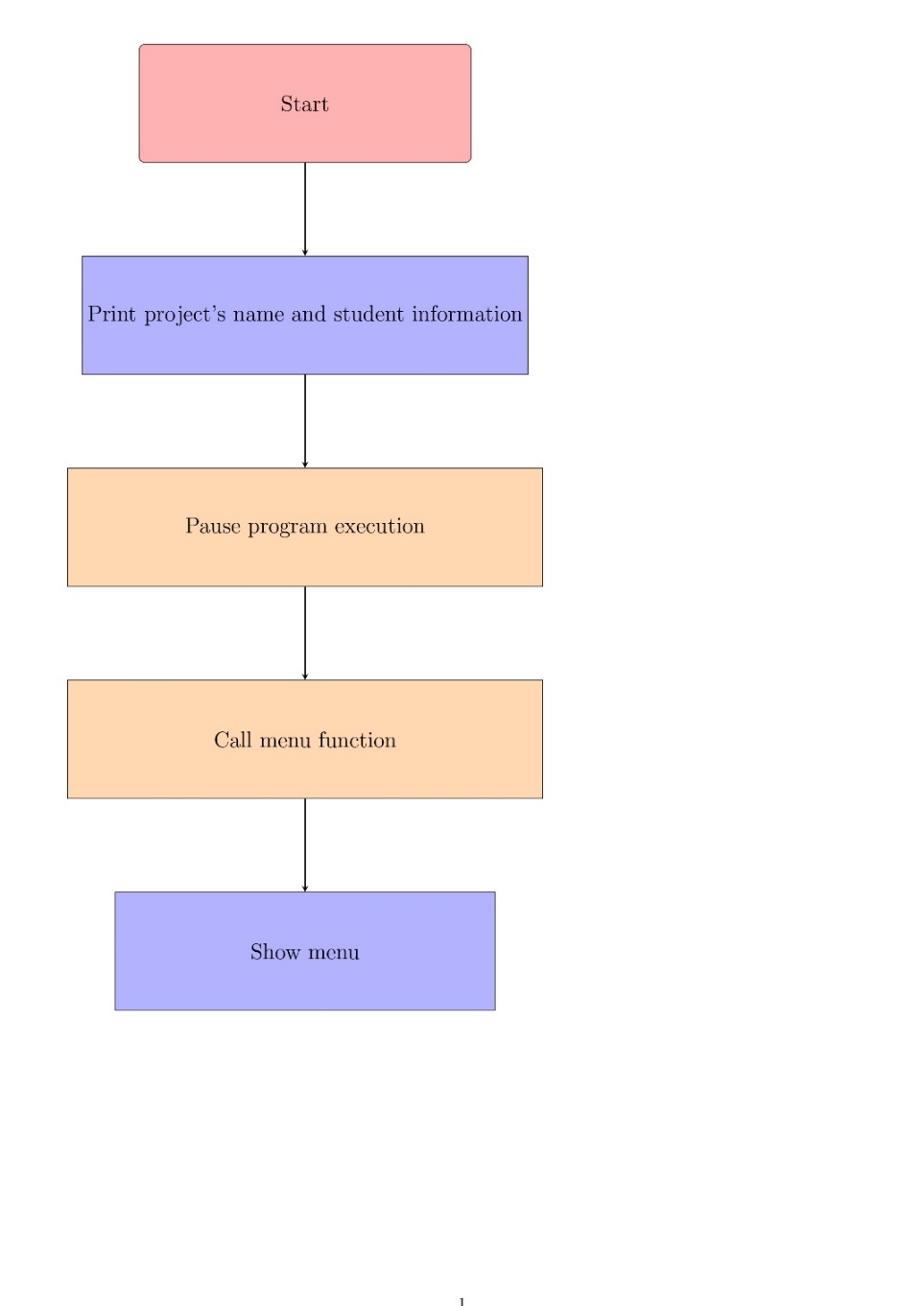
* Định nghĩa nhiều hàm để xử lý chuyển đổi giữa các hệ thống số khác nhau (thập phân, nhị phân, bát phân, thập lục phân).
* Mỗi hàm nhận các tham số đầu vào và trả về giá trị đã được chuyển đổi.
* Các hàm chuyển đổi sử dụng các kiểu dữ liệu cơ bản như `int`, `double` và `string`.
* Các hàm sử dụng vòng lặp, các phép toán số học và xử lý chuỗi để thực hiện chuyển đổi.

Tổng thể, chương trình chủ yếu dựa vào các kiểu dữ liệu cơ bản và cấu trúc luồng điều khiển để xử lý chuyển đổi giữa các hệ thống số. Nó không sử dụng các cấu trúc dữ liệu phức tạp từ thư viện tiêu chuẩn hoặc định nghĩa các cấu trúc dữ liệu tùy chỉnh.

* 1. Thuật toán
* **Hàm main:**

Đây là thuật toán của hàm `main()`:

1. Hiển thị thông tin về dự án và sinh viên thực hiện.
2. Dừng thực thi chương trình để người dùng có thời gian đọc thông tin.
3. Gọi hàm `menu()` để hiển thị các tùy chọn trong menu và xử lý chuyển đổi giữa các hệ đếm khác nhau.
4. Trả về 0 để chỉ ra rằng chương trình đã thực thi thành công.



Hình 1: Flowchart Hàm main.

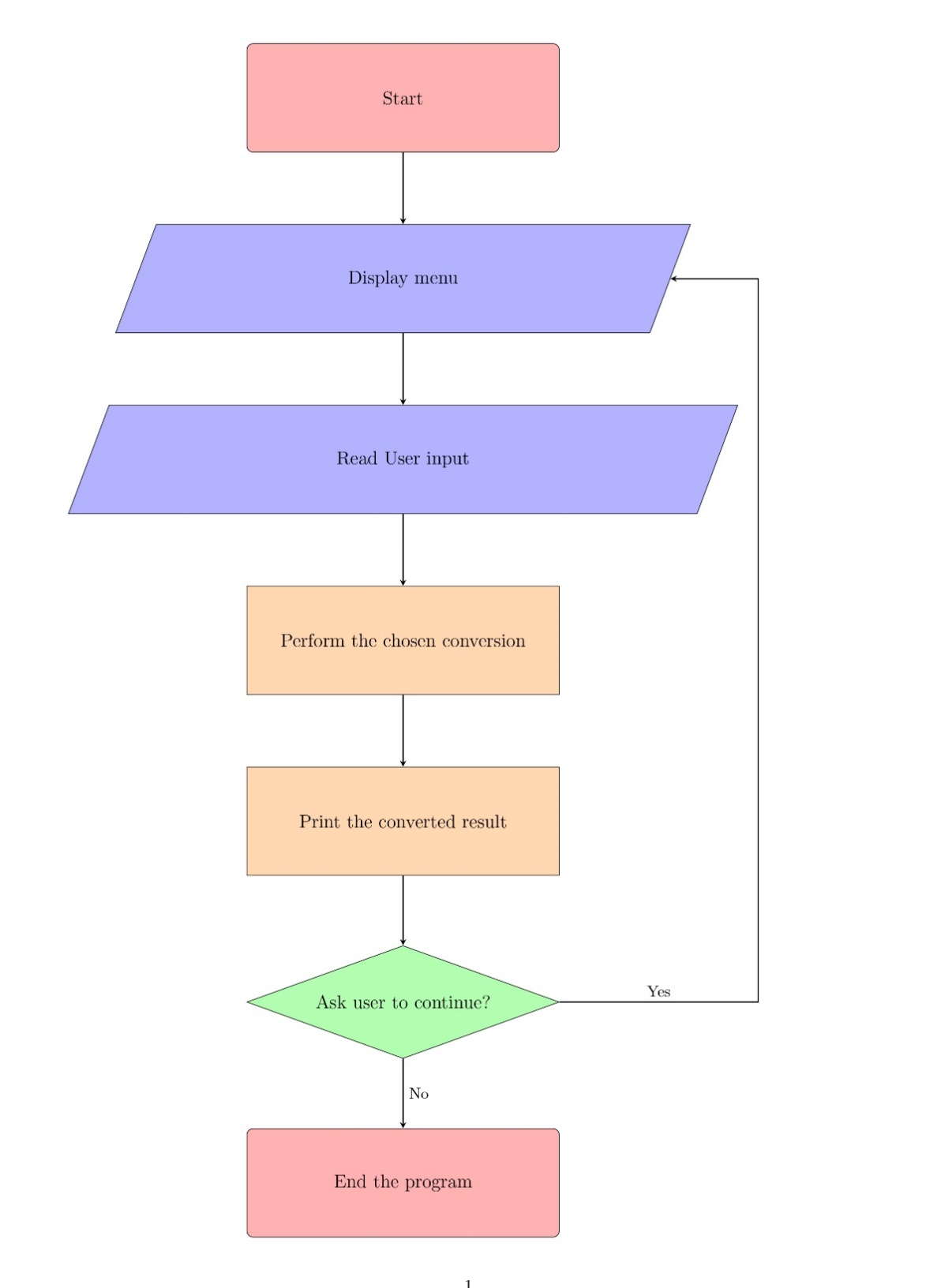
* **Hàm menu:**

Hàm `menu()` trong chương trình này được sử dụng để hiển thị một menu các tùy chọn cho việc chuyển đổi số. Dưới đây là cách hoạt động của hàm:

1. Khai báo biến: Hàm khai báo một số biến như `choice` (lưu trữ lựa chọn của người dùng), `decimal` (lưu trữ số thập phân), `binary`, `octal`, và `hexadecimal` (lưu trữ chuỗi biểu diễn các số nhị phân, bát phân và thập lục phân tương ứng).
2. Hiển thị menu: Hàm hiển thị một menu với 11 tùy chọn sử dụng lệnh `cout`. Người dùng được yêu cầu chọn một tùy chọn.
3. Nhập và xử lý lựa chọn: Hàm sử dụng `cin` để đọc lựa chọn từ người dùng. Nếu đầu vào là một số hợp lệ (số nguyên), một câu lệnh `switch` được sử dụng để xử lý tùy chọn được chọn.
4. Xử lý từng tùy chọn: Đối với mỗi tùy chọn được chọn bởi người dùng, hàm thực hiện phép chuyển đổi tương ứng và hiển thị kết quả.

* Tùy chọn 1-3: Chuyển đổi từ số thập phân sang số nhị phân, bát phân và thập lục phân. Người dùng được yêu cầu nhập một số thập phân, sau đó hàm chuyển đổi tương ứng (`decimalToBinary()`,`decimalToOctal()`,`decimalToHexadecimal()`) được sử dụng để thực hiện chuyển đổi.
* Tùy chọn 4-7: Chuyển đổi từ số nhị phân sang số thập phân, bát phân và thập lục phân. Người dùng được yêu cầu nhập một số nhị phân, sau đó hàm chuyển đổi tương ứng (`binaryToDecimal()`, `binaryToOctal()`, `binaryToHexadecimal()`) được sử dụng để thực hiện chuyển đổi.
* Tùy chọn 8-10: Chuyển đổi từ số bát phân sang số nhị phân, thập phân và thập lục phân. Người dùng được yêu cầu nhập một số bát phân, sau đó hàm chuyển đổi tương ứng (`octalToBinary()`, `octalToDecimal()`, `octalToHexadecimal()`) được sử dụng để thực hiện chuyển đổi.
* Tùy chọn 11: Thoát chương trình. Điều này dẫn đến kết thúc vòng lặp và kết thúc hàm.
* Trường hợp mặc định: Xử lý lựa chọn không hợp lệ. Hiển thị thông báo lỗi.

1. Tương tác với người dùng: Sau mỗi lần chuyển đổi, người dùng được hỏi liệu họ muốn tiếp tục hay không.
2. Xử lý lỗi: Nếu người dùng nhập một lựa chọn không hợp lệ (giá trị không phải số), hiển thị thông báo lỗi. Sau đó, chương trình tạm dừng, xóa màn hình và gọi lại hàm `menu()` để bắt đầu lại menu.



Hình 2: Flowchart Hàm menu.

1. **Hàm chuyển đổi từ hệ thập phân sang nhị phân (decimalToBinary):**
2. Khởi tạo chuỗi rỗng `binary` để lưu trữ biểu diễn nhị phân.
3. Kiểm tra xem giá trị decimal đầu vào có âm hay không. Nếu có, đặt cờ `isNegative` thành true và lấy giá trị tuyệt đối của decimal.
4. Chia decimal thành phần nguyên và phần thập phân:

* Chuyển decimal thành chuỗi sử dụng `to\_string`.
* Tìm vị trí dấu thập phân trong chuỗi sử dụng `find('.')` và lưu vào `decimalPointPos`.
* Trích xuất phần nguyên từ vị trí 0 đến `decimalPointPos` trong chuỗi và chuyển đổi thành số nguyên sử dụng `stoi`.
* Trích xuất phần thập phân từ `decimalPointPos` đến cuối chuỗi và chuyển đổi thành số thực sử dụng `stod`. Nếu `decimalPointPos` là `string::npos`, tức là không có dấu thập phân, đặt phần thập phân thành 0.

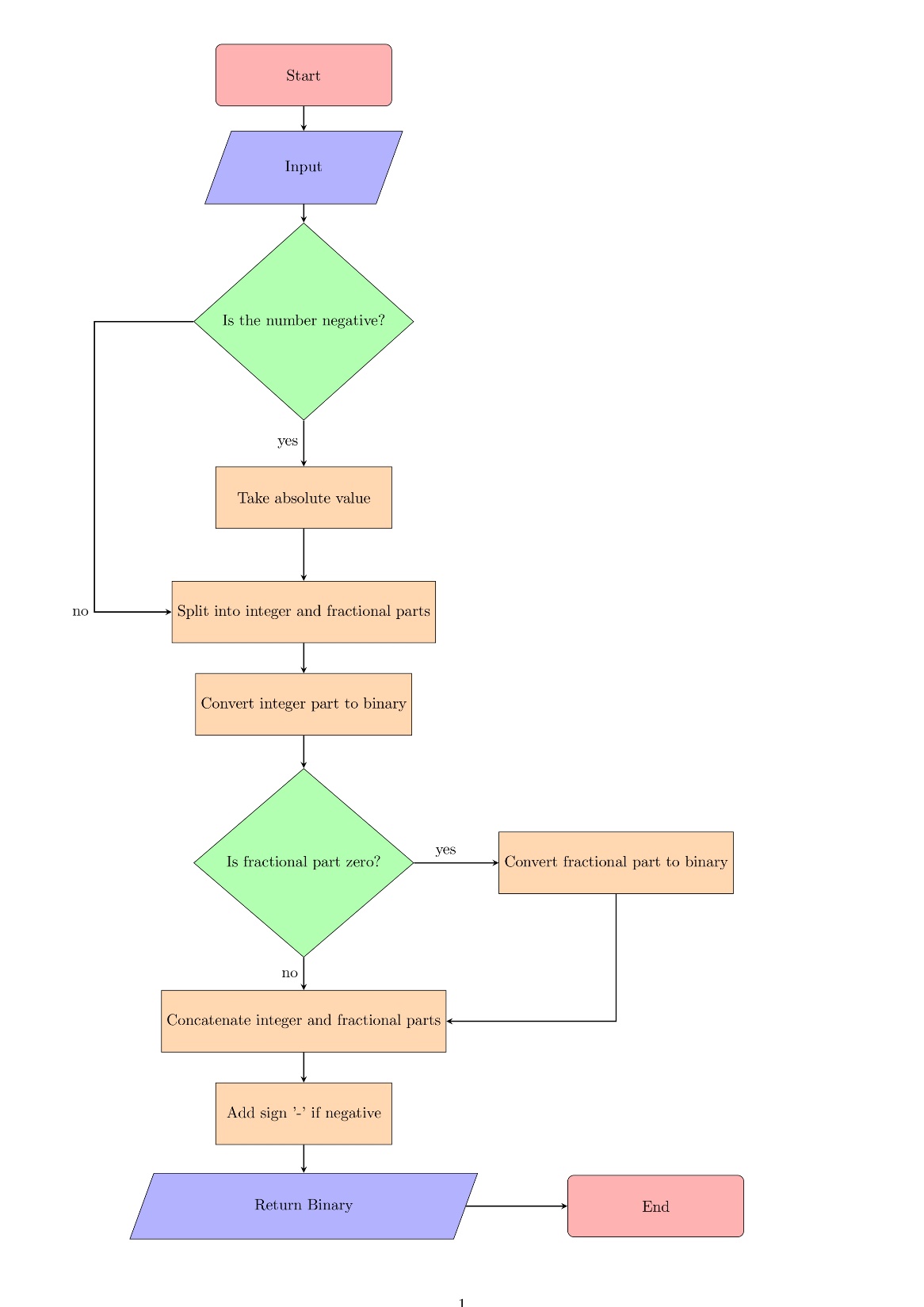
1. Chuyển đổi phần nguyên thành nhị phân:

* Sử dụng vòng lặp để chia lần lượt phần nguyên cho 2 cho đến khi giá trị trở thành 0.
* Lấy phần dư sau mỗi lần chia và thêm vào chuỗi `binary` sử dụng `to\_string`.
* Cập nhật phần nguyên bằng cách chia cho 2.

1. Nếu phần thập phân khác 0, thêm dấu thập phân vào chuỗi `binary`:

* Thêm dấu thập phân vào chuỗi `binary`.
* Chuyển đổi phần thập phân sang nhị phân:
* Đặt một giới hạn số bit tối đa cho phần thập phân, trong trường hợp này là 32.
* Sử dụng vòng lặp để nhân phần thập phân cho 2 cho đến khi giá trị trở thành 0 hoặc độ dài chuỗi `binary` đạt tới giới hạn số bit tối đa.
* Lấy phần nguyên của giá trị nhân với 2 và thêm vào chuỗi `binary` sử dụng `to\_string`.
* Cập nhật phần thập phân bằng cách trừ phần nguyên từ giá trị đó.

1. Nếu giá trị decimal ban đầu là âm, thêm dấu "-" vào đầu chuỗi `binary`.
2. Trả về chuỗi `binary` kết quả.

****

Hình : Flowchart Hàm chuyển đổi từ hệ thập phân sang nhị phân.

1. **Hàm chuyển đổi từ hệ thập phân sang bát phân (decimalToOctal):**
2. Khởi tạo một biến chuỗi rỗng được gọi là `octal`.
3. Khởi tạo một biến boolean gọi là `isNegative` và đặt giá trị là false.
4. Nếu giá trị decimal đầu vào là âm (decimal < 0):

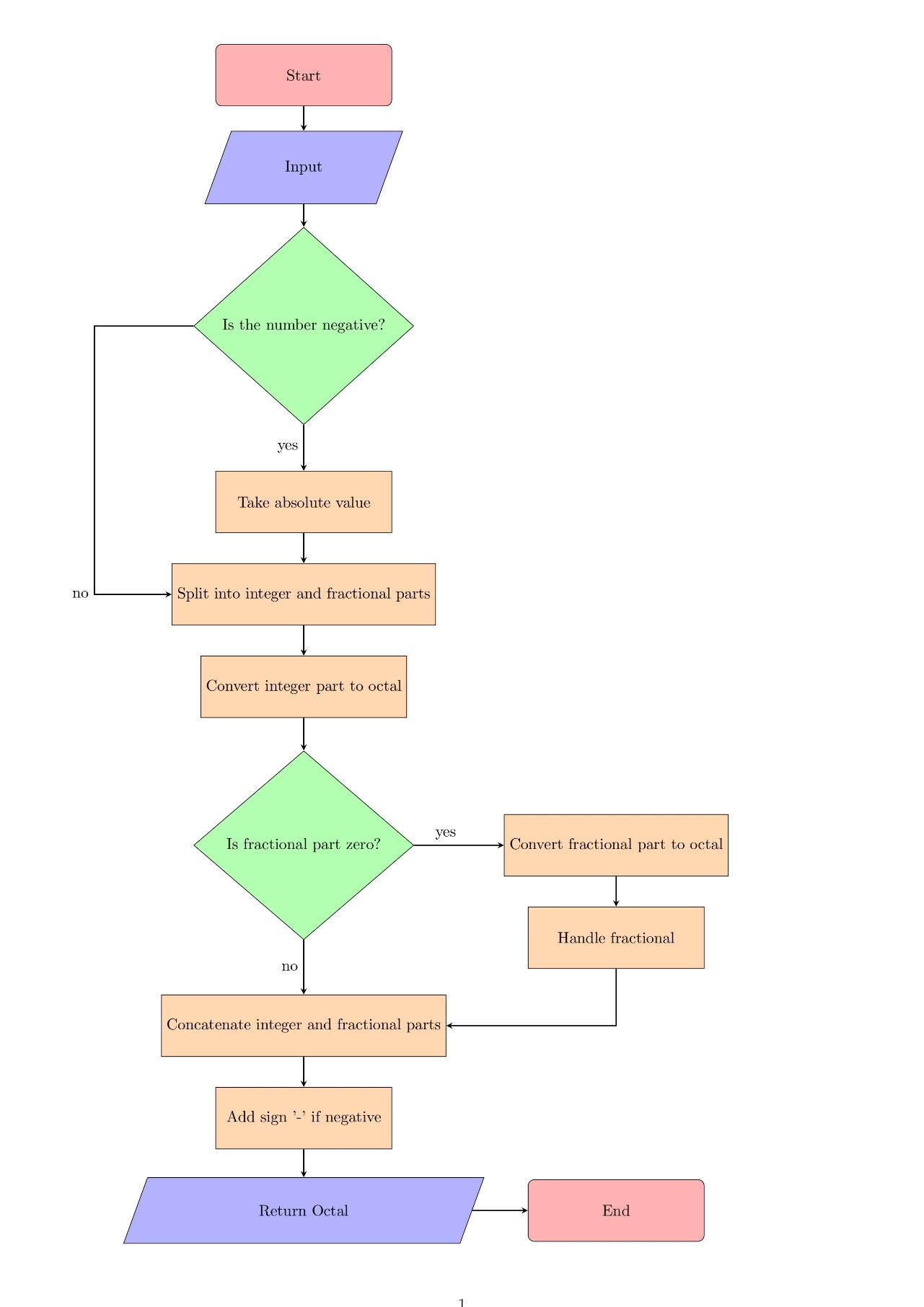
* Đặt `isNegative` thành true.
* Lấy giá trị tuyệt đối của decimal bằng cách sử dụng hàm abs().

1. Chuyển đổi giá trị decimal thành chuỗi sử dụng hàm to\_string() và lưu vào biến `decimalStr`.
2. Tìm vị trí dấu chấm thập phân trong `decimalStr` bằng cách sử dụng hàm find('.') và lưu vị trí vào `decimalPointPos`.
3. Trích xuất phần số nguyên của decimal bằng cách sử dụng hàm substr() từ vị trí 0 đến `decimalPointPos` trong `decimalStr` và chuyển đổi thành số nguyên bằng hàm stoi(). Lưu kết quả vào biến `integerPart`.
4. Trích xuất phần thập phân của decimal bằng cách sử dụng hàm substr() từ `decimalPointPos` tới cuối `decimalStr` và chuyển đổi thành số thực bằng hàm stod(). Nếu `decimalPointPos` bằng `string::npos`, tức là không có dấu chấm thập phân, đặt phần thập phân bằng 0.
5. Chuyển đổi phần số nguyên thành hệ bát phân:

* Nếu phần số nguyên bằng 0, đặt `octal` thành "0".
* Trong khi phần số nguyên lớn hơn 0:
* Tính phần dư của phần số nguyên chia cho 8 bằng toán tử %.
* Chuyển đổi phần dư thành chuỗi bằng hàm to\_string().
* Thêm phần dư vào đầu chuỗi `octal`.
* Cập nhật phần số nguyên bằng cách chia nó cho 8.

1. Nếu phần thập phân lớn hơn 0:

* Thêm dấu chấm thập phân vào chuỗi `octal`.
* Đặt một giới hạn tối đa cho số chữ số thập phân (ví dụ: 12).
* Lặp lại cho mỗi chữ số thập phân:
* Nhân phần thập phân với 8.
* Chuyển đổi phần số nguyên của kết quả nhân thành số nguyên bằng hàm int().
* Chuyển đổi phần số nguyên thành chuỗi bằng hàm to\_string().
* Thêm phần số nguyên vào chuỗi `octal`.
* Trừ phần số nguyên từ kết quả nhân để cập nhật phần thập phân.
* Xóa các số 0 dư thừa ở cuối chuỗi `octal`.
* Nếu ký tự cuối cùng trong chuỗi `octal` là dấu chấm thập phân, xóa nó đi.

1. Nếu giá trị decimal ban đầu là âm, thêm dấu "-" vào đầu chuỗi `octal`.
2. Trả về chuỗi `octal` là kết quả của hàm.

Hình : Flowchart Hàm chuyển đổi từ hệ thập phân sang bát phân.

1. **Hàm chuyển đổi từ hệ thập phân sang thập lục phân (decimalToHexadecimal):**
2. Khởi tạo một chuỗi rỗng có tên `hexadecimal`.
3. Kiểm tra xem số `decimal` đầu vào có âm không:

* Nếu có, đặt biến boolean `isNegative` thành true và lấy giá trị tuyệt đối của `decimal`.

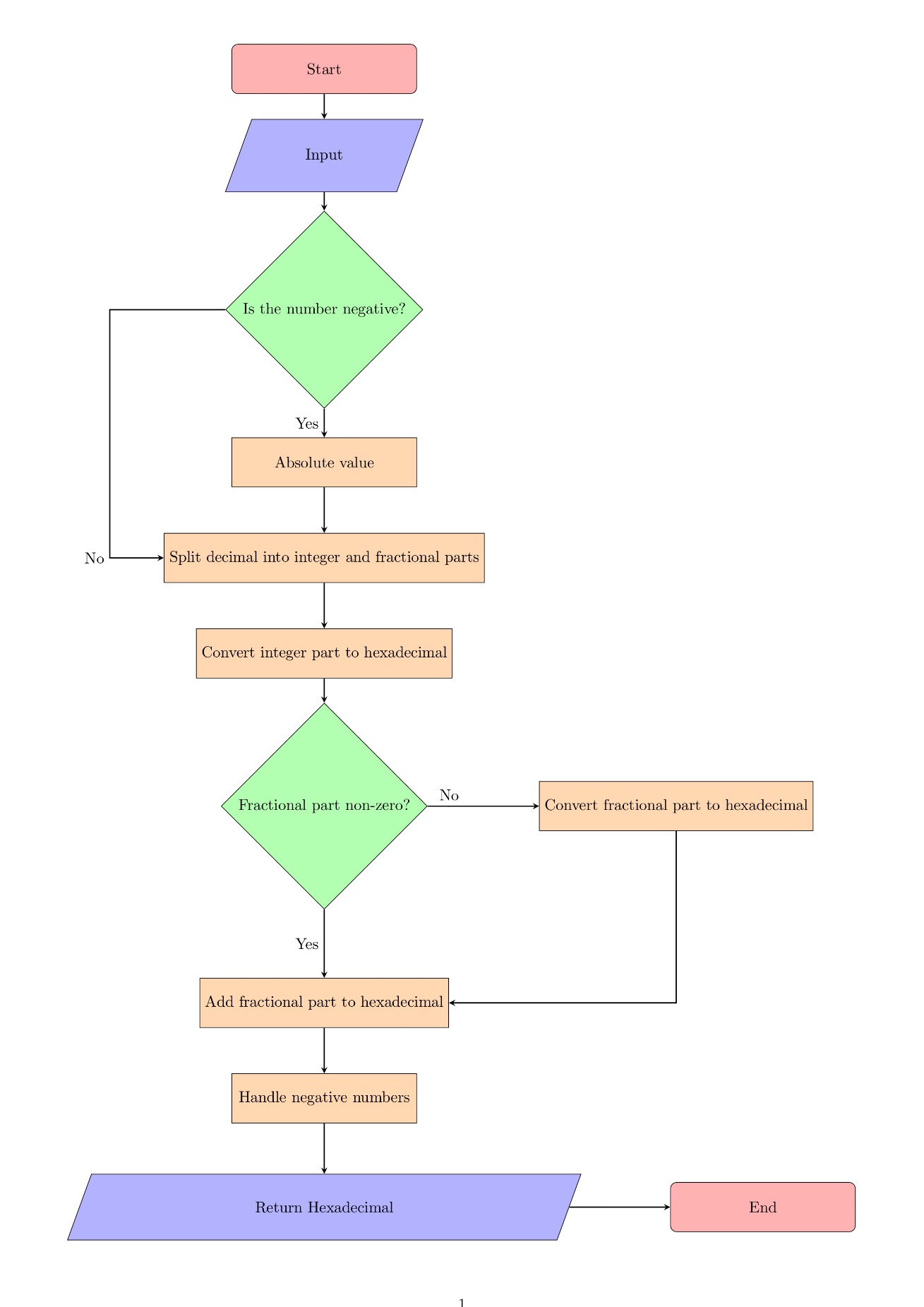
1. Chuyển `decimal` thành một chuỗi bằng cách sử dụng `to\_string()` và lưu vào biến `decimalStr`.
2. Tìm vị trí dấu chấm thập phân trong `decimalStr` bằng cách sử dụng `find('.')` và lưu vào `decimalPointPos`.
3. Trích xuất phần nguyên của decimal bằng cách sử dụng `substr()` từ vị trí 0 đến `decimalPointPos` trong `decimalStr` và lưu vào `integerPartStr`.
4. Trích xuất phần thập phân của decimal bằng cách sử dụng `substr()` từ `decimalPointPos + 1` đến cuối `decimalStr` và lưu vào `fractionalPartStr`. Nếu `decimalPointPos` bằng `string::npos`, tức là không có dấu chấm thập phân, thì đặt phần thập phân thành một chuỗi rỗng.
5. Chuyển phần nguyên thành hệ thập lục phân:

* Nếu phần nguyên là "0", đặt `hexadecimal` thành "0".
* Chuyển `integerPartStr` thành một số nguyên long long bằng cách sử dụng `stoll()`.
* Trong khi `integerPart` lớn hơn 0:
* Tính phần dư khi chia `integerPart` cho 16 bằng cách sử dụng phép chia lấy dư (%).
* Nếu phần dư nhỏ hơn 10, chuyển thành chuỗi bằng cách sử dụng `to\_string()` và đặt vào đầu chuỗi `hexadecimal`.
* Ngược lại, tính ký tự hex tương ứng từ 'A' đến 'F' và đặt vào đầu chuỗi `hexadecimal`.
* Cập nhật `integerPart` bằng cách chia cho 16.

1. Nếu phần thập phân không rỗng:

* Thêm dấu chấm thập phân vào cuối chuỗi `hexadecimal`.
* Chuyển phần thập phân thành một số thực bằng cách chuyển chuỗi "0." + `fractionalPartStr` thành một số thực bằng cách sử dụng `stod()`.
* Đặt một giá trị tối đa cho số chữ số thập phân, `maxFractionalDigits`, là 6.
* Lặp từ 0 đến `maxFractionalDigits` và trong khi phần thập phân lớn hơn 0:
* Nhân phần thập phân với 16.
* Chuyển phần nguyên của phần thập phân thành một số nguyên bằng cách sử dụng ép kiểu.
* Nếu số nguyên nhỏ hơn 10, chuyển thành chuỗi và thêm vào cuối chuỗi `hexadecimal`.
* Ngược lại, tính ký tự hex tương ứng từ 'A' đến 'F' và thêm vào cuối chuỗi `hexadecimal`.
* Cập nhật phần thập phân bằng cách trừ đi phần nguyên đã sử dụng.

1. Nếu giá trị decimal ban đầu là âm, thêm dấu "-" vào đầu chuỗi `hexadecimal`.
2. Trả về chuỗi `hexadecimal` là kết quả của hàm.



Hình 5: Flowchart Hàm chuyển đổi từ hệ thập phân sang thập lục phân.

1. **Hàm chuyển đổi từ nhị phân sang hệ thập phân (binaryToDecimal):**
2. Khởi tạo biến boolean `isNegative` với giá trị mặc định là false.
3. Kiểm tra xem số nhị phân `binary` có âm không:

* Nếu có, đặt `isNegative` thành true và loại bỏ dấu "-" từ `binary`.

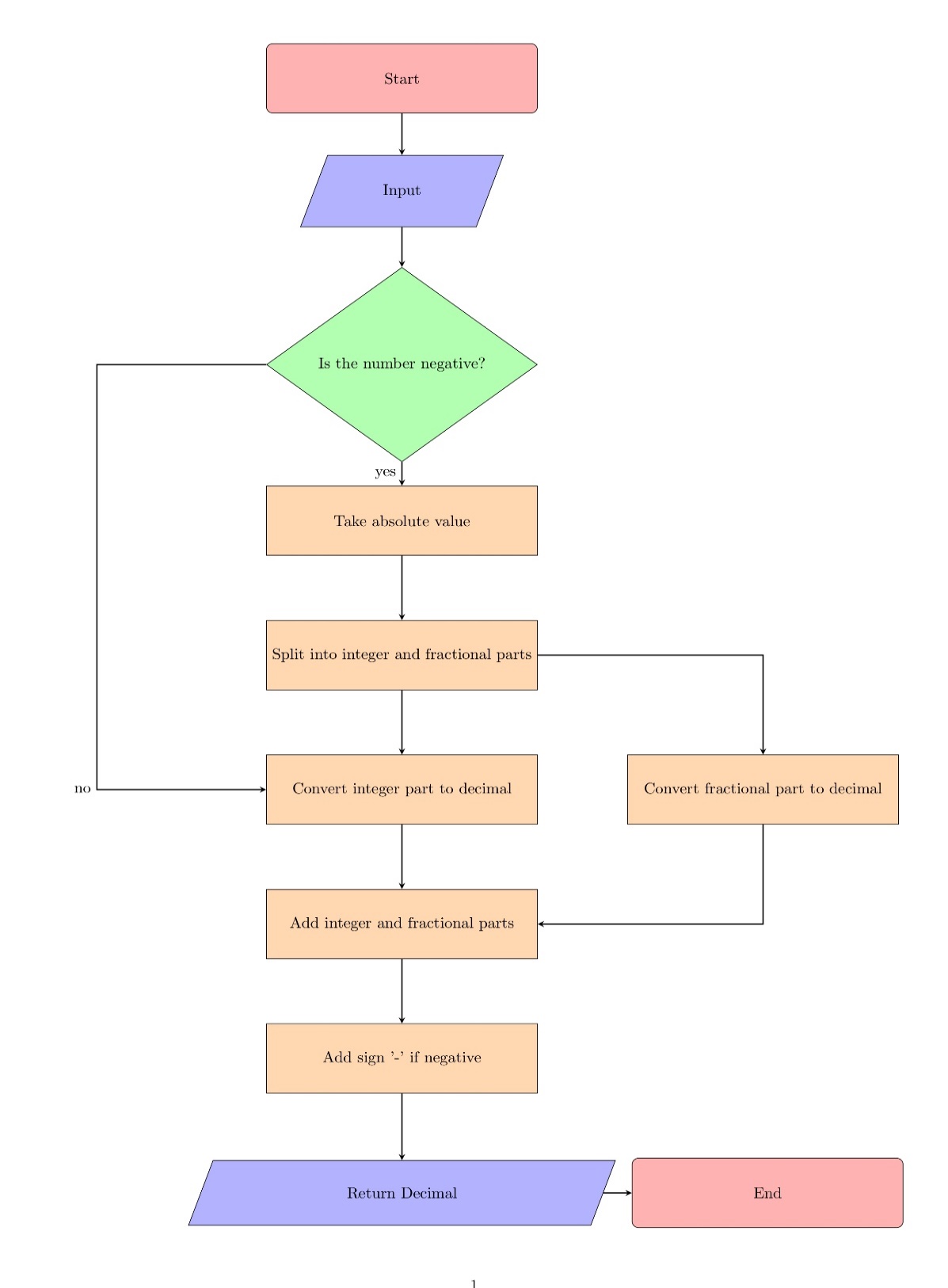
1. Tìm vị trí dấu chấm thập phân trong `binary` bằng cách sử dụng `find('.')` và lưu vào `decimalPointPos`.
2. Trích xuất phần nguyên của `binary` bằng cách sử dụng `substr()` từ vị trí 0 đến `decimalPointPos` trong `binary` và lưu vào `integerPartStr`. Nếu `decimalPointPos` bằng `string::npos`, tức là không có dấu chấm thập phân, thì đặt phần nguyên thành `binary`.
3. Trích xuất phần thập phân của `binary` bằng cách sử dụng `substr()` từ `decimalPointPos + 1` đến cuối `binary` và lưu vào `fractionalPartStr`. Nếu `decimalPointPos` bằng `string::npos`, tức là không có dấu chấm thập phân, thì đặt phần thập phân thành một chuỗi rỗng.
4. Chuyển phần nguyên thành số thập phân:

* Khởi tạo biến `integerPart` với giá trị ban đầu là 0.
* Lặp qua từng ký tự trong `integerPartStr`:
* Tính giá trị của ký tự nhị phân trên vị trí hiện tại bằng cách chuyển đổi từ ký tự sang số và nhân với lũy thừa 2 tương ứng với vị trí.
* Cộng giá trị này vào `integerPart`.

1. Chuyển phần thập phân thành số thập phân:

* Khởi tạo biến `fractionalPart` với giá trị ban đầu là 0.
* Lặp qua từng ký tự trong `fractionalPartStr`:
* Tính giá trị của ký tự nhị phân trên vị trí hiện tại bằng cách chuyển đổi từ ký tự sang số và nhân với lũy thừa 2 âm tương ứng với vị trí.
* Cộng giá trị này vào `fractionalPart`.

1. Tính tổng của phần nguyên và phần thập phân để có giá trị số thập phân.
2. Nếu số nhị phân ban đầu là âm, đổi dấu của `decimal` thành số âm bằng cách nhân với -1.
3. Trả về `decimal` là kết quả của hàm.



Hình 6 : Flowchart Hàm chuyển đổi từ nhị phân sang hệ thập phân.

1. **Hàm chuyển đổi từ bát phân sang hệ thập phân (octalToDecimal):**

Dưới đây là phân tích thuật toán từng bước:

1. Khởi tạo các biến:

* `isNegative` (bool): Đánh dấu xem số octal có âm hay không.
* `decimal` (double): Biến để lưu giá trị thập phân.
* `base` (double): Giá trị cơ số để chuyển đổi từng chữ số.

1. Kiểm tra số có âm hay không:

* Nếu chữ số đầu tiên của `octal` là '-', gán `isNegative = true` và loại bỏ dấu trừ từ số octal bằng cách sử dụng `octal.substr(1)`.

1. Xử lý phần nguyên:

* Tìm vị trí của dấu chấm thập phân trong số octal.
* Nếu không tìm thấy dấu chấm, xem toàn bộ chuỗi là phần nguyên.
* Duyệt qua từng chữ số của phần nguyên từ phải qua trái:
  + Chuyển đổi từng chữ số octal sang thập phân bằng cách `(octal[i] - '0')`.
  + Nhân chữ số thập phân với cơ số hiện tại `base` và cộng vào `decimal`.
  + Cập nhật giá trị cơ số `base` lên 8 (vì hệ octal).

1. Xử lý phần thập phân (nếu có):

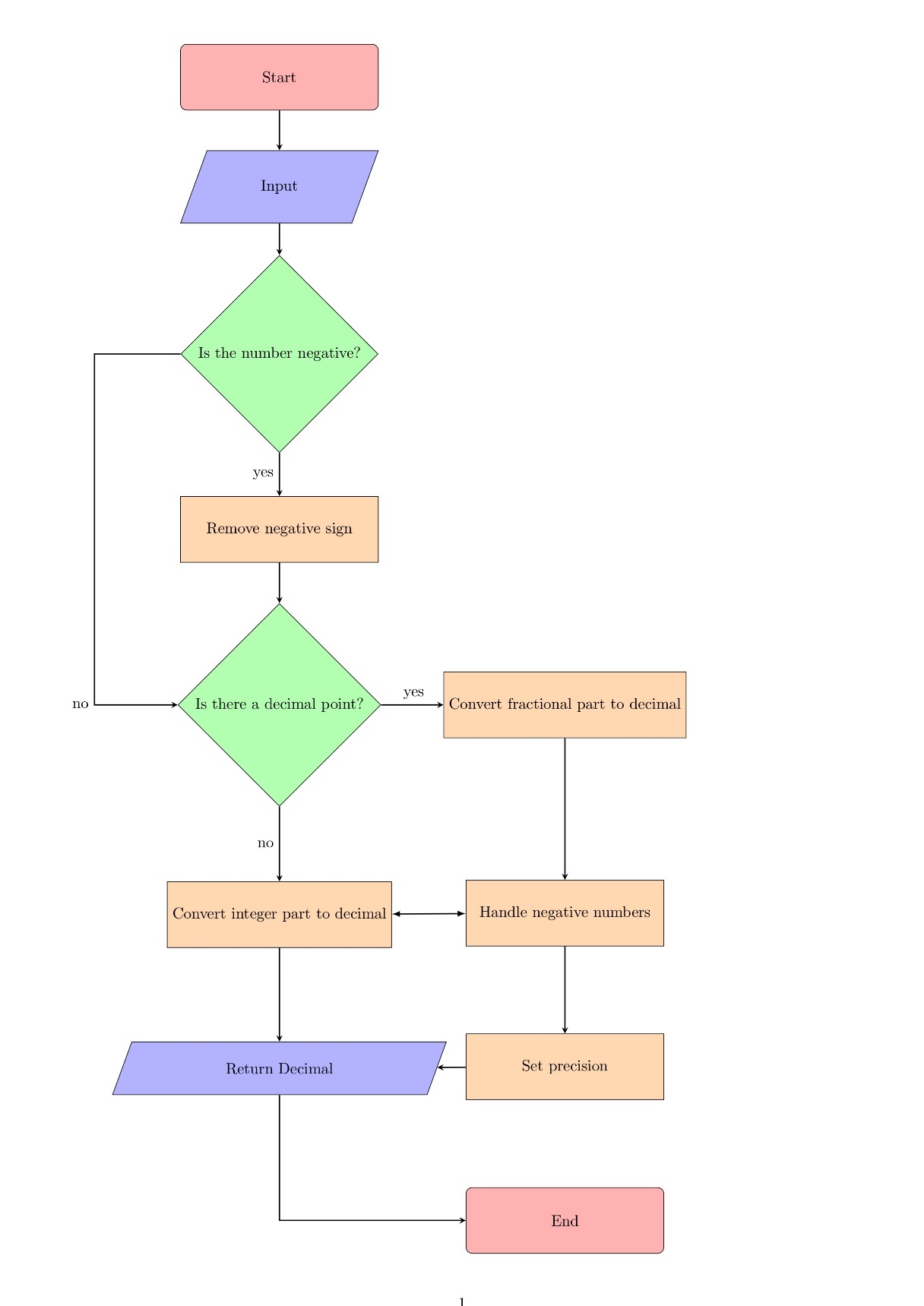
* Nếu tìm thấy dấu chấm, tiếp tục xử lý phần thập phân.
* Đặt lại giá trị cơ số `base` thành 1/8 (1 chia 8) để xử lý phần thập phân.
* Duyệt qua từng chữ số của phần thập phân từ trái qua phải:
* Chuyển đổi từng chữ số octal sang thập phân bằng cách `(octal[i] - '0')`.
* Nhân chữ số thập phân với cơ số hiện tại `base` và cộng vào `decimal`.
* Cập nhật giá trị cơ số `base` bằng cách chia 8.

1. Xử lý số âm (nếu có):

* Nếu số octal ban đầu là âm, đảo dấu của số thập phân.

1. Xác định số lượng chữ số có ý nghĩa tối đa:

* Tính giá trị epsilon, tức là sự sai biệt nhỏ nhất có thể có trong số thực.
* Tính toán số lượng chữ số có ý nghĩa tối đa bằng cách tính `log10(epsilon) - log10(decimal)` và lấy phần nguyên âm của nó.
* Đặt độ chính xác cho giá trị thập phân bằng cách sử dụng `cout.precision()`.

1. Trả về giá trị thập phân.

Hình 7: Flowchart Hàm chuyển đổi từ bát phân sang hệ thập phân.

1. **Hàm chuyển đổi từ thập lục phân sang hệ thập phân (hexadecimalToDecimal):**

Dưới đây là phân tích thuật toán từng bước:

1. Khởi tạo các biến:

* `decimal` (double): Biến để lưu giá trị thập phân.
* `fractional` (double): Biến để lưu phần thập phân của giá trị thập phân.
* `isNegative` (bool): Đánh dấu xem giá trị thập lục phân có âm hay không.

1. Kiểm tra số có âm hay không:

* Nếu chữ số đầu tiên của `hexadecimal` là '-', gán `isNegative = true` và loại bỏ dấu trừ từ giá trị thập lục phân bằng cách sử dụng `hexadecimal.substr(1)`.

1. Tìm vị trí của dấu chấm thập phân trong giá trị thập lục phân bằng `decimalPointIndex = hexadecimal.find(".")`.

* Nếu không tìm thấy dấu chấm, xem toàn bộ chuỗi là phần nguyên.

1. Chuyển đổi phần nguyên từ hệ thập lục phân sang hệ thập phân:

* Duyệt qua từng chữ số của phần nguyên từ phải qua trái.
* Nếu chữ số là chữ số thập phân (0-9), thực hiện chuyển đổi bằng cách `(digit - '0') \* pow(16, decimalPointIndex - 1 - i)`.
* Nếu chữ số là chữ cái thập lục phân (A-F), thực hiện chuyển đổi bằng cách `(digit - 'A' + 10) \* pow(16, decimalPointIndex - 1 - i)`.

1. Chuyển đổi phần thập phân từ hệ thập lục phân sang hệ thập phân (nếu có):

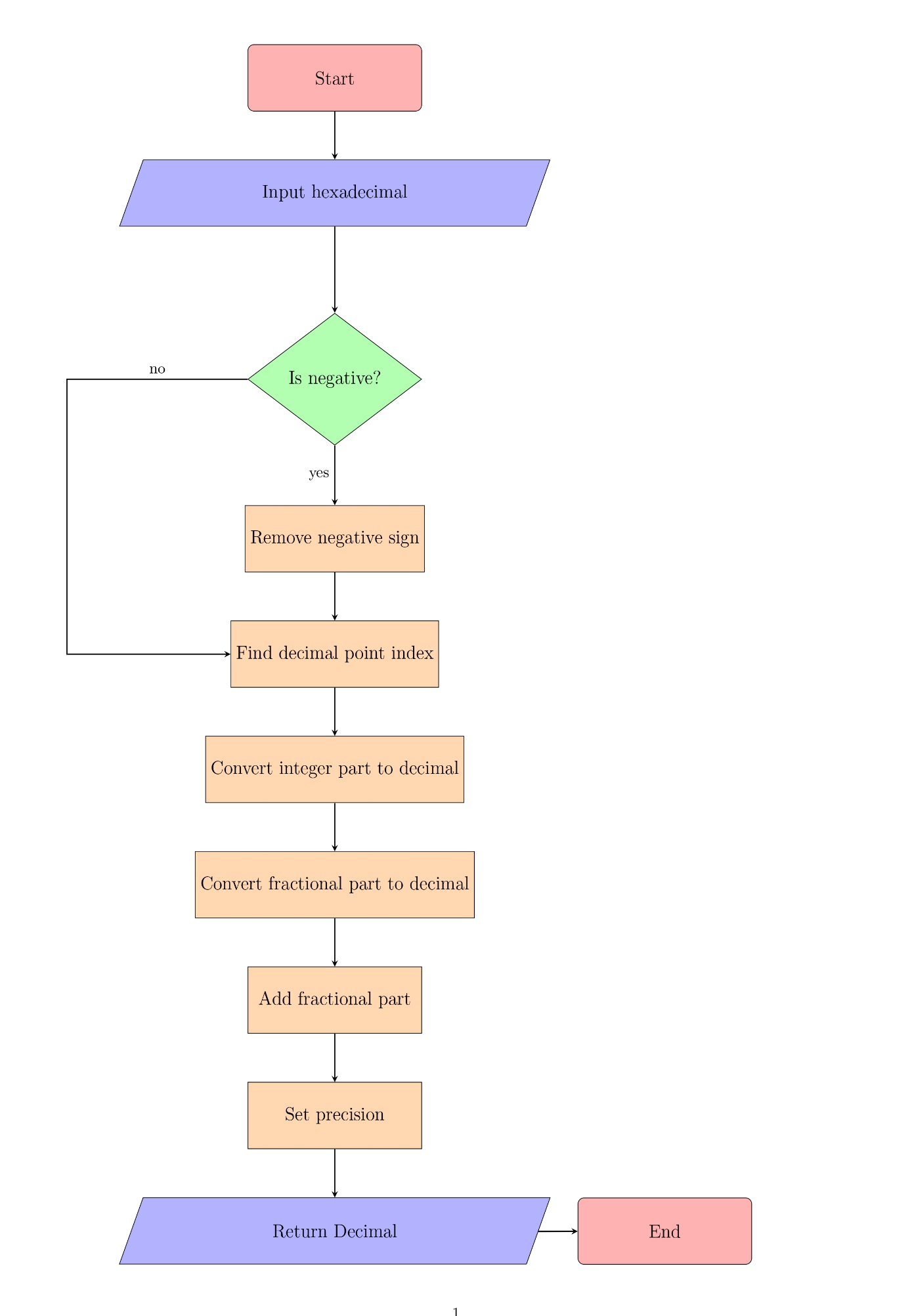
* Duyệt qua từng chữ số của phần thập phân từ trái qua phải.
* Tương tự như bước 4, thực hiện chuyển đổi các chữ số thành giá trị thập phân và nhân với cơ số tương ứng.

1. Tính toán giá trị thập phân cuối cùng:

* Cộng giá trị phần thập phân vào giá trị phần nguyên.

1. Xác định số lượng chữ số thập phân có ý nghĩa tối đa:

* Tính giá trị epsilon, tức là sự sai biệt nhỏ nhất có thể có trong số thực.
* Tính toán số lượng chữ số thập phân có ý nghĩa tối đa bằng cách tính `log10(epsilon) - log10(decimal)` và lấy phần nguyên âm của nó.
* Đặt độ chính xác cho giá trị thập phân bằng cách sử dụng `cout.precision()`.

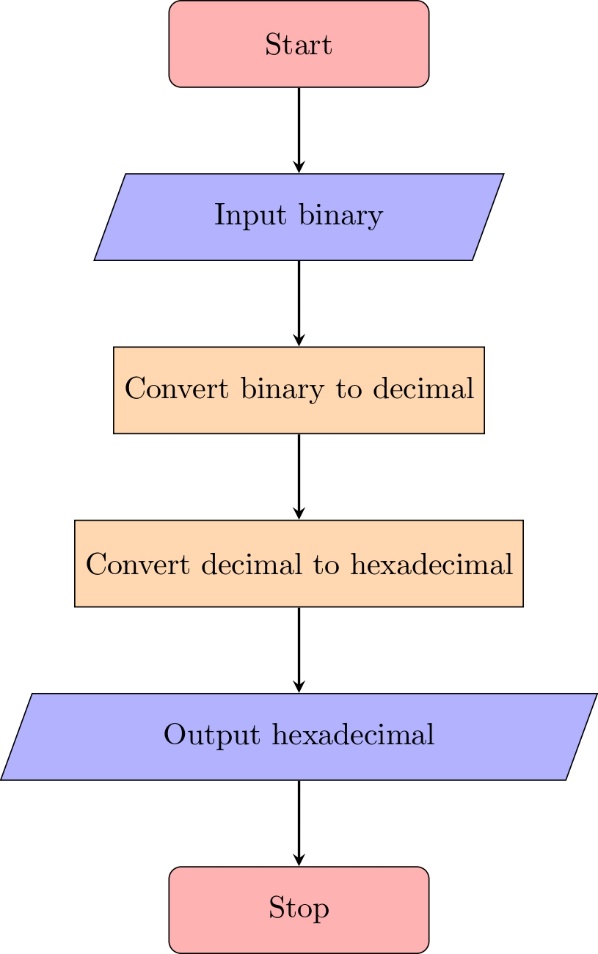
1. Trả về giá trị thập phân.

Hình : Flowchart Hàm chuyển đổi từ thập lục phân sang hệ thập phân.

1. **Hàm chuyển đổi từ nhi phân sang thập lúc phân (binaryToHexadecimal):**

* Bước 1: Đổi từ hệ nhị phân sang hệ thập phân bằng cách gọi hàm ‘binaryToDecimal’.
* Bước 2: Đổi từ hệ thập phân sang hệ thập lục phân bằng cách gọi hàm ‘decimalToHexadecimal’.

Tóm lại, hàm `binaryToHexadecimal` chuyển đổi một số nhị phân thành số thập lục phân bằng cách trước tiên chuyển đổi số nhị phân thành số thập phân, sau đó chuyển đổi số thập phân thành số thập lục phân.

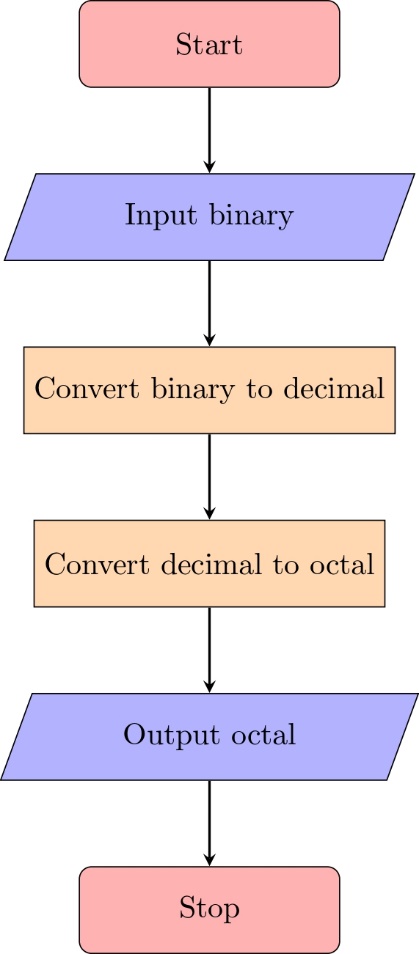


Hình 9: Flowchart Hàm chuyển đổi từ nhi phân sang thập lúc phân.

1. **Hàm chuyển đổi từ nhi phân sang bát phân (binaryToOctal):**

* Bước 1: Đổi từ hệ nhị phân sang hệ thập phân bằng cách gọi hàm ‘binaryToDecimal’.
* Bước 2: Đổi từ hệ thập phân sang hệ bát phân bằng cách gọi hàm ‘decimalToOctal’.

Tóm lại, hàm `binaryToOctal` chuyển đổi một số nhị phân thành số bát phân bằng cách trước tiên chuyển đổi số nhị phân thành số thập phân, sau đó chuyển đổi số thập phân thành số bát phân.

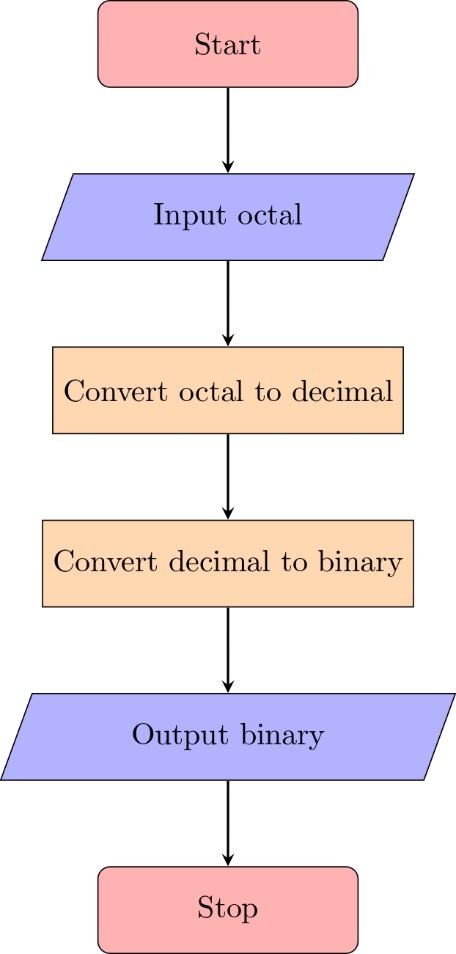


Hình 10: Flowchart Hàm chuyển đổi từ nhi phân sang bát phân.

1. **Hàm chuyển đổi từ bát phân sang nhi phân (octalToBinary):**

* Bước 1: Đổi từ hệ bát phân sang hệ thập phân bằng cách gọi hàm ‘octalToDecimal’.
* Bươc 2: Đổi từ hệ thập phân sang hệ nhị phân bằng cách gọi hàm ‘decimalToBinary’.

Tóm lại, hàm `octalToBinary` chuyển đổi một số bát phân thành số nhị phân bằng cách trước tiên chuyển đổi số bát phân thành số thập phân, sau đó chuyển đổi số thập phân thành số nhị phân.

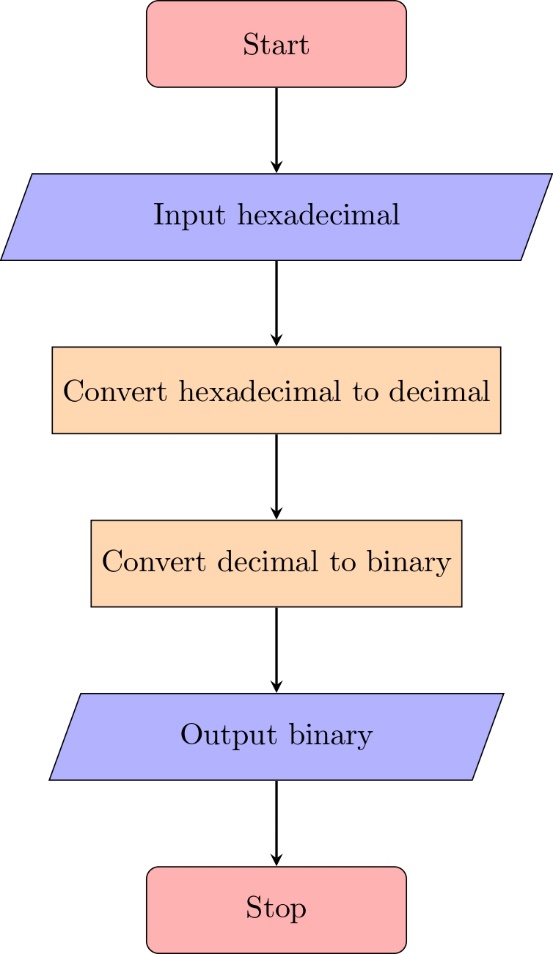


Hình 11: Flowchart Hàm chuyển đổi từ bát phân sang nhi phân.

1. **Hàm chuyển đổi từ thập lúc phân sang nhi phân (hexadecimalToBinary):**

* Bước 1: Đổi từ hệ thập lục phân sang hệ thập phân bằng cách gọi hàm ‘hexadecimalToDecimal’.
* Bươc 2: Đổi từ hệ thập phân sang hệ nhị phân bằng cách gọi hàm ‘decimalToBinary’.

Tóm lại, hàm `hexadecimalToBinary` chuyển đổi một số thập lục phân thành số nhị phân bằng cách trước tiên chuyển đổi số thập lục phân thành số thập phân, sau đó chuyển đổi số thập phân thành số nhị phân.



Hình 12: Flowchart Hàm chuyển đổi từ thập lúc phân sang nhi phân.

1. CHƯƠNG TRÌNH VÀ KẾT QUẢ
   1. Tổ chức chương trình

* Chương trình được viết trong một hàm menu(), đây là hàm chính của chương trình.
* Hàm menu() hiển thị menu chọn lựa và sử dụng câu lệnh switch-case để thực hiện các phép chuyển đổi tương ứng với lựa chọn của người dùng.
* Các chức năng chuyển đổi được triển khai trong các hàm riêng biệt như string decimalToBinary(double decimal);

string decimalToOctal(double decimal);

string decimalToHexadecimal(double decimal);

double binaryToDecimal(string binary);

double octalToDecimal(string octal);

double hexadecimalToDecimal(string hexadecimal);

string binaryToHexadecimal(string binary);

string binaryToOctal(string binary);

string octalToBinary(string octal);

string hexadecimalToBinary(string hexadecimal);

Chương trình chạy trong một vòng lặp do-while để cho phép người dùng tiếp tục thực hiện các phép chuyển đổi hoặc thoát khỏi chương trình.

* 1. Kết quả
     1. Giao diện chính của chương trình

Giao diện chính của chương trình bao gồm:

1. Tiêu đề chương trình:

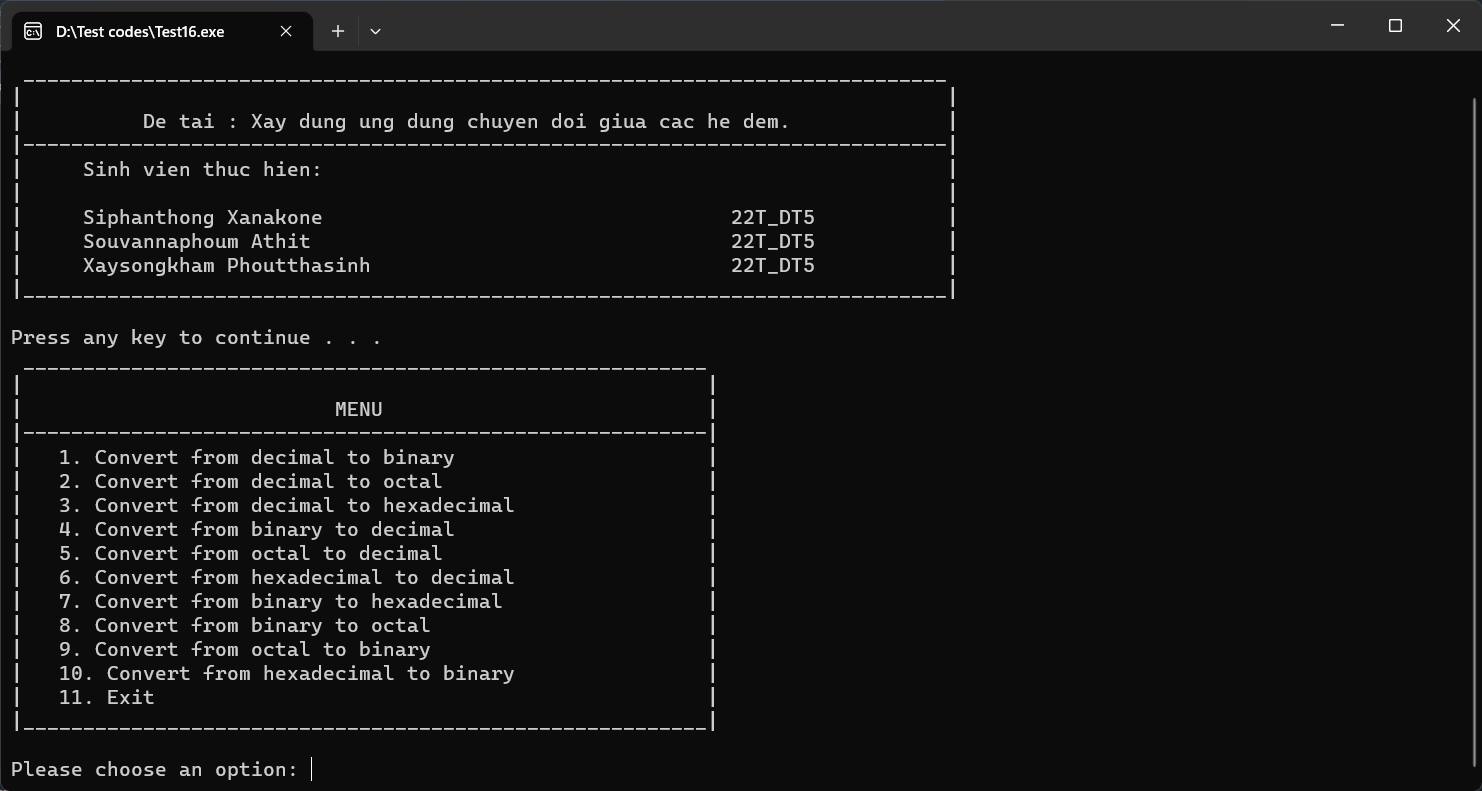
* Hiển thị tiêu đề của chương trình và thông tin về dự án cùng với các sinh viên đã thực hiện.

1. Menu:

* Hiển thị các tùy chọn menu cho người dùng.
* Chấp nhận đầu vào từ người dùng để lựa chọn một tùy chọn.
* Gọi các hàm chuyển đổi tương ứng dựa trên lựa chọn của người dùng.
* Xử lý đầu vào không hợp lệ và yêu cầu người dùng thử lại.

1. Tiếp tục và thoát:

* Sau khi thực hiện một chuyển đổi, chương trình yêu cầu người dùng tiếp tục hoặc thoát.
* Nếu người dùng chọn tiếp tục, menu sẽ được hiển thị lại.
* Nếu người dùng chọn thoát, chương trình hiển thị thông báo thoát và kết thúc.



Hình : Giao diện chính của chương trình.

* + 1. Kết quả thực thi của chương trình

Chương trình lấy các số mà người dùng nhập để tính bằng các hàm chuyển đổi tương ứng và xuất kết quả.

A screenshot of a computer

Description automatically generatedVí dụ:

Hình : Kết của sau chương trình đã chuyển đổi hệ của số.

* + 1. Nhận xét đánh giá

Nhận xét và đánh giá về chương trình:

1. Sự cần thiết và tính đầy đủ: Chương trình đã xây dựng chức năng chuyển đổi giữa các hệ đếm như nhị phân, bát phân và thập lục phân. Ngoài ra, nó cũng hỗ trợ chuyển đổi giữa các hệ đếm khác như từ nhị phân sang thập phân, từ bát phân sang thập phân và từ thập lục phân sang thập phân.
2. Giao diện người dùng: Chương trình có một giao diện menu đơn giản, cho phép người dùng lựa chọn các chức năng chuyển đổi. Mỗi lần sau khi thực hiện một chức năng, người dùng có thể tiếp tục chương trình hoặc thoát.
3. Xử lý lỗi: Chương trình đã xử lý một số lỗi nhập liệu không hợp lệ. Nếu người dùng nhập một lựa chọn không hợp lệ hoặc một số không đúng định dạng, chương trình sẽ thông báo lỗi và yêu cầu nhập lại.
4. Xử lý số âm: Chương trình hỗ trợ chuyển đổi số âm bằng cách đưa dấu trừ (-) vào phần trước số. Ví dụ: -1101 là biểu diễn của số âm trong hệ nhị phân.
5. Độ chính xác: Chương trình xử lý số thực trong phần thập phân của các hệ đếm. Nó sử dụng độ chính xác cao để đảm bảo kết quả chính xác trong phạm vi khả thi.
6. Cấu trúc mã: Mã chương trình được chia thành các hàm nhỏ để thực hiện các chức năng riêng biệt. Điều này giúp tăng tính module và dễ dàng bảo trì và mở rộng mã nguồn.

Nhưng vẫn còn một số phần chưa chính xác như mong muốn. cần phải rút kinh nghiệm và sửa chữa trong tương lai.

Tổng quan, chương trình có một giao diện tương đối tốt, cung cấp các chức năng chuyển đổi giữa các hệ đếm phổ biến và xử lý các trường hợp đặc biệt như số âm. Mã nguồn được tổ chức tốt và dễ hiểu. Tuy nhiên, có thể cải thiện bằng cách thêm xử lý lỗi và thông báo lỗi chi tiết hơn để người dùng biết được lỗi xảy ra và cách sửa chúng.

1. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN
   1. Kết luận

Đây là một chương trình C++ để chuyển đổi giữa các hệ đếm, như chuyển từ thập phân sang nhị phân, bát phân, thập lục phân và ngược lại. Chương trình bao gồm một menu cho phép người dùng chọn loại chuyển đổi người dùng muốn thực hiện.

Dưới đây là một số hàm quan trọng trong chương trình:

1. decimalToBinary: Chuyển đổi số thập phân sang nhị phân.
2. decimalToOctal: Chuyển đổi số thập phân sang bát phân.
3. decimalToHexadecimal: Chuyển đổi số thập phân sang thập lục phân.
4. binaryToDecimal: Chuyển đổi số nhị phân sang thập phân.
5. octalToDecimal: Chuyển đổi số bát phân sang thập phân.
6. hexadecimalToDecimal: Chuyển đổi số thập lục phân sang thập phân.
7. binaryToHexadecimal: Chuyển đổi số nhị phân sang thập lục phân.
8. binaryToOctal: Chuyển đổi số nhị phân sang bát phân.
9. octalToBinary: Chuyển đổi số bát phân sang nhị phân.
10. hexadecimalToBinary: Chuyển đổi số thập lục phân sang nhị phân.

Chương trình có một hàm menu để hiển thị menu và xử lý lựa chọn của người dùng. Người dùng có thể chọn chức năng muốn sử dụng từ menu và thực hiện các chuyển đổi tương ứng.

Các hàm chuyển đổi được viết một cách chi tiết và xử lý các trường hợp đặc biệt như số âm, phần thập phân, số lượng chữ số hợp lệ cho phép và các ký tự không hợp lệ.

Ngoài ra, chương trình cũng có một số mã lệnh liên quan đến hiển thị và xử lý dữ liệu đầu vào/đầu ra trên console.

* 1. Hướng phát triển
* Tối ưu hóa chương trình để tăng tính thú vị và sự hấp dẫn bằng việc phát triển ứng dụng với giao diện đồ họa hấp dẫn và cuốn hút người dùng.
* Nghiên cứu sâu và khắc phục các lỗi tính toán, cải thiện hiển thị kết quả và thông báo để đảm bảo tính chính xác và sự linh hoạt khi gặp lỗi.
* Tìm hiểu và sửa lại cấu trúc chương trình bổ sung.

TÀI LIỆU THAM KHẢO

1. Tài liệu của các thầy cô giáo, trường Đại học Bách Khoa - Đại học Đà Nẵng.
2. Cơ bản của ngôn ngữ C++: <https://www.w3schools.com/cpp/default.asp>
3. Kiến thức về ngôn ngữ C++: <https://vi.wikipedia.org/wiki/C%2B%2B>
4. Làm Flowcharts: <https://www.overleaf.com/read/vjxhrdzbxjtd>

PHỤ LỤC

|  |
| --- |
| #include <iostream>  #include <string>  #include <cmath>  #include <limits>  #include <iomanip>  using namespace std;  // function prototypes  void menu();  string decimalToBinary(double decimal);  string decimalToOctal(double decimal);  string decimalToHexadecimal(double decimal);  double binaryToDecimal(string binary);  double octalToDecimal(string octal);  double hexadecimalToDecimal(string hexadecimal);  string binaryToHexadecimal(string binary);  string binaryToOctal(string binary);  string octalToBinary(string octal);  string hexadecimalToBinary(string hexadecimal);  int main(){  cout << endl;  cout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n";  cout << "| |\n";  cout << "| De tai : Xay dung ung dung chuyen doi giua cac he dem. |\n";  cout << "|-----------------------------------------------------------------------------|\n";  cout << "| Sinh vien thuc hien: |\n";  cout << "| |\n";  cout << "| Siphanthong Xanakone 22T\_DT5 |\n";  cout << "| Souvannaphoum Athit 22T\_DT5 |\n";  cout << "| Xaysongkham Phoutthasinh 22T\_DT5 |\n";  cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";  cout << endl;    system("pause");  // system("cls");  menu();    return 0;    }  void menu(){  int choice;  double decimal;  string binary, octal, hexadecimal;  //system("Color 3");  do{  cout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n";  cout << "| |\n";  cout << "| MENU |\n";  cout << "|---------------------------------------------------------|\n";  cout << "| 1. Convert from decimal to binary |\n";  cout << "| 2. Convert from decimal to octal |\n";  cout << "| 3. Convert from decimal to hexadecimal |\n";  cout << "| 4. Convert from binary to decimal |\n";  cout << "| 5. Convert from octal to decimal |\n";  cout << "| 6. Convert from hexadecimal to decimal |\n";  cout << "| 7. Convert from binary to hexadecimal |\n";  cout << "| 8. Convert from binary to octal |\n";  cout << "| 9. Convert from octal to binary |\n";  cout << "| 10. Convert from hexadecimal to binary |\n";  cout << "| 11. Exit |\n";  cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";  cout << endl;  cout << "Please choose an option: ";  if (cin >> choice) {  switch (choice) {  case 1:  cout << endl;  cout << "Enter decimal number: " /\* << "\033[33m" \*/;  cin >> decimal;  cout << "Binary equivalent: " << decimalToBinary(decimal) << endl;  break;  case 2:  cout << endl;  cout << "Enter decimal number: ";  cin >> decimal;  cout << "Octal equivalent: " << decimalToOctal(decimal) << endl;  break;  case 3:  cout << endl;  cout << "Enter decimal number: ";  cin >> decimal;  cout << "Hexadecimal equivalent: " << decimalToHexadecimal(decimal) << endl;  break;  case 4:  cout << endl;  cout << "Enter binary number: ";  cin >> binary;  cout << "Decimal equivalent: " << binaryToDecimal(binary) << endl;  break;  case 5:  cout << endl;  cout << "Enter octal number: ";  cin >> octal;  if (octal.find('.') == std::string::npos) {  // If there is no decimal point, convert octal to integer  int decimalEquivalent = stoi(octal, 0, 8);  cout << "Decimal equivalent: " << decimalEquivalent << endl;  } else {  cout << "Decimal equivalent: " << octalToDecimal(octal) << endl;  }  break;  case 6:  cout << endl;  cout << "Enter hexadecimal number: ";  cin >> hexadecimal;  try {  if (hexadecimal.find('.') == std::string::npos) {  // If there is no decimal point, convert hexadecimal to integer  int decimal\_int = stoi(hexadecimal, nullptr, 16);  cout << "Decimal equivalent: " << decimal\_int << endl;  } else {  cout << "Decimal equivalent: " << hexadecimalToDecimal(hexadecimal) << endl;  }  } catch (const invalid\_argument& e) {  cout << "Error: " << e.what() << endl;  }  break;  case 7:  cout << endl;  cout << "Enter binary number: ";  cin >> binary;  cout << "Hexadecimal equivalent: " << binaryToHexadecimal(binary) << endl;  break;  case 8:  cout << "Enter binary number: ";  cin >> binary;  cout << "Octal equivalent: " << binaryToOctal(binary) << endl;  break;  case 9:  cout << "Enter octal number: ";  cin >> octal;  cout << "Binary equivalent: " << octalToBinary(octal) << endl;  break;  case 10:  cout << "Enter hexadecimal number: ";  cin >> hexadecimal;  cout << "Binary equivalent: " << hexadecimalToBinary(hexadecimal) << endl;  break;  case 11:  cout << endl;  cout << "\033[31m" << "Exiting program...." << "\033[0m" << endl;  break;  default:  cout << "Invalid choice. Please enter a number from 1 to 11." << endl;  }  }else{  cout << "Invalid input. Please enter a number." << endl;  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  system("pause");  //system("cls");  return menu();  }  if (choice == 11) {  break;  }  cout << endl;  // prompt the user to continue  cout << "Do you want to continue? (y/n): ";  char answer;  cin >> answer;  // check user's answer  if (answer == 'y' || answer == 'Y') {  // user wants to continue, so loop back to the beginning  //system("cls");  continue;  } else if (answer == 'n' || answer == 'N') {  // user wants to exit the program, so break out of the loop  cout << endl;  cout << "\033[31m" <<"Exiting program...." << "\033[0m";  break;  } else {  // user entered an invalid response, so ask again  cout << "Invalid response. Please enter 'y' or 'n'." << endl;    system("pause");  //system("cls");  }  }while (choice != 11);}  // 1 function to convert decimal to binary  string decimalToBinary(double decimal) {  string binary = "";  bool isNegative = false;  if (decimal < 0) {  isNegative = true;  decimal = abs(decimal);  }  // Split decimal into integer and fractional parts  string decimalStr = to\_string(decimal);  size\_t decimalPointPos = decimalStr.find('.');  int integerPart = stoi(decimalStr.substr(0, decimalPointPos));  double fractionalPart = (decimalPointPos == string::npos) ? 0 : stod(decimalStr.substr(decimalPointPos));  // Convert integer part to binary  while (integerPart > 0) {  int remainder = integerPart % 2;  binary = to\_string(remainder) + binary;  integerPart /= 2;  }  // Add decimal point to binary string if fractional part is not zero  if (fractionalPart > 0) {  binary += '.';  // Convert fractional part to binary  int maxFractionalBits = 32; // maximum number of bits for fractional part  while (fractionalPart > 0 && binary.length() < maxFractionalBits) {  fractionalPart \*= 2;  int digit = int(fractionalPart);  binary += to\_string(digit);  fractionalPart -= digit;  }  }  if (isNegative) {  binary = "-" + binary;  }  return binary;  }  // 2 function to convert decimal to octal  string decimalToOctal(double decimal) {  string octal = "";  bool isNegative = false;  if (decimal < 0) {  isNegative = true;  decimal = abs(decimal);  }  // Split decimal into integer and fractional parts  string decimalStr = to\_string(decimal);  size\_t decimalPointPos = decimalStr.find('.');  int integerPart = stoi(decimalStr.substr(0, decimalPointPos));  double fractionalPart = (decimalPointPos == string::npos) ? 0 : stod(decimalStr.substr(decimalPointPos));  // Convert integer part to octal  if (integerPart == 0) {  octal = "0";  } else {  while (integerPart > 0) {  int remainder = integerPart % 8;  octal = to\_string(remainder) + octal;  integerPart /= 8;  }  }  // Add fractional part to octal string if not zero  if (fractionalPart > 0) {  octal += '.';  int maxFractionalDigits = 12; // maximum number of digits for fractional part  for (int i = 0; i < maxFractionalDigits; i++) {  fractionalPart \*= 8;  int digit = int(fractionalPart);  octal += to\_string(digit);  fractionalPart -= digit;  }  // Remove trailing zeros  while (octal.back() == '0') {  octal.pop\_back();  }  // Remove decimal point if no fractional digits  if (octal.back() == '.') {  octal.pop\_back();  }  }  if (isNegative) {  octal = "-" + octal;  }  return octal;  }  // 3 function to convert decimal to hexadecimal  string decimalToHexadecimal(double decimal) {  string hexadecimal = "";  bool isNegative = false;  if (decimal < 0) {  isNegative = true;  decimal = abs(decimal);  }  // Split decimal into integer and fractional parts  string decimalStr = to\_string(decimal);  size\_t decimalPointPos = decimalStr.find('.');  string integerPartStr = decimalStr.substr(0, decimalPointPos);  string fractionalPartStr = (decimalPointPos == string::npos) ? "" : decimalStr.substr(decimalPointPos + 1);  // Convert integer part to hexadecimal  if (integerPartStr == "0") {  hexadecimal = "0";  } else {  long long integerPart = stoll(integerPartStr);  while (integerPart > 0) {  int remainder = integerPart % 16;  if (remainder < 10) {  hexadecimal = to\_string(remainder) + hexadecimal;  } else {  char hexChar = 'A' + remainder - 10;  hexadecimal = hexChar + hexadecimal;  }  integerPart /= 16;  }  }  // Add fractional part to hexadecimal string if not zero  if (!fractionalPartStr.empty()) {  hexadecimal += '.';  double fractionalPart = stod("0." + fractionalPartStr);  int maxFractionalDigits = 6; // maximum number of digits for fractional part  for (int i = 0; i < maxFractionalDigits && fractionalPart > 0; i++) {  fractionalPart \*= 16;  int digit = int(fractionalPart);  if (digit < 10) {  hexadecimal += to\_string(digit);  } else {  char hexChar = 'A' + digit - 10;  hexadecimal += hexChar;  }  fractionalPart -= digit;  }  }  if (isNegative) {  hexadecimal = "-" + hexadecimal;  }  return hexadecimal;  }  // 4 function to convert Binary to Decimal  double binaryToDecimal(string binary) {  bool isNegative = false;  // Check if binary number is negative  if (binary[0] == '-') {  isNegative = true;  binary = binary.substr(1);  }  // Split binary string into integer and fractional parts  size\_t decimalPointPos = binary.find('.');  string integerPartStr = (decimalPointPos == string::npos) ? binary : binary.substr(0, decimalPointPos);  string fractionalPartStr = (decimalPointPos == string::npos) ? "" : binary.substr(decimalPointPos + 1);  // Convert integer part to decimal  double integerPart = 0;  for (int i = 0; i < integerPartStr.length(); i++) {  integerPart += (integerPartStr[i] - '0') \* pow(2, integerPartStr.length() - i - 1);  }  // Convert fractional part to decimal  double fractionalPart = 0;  for (int i = 0; i < fractionalPartStr.length(); i++) {  fractionalPart += (fractionalPartStr[i] - '0') \* pow(2, -(i + 1));  }  double decimal = integerPart + fractionalPart;  if (isNegative) {  decimal = -decimal;  }  return decimal;  }  // 5 Function to convert Octal to Decimal  double octalToDecimal(string octal) {  bool isNegative = false; // Flag to indicate if the octal number is negative  double decimal = 0.0; // Variable to store the decimal value  double base = 1.0; // Base value for converting each digit  // Check if the number is negative  if (octal[0] == '-') {  isNegative = true;  octal = octal.substr(1); // Remove the negative sign from the octal number  }  // Handle integer part  int decimalPointIndex = -1; // Index of the decimal point in the octal number  for (int i = 0; i < octal.length(); i++) {  if (octal[i] == '.') {  decimalPointIndex = i;  break;  }  }  if (decimalPointIndex == -1) {  decimalPointIndex = octal.length(); // If no decimal point is found, consider the entire string as the integer part  }  for (int i = decimalPointIndex - 1; i >= 0; i--) {  decimal += (octal[i] - '0') \* base; // Convert each octal digit to decimal and multiply by the base  base \*= 8.0; // Multiply the base by 8 for each digit  }  // Handle fractional part  if (decimalPointIndex != octal.length()) {  base = 1.0 / 8.0; // Reset the base to handle the fractional part  for (int i = decimalPointIndex + 1; i < octal.length(); i++) {  decimal += (octal[i] - '0') \* base; // Convert each octal digit to decimal and multiply by the base  base /= 8.0; // Divide the base by 8 for each digit to handle the fractional part  }  }  // Handle negative numbers  if (isNegative) {  decimal = -decimal; // If the octal number was originally negative, make the decimal number negative  }  double epsilon = 1.0 / pow(10, numeric\_limits<double>::digits10); // Calculate the smallest possible change in double precision  int maxSignificantDigits = -floor(log10(epsilon) - log10(decimal)); // Calculate the maximum number of significant digits required for precision  cout.precision(numeric\_limits<double>::digits10 - floor(log10(epsilon) - log10(decimal)) + 1); // Set the precision of the decimal value  return decimal; // Return the decimal value  }  // 6 Function to convert hexadecimal to decimal  double hexadecimalToDecimal(string hexadecimal) {  double decimal = 0; // Variable to store the decimal value  double fractional = 0; // Variable to store the fractional part of the decimal value  bool isNegative = false; // Flag to indicate if the hexadecimal value is negative  if (hexadecimal[0] == '-') {  isNegative = true;  hexadecimal = hexadecimal.substr(1); // Remove the negative sign from the hexadecimal value  }  int decimalPointIndex = hexadecimal.find("."); // Find the index of the decimal point in the hexadecimal value  if (decimalPointIndex == -1) {  decimalPointIndex = hexadecimal.length();  }  // Convert the integer part of the hexadecimal value to decimal  for (int i = decimalPointIndex - 1; i >= 0; i--) {  char digit = hexadecimal[i];  if (isdigit(digit)) {  decimal += (digit - '0') \* pow(16, decimalPointIndex - 1 - i);  } else {  digit = toupper(digit);  decimal += (digit - 'A' + 10) \* pow(16, decimalPointIndex - 1 - i);  }  }  // Convert the fractional part of the hexadecimal value to decimal  if (decimalPointIndex < hexadecimal.length()) {  for (int i = decimalPointIndex + 1; i < hexadecimal.length(); i++) {  char digit = hexadecimal[i];  if (isdigit(digit)) {  fractional += (digit - '0') \* pow(16, decimalPointIndex - i);  } else {  digit = toupper(digit);  fractional += (digit - 'A' + 10) \* pow(16, decimalPointIndex - i);  }  }  }  decimal += fractional; // Add the fractional part to the decimal value  double epsilon = 1.0 / pow(10, numeric\_limits<double>::digits10); // Calculate the smallest possible change in double precision  int maxFractionalDigits = -floor(log10(epsilon) - log10(decimal)); // Calculate the maximum number of fractional digits required for precision  if (isNegative) {  decimal = -decimal;  }  // Set the precision of the decimal value  cout.precision(numeric\_limits<double>::digits10 - floor(log10(epsilon) - log10(decimal)) + 1);  return decimal; // Return the decimal value  }  // 7 function to convert binary to hexadecimal  string binaryToHexadecimal(string binary) {  double decimal = binaryToDecimal(binary);  return decimalToHexadecimal(decimal);  }  // 8 function to convert binary to octal  string binaryToOctal(string binary) {  double decimal = binaryToDecimal(binary);  return decimalToOctal(decimal);  }  // 9 function to convert octal to binary  string octalToBinary(string octal) {  double decimal = octalToDecimal(octal);  return decimalToBinary(decimal);  }  // 10 function to convert hexadecimal to binary  string hexadecimalToBinary(string hexadecimal) {  double decimal = hexadecimalToDecimal(hexadecimal);  return decimalToBinary(decimal);  } |