BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

----------ଓ✧ଌ----------

ĐỒ ÁN TIN HỌC

ĐỀ TÀI : PURE PUZZLES

Giáo viên hướng dẫn: TS. Nguyễn Hồng Bửu Long

Sinh viên thực hiện:

MSSV	**Họ tên**	**Lớp**	**Điểm**
DH52111255	Phan Tấn Lộc	D21_TH09	
DH52111186	Âu Dương Thiên Kim	D21_TH10	

Tp.Hồ Chí Minh, ngày 01 tháng 12 năm 2023

Giáo viên chấm điểm

TS Nguyễn Hồng Bửu Long

MỤC LỤC

LỜI CẢM ƠN

Kính thưa thầy Nguyễn Hồng Bửu Long

Nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Nguyễn Hồng Bửu Long đã tận tình hướng dẫn, giúp đỡ chúng em trong suốt quá trình thực hiện Đồ án Tin học với đề tài Các Câu Đố Thuần Túy (Pure Puzzles)

Thầy đã dành rất nhiều thời gian và tâm huyết để hướng dẫn em hoàn thành đồ án này. Thầy đã giúp chúng em định hướng đề tài, tìm kiếm tài liệu, chỉnh sửa và hoàn thiện nội dung đồ án. Thầy cũng đã chia sẻ cho em rất nhiều kinh nghiệm quý báu trong thời gian vừa qua.

Nhờ sự giúp đỡ tận tình của Thầy đã giúp chúng em hoàn thành đồ án một cách tốt nhất. Em xin chân thần cảm ơn Thầy

Em cũng xin gửi lời tới các Thầy/Cô trong khoa Công Nghệ Thông Tin đã tận tình giảng dạy và truyền đạt cho em những kiến thức quý báu.

Em xin chúc Thầy/Cô luôn mạnh khỏe, hạnh phúc, đạt được nhiều thành tựu trong nghiên cứu và thành công trong công tác giảng dạy .

Em xin chân thành cảm ơn!

LỜI MỞ ĐẦU

Trong thời đại công nghệ 4.0 hiện nay, các quốc gia trên thế giới đang hướng đến phát triển công nghệ và tuyệt nhiên công nghệ trở thành một trong các yếu tố để đánh giá một quốc gia có giàu có và hùng mạnh hay không? Tuy là vậy nhưng công nghệ nếu không có con người thì công nghệ cũng sẽ không phát triển và yếu tố con người được đặt trên tất cả. Thế giới càng phát triển đòi hỏi chúng ta phải có những kỹ năng để sử dụng và phát triển theo xu hướng Thế giới. Để làm được điều đó các quốc gia phát triển liên tục đầu tư về giáo dục, đầu tư liên tục về trí tuệ con người.Thế giới bước vào thời đại toàn cầu hóa tri thức và giáo dục. ***Cuộc cạnh tranh chất xám vĩ đại (The Great Brain Race)*** của Giáo Sư Ben Wildavsky đã nói lên bức tranh tổng thể của cuộc cạnh tranh toàn cầu thu hút tài năng trí tuệ sự chạy đua nâng cấp hệ thống giáo dục đại học nhằm tạo ra hiền tài tri thức và tinh hoa xã hội. Tư duy luôn là cụm từ đi đầu trong cuộc cách mạng Công Nghệ và Cuộc cạnh tranh chất xám.

Tư duy là từ không còn xa lạ với mọi người trong cuộc sống và công việc. Cần phải có tư duy để tiến bộ và phát triển. Tuy nhiên đặc biệt trong lĩnh vực công nghệ thông tin thì những người học lập trình luôn tin rằng khả năng tư duy sẽ giúp họ chinh phục lĩnh vực này tốt hơn. Đúng là như vậy, tư duy là điểm cộng cho những người học lập trình đặc biệt là tư duy logic. Tư duy logic là khả năng suy nghĩ một cách có kỷ luật dựa trên các sự kiện và bằng chứng rõ rang. Hiểu một cách đơn giản là kết hợp tính hợp lý vào quá trình suy nghĩ khi phân tích vấn đề để đưa ra giải phải và đòi hỏi sử dụng các kỹ năng lập luận để nghiên cứu một cách khách quan, cân nhắc các dữ kiện, số liệu và đưa ra các quyết định quan trọng dựa trên ưu và nhược điểm. Kỹ năng tư duy logic không bao gồm các yếu tố tình cảm và cảm xúc. Vậy việc rèn luyện tư duy logic như thế nào?

Để rèn luyện, bạn nên chịu khó quan sát, để ý chi tiết và tạo ra thói quen đặt câu hỏi giúp bạn rèn luyện tư duy logic hiệu quả, đặt ra nhiều góc nhìn để đưa ra các phán đoán thực tế. Trong công nghệ thông tin cụm từ Pure Puzzles không còn xa lạ với các lập trình viên. Chúng rèn luyện khả năng tư duy logic và sáng tạo, là những kỹ năng vô cùng quan trọng cần thiết trong lĩnh vực Công nghệ thông tin. Đòi hỏi người ta phải liên tục tập trung suy nghĩ từ đó đưa ra các giải pháp hữu hiệu giải quyết các vấn đề xảy ra. Trong đề tài này hãy tìm hiểu các câu đố thuần túy và bắt đầu xử lý các mã thực tế. Tập trung vào việc giải quyết vấn đề không phải vào các cú pháp lập trình. Đây là cách giải quyết vấn đề thuần túy nhất.

LÝ DO CHỌN C++ LÀ NGÔN NGỮ LẬP TRÌNH CHÍNH TRONG ĐỀ TÀI:

C++ là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, bao gồm hệ điều hành, lập trình ứng dụng, đồ họa máy tính, phát triển web và nhiều lĩnh vực khác

Các ưu điểm của C++ bao gồm:

- **Tốc độ và hiệu suất cao:** C++ là ngôn ngữ lập trình cấp thấp, cho phép lập trình viên kiểm soát chặt chẽ phần cứng máy tính. Điều này dẫn đến tốc độ và hiệu suất cao hơn so với các ngôn ngữ cấp cao hơn
- **Khả năng mở rộng và tùy biến:** C++ là một ngôn ngữ lập trình mạnh mẽ, cho phép lập trình viên tạo ra các ứng dụng phức tạp và tùy biến cao
- **Tính di động :** C++ là ngôn ngữ lập trình di động, có thể được sử dụng trên nhiều nền tảng khác nhau, bao gồm máy tính để bàn, máy tính xác tay,máy tính bảng, điện thoại thông minh và nhiều thiết bị khác
- **Khả năng tương thích:** C++ là ngôn ngữ lập trình tương thích có thể sử dụng để tạo ra các ứng dụng tương thích với các hệ điều hành khác nhau

Sự phổ biến của C++:

- **C++ được sử dụng trong nhiều lĩnh vực khác nhau:** C++ là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, có thể được sử dụng trong nhiều lĩnh vực khác nhau.Điều này đã giúp C++ trở nên phổ biến trong nhiều ngành công nghiệp khác nhau
- **C++ có nhiều thư viện và công cụ hỗ trợ:** C++ có nhiều thư viện và công cụ hỗ trợ, giúp lập trình viên C++ dễ dàng thực hiện các nhiệm vụ phức tạp. Điều này đã giúp C++ trở nên hiệu quả và dễ sử dụng hơn

C++ là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, có thể được sử dụng để tạo ra các ứng dụng hiệu quả và tùy biến cao. Các ưu điểm của C++ và một cộng đồng người dùng lớn và tích cực đã giúp C++ trở thành một ngôn ngữ phổ biến trong nhiều lĩnh vực khác nhau.

Mong muốn tìm hiểu sâu vào ngôn ngữ C++ và tiếp cận nhiều thử thách, câu hỏi , bài tập mang tính nâng cao kiến thức về ngôn ngữ. Để phát triển các tư duy lập trình, phát triển các kỹ năng mà một lập trình viên cần có. Thúc đẩy việc động não và rèn luyện tư duy giải quyết các vấn đề qua ngôn ngữ C++. Có thể hạn chế tối đa việc sai phạm trong quá trình làm đồ án.

CHƯƠNG I . TÌM HIỂU VỀ INPUT PROCESSING

1.INPUT PROCESSING: là quá trình xử lý dữ liệu đầu vào. Đây là một bước quan trọng trong quá trình xử lí thông tin. Quá trình này có nhiệm vụ chuyển đổi dữ liệu đầu vào từ dạng thô chuẩn hóa thành định dạng phù hợp mà hệ thống có thể xử lý được.

2.CÁC BƯỚC XỬ LÝ DỮ LIỆU ĐẦU VÀO

- Bao gồm các bước sau:

- Đọc dữ liệu đầu vào: Dữ liệu đầu vào có thể được nhập từ nhiều nguồn khác nhau như bàn phím, chuột, thiết bị cảm biến, ... Quá trình này sẽ chuyển đổi dữ liệu từ dạng thô sang dạng nhị phân, dạng mà máy tính có thể hiểu được.
- Kiểm tra dữ liệu đầu vào: Sau khi dữ liệu đầu vào được đọc, hệ thống cần kiểm tra và đảm bảo rằng dữ liệu đầu vào là hợp lệ. Điều này nhằm đảm bảo rằng dữ liệu đầu vào là đầy đủ, chính xác và phù hợp với yêu cầu của hệ thống.
- Chuyển đổi dữ liệu đầu vào: Dữ liệu đầu vào sau khi được kiểm tra cần được chuyển đổi sang dạng mà hệ thống có thể xử lý được. Ví dụ, dữ liệu đầu vào là văn bản cần được chuyển đổi thành mã ASCII hoặc Unicode.
- Tách biệt dữ liệu: Bước này tách dữ liệu đầu vào thành các thành phần riêng lẻ để xử lý. Ví dụ, dữ liệu đầu vào có thể được tách thành các trường hoặc bản ghi.

Sử dụng biến để lưu trữ đầu vào: Đây là cách đơn giản nhất để xử lý đầu vào.

Ví dụ:

Code C++

```
int so;
cout<< “Nhap so nguyen:”;
cin>>so;
if(so>0){
cout<< “\n So ban vua nhap la so duong,”;
}
else if(so<0){
cout<< “\n So ban vua nhap la so am.”;
}
else {
cout<< “\n So ban vua nhap la 0.”;
}
return 0;
```

Trong ví dụ trên, chúng ta sử dụng kiểu dữ liệu int () để đọc số nhập từ người dùng. Sau đó, chúng ta sử dụng lần lượt if, else, else if để kiểm tra số có thỏa điều kiện hay không và cuối cùng in ra câu lệnh tương thích.

Các chương trình trước đây chỉ tạo ra đầu ra. Chúng ta hãy thay đổi một chút và thử các chương trình tập trung vào việc xử lý đầu vào. Mỗi chương trình này đều có một hạn chế: Đầu vào sẽ được đọc từng ký tự một, và chương trình phải xử lý từng ký tự trước khi đọc ký tự tiếp theo. Nói cách khác, các chương trình sẽ không lưu trữ các ký tự trong cấu trúc dữ liệu để xử lý sau mà sẽ xử lý khi chúng đi qua.

Trong vấn đề đầu tiên này, chúng ta sẽ thực hiện xác thực số nhận dạng. Trong thế giới hiện đại, hầu hết mọi thứ đều có số nhận dạng, chẳng hạn như ISBN hoặc số khách hàng. Đôi khi những số này phải được nhập bằng tay, điều này dẫn đến khả năng mắc lỗi. Nếu một số được nhập nhầm không khớp với bất kỳ số nhận dạng hợp lệ nào, hệ thống có thể dễ dàng từ chối nó. Nhưng điều gì sẽ xảy ra nếu số đó sai nhưng vẫn hợp lệ? Ví dụ: nếu một nhân viên thu ngân, cố gắng ghi có vào tài khoản của bạn cho một sản phẩm được trả lại, nhập số tài khoản của khách hàng khác? Khách hàng khác sẽ nhận được khoản tín dụng của bạn.

Để tránh tình trạng này, các hệ thống đã được phát triển để phát hiện lỗi trong số nhận dạng. Chúng hoạt động bằng cách chạy số nhận dạng qua một công thức tạo ra một hoặc nhiều chữ số bổ sung, trở thành một phần của số nhận dạng mở rộng. Nếu bất kỳ chữ số nào được thay đổi, phần gốc của số và các chữ số bổ sung sẽ không còn khớp, và số có thể bị từ chối.

CHƯƠNG II. PROBLEMS CỦA INPUT PROCESSING

1. PROBLEM: LUHN CHECKSUM VALIDATION

LUHN CHECKSUM VALIDATION hay phương pháp Modulus 10, là một thuật toán đơn giản được sử dụng để kiểm tra tính hợp lệ của một số nhận dạng, thường là số thẻ tín dụng, số điện thoại, và các số định danh tương tự. Thuật toán đơn giản nhưng hiệu quả để kiểm tra tính hợp lệ của số. Thuật toán này được phát triển bởi Hans Peter Luhn, một nhà khoa học máy tính người Đức làm việc tại IBM vào năm 1954.

Các bước thực hiện thuật toán Luhn như sau:

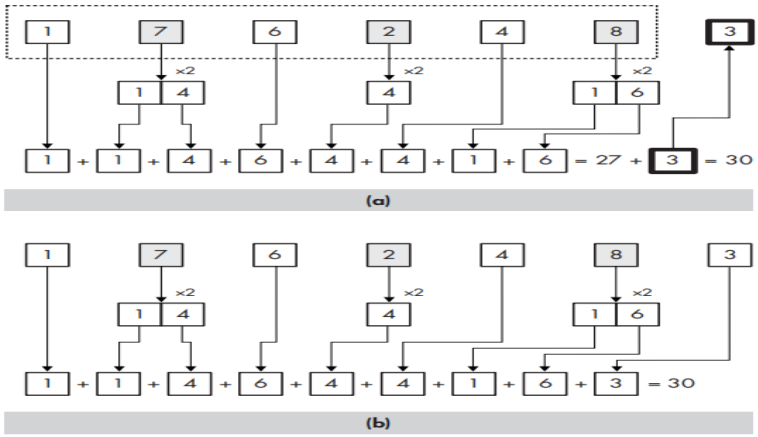
Bước 1: Nhân đôi giá trị của những số ở vị trí chẵn tính từ phải sang.

Bước 2: Cộng dồn tất cả các chữ số riêng lẻ của các số thu được ở bước 1, cùng với các số ở vị trí lẻ.

Bước 3: Nếu kết quả ở bước 2 là một số chia hết cho 10 thì số A sẽ bằng 0. Nếu kết quả ở bước 2 không chia hết cho 10 thì A sẽ bằng số chia hết cho 10 lớn hơn gần nhất trừ đi chính kết quả đó, tức là tổng các số kể cả A phải chia hết cho 10.

Thuật toán Luhn là một phương pháp kiểm tra tính hợp lệ đơn giản và hiệu quả. Tuy nhiên, thuật toán này không phải là một phương pháp mã hóa an toàn và không thể ngăn chặn tất cả các hành vi gian lận.

Ví dụ: Giải thích: Đây là 1 chương trình chỉ xác thực một số nhận dạng, không phải tạo chữ số kiểm tra. Đi qua cả hai đầu của quá trình: Comput sẽ đánh dấu số kiểm tra và xác thực kết quả.



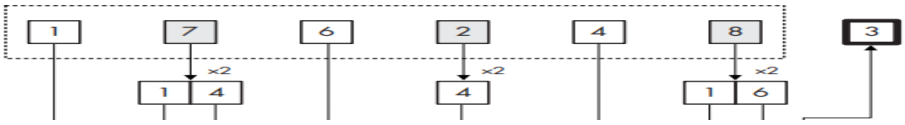
Hình 2-3: Công thức tổng kiểm tra Luhn

Ví dụ: Hình 2-3.

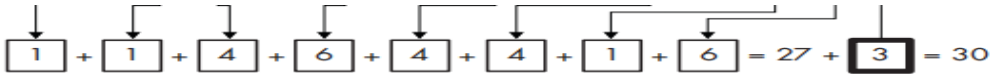
Trong phần (a), tính toán chữ số kiểm tra.

- Bản gốc identifica Số tion, 176248, được hiển thị trong hình chữ nhật đứt nét.

- Bắt đầu từ chữ số phải nhất sau khi thêm chữ số kiểm tra thì số ngoài cùng bên phải thứ hai(vị trí chẵn) sẽ được nhân đôi và cho ra kết quả, lưu ý là kết quả này sẽ được ghi chú dưới dưới dạng chia làm 2 chữ số được xét riêng sau khi ghi chú dưới dạng này mỗi chữ số sẽ được đem cộng với nhau.

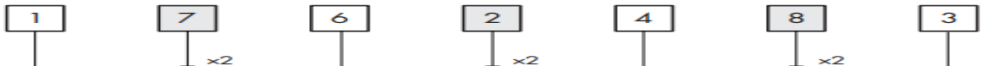


- Chẳng hạn khi 7 được nhân đôi tạo ra 14, nó không phải là 14 mà nó sẽ được thêm vào dưới dạng 1 và 4 riêng lẻ. Trong trường hợp này, tổng kiểm tra là 27, vì vậy chữ số kiểm tra là 3 vì đó là giá trị chữ số sẽ làm cho tổng số 30. Lưu ý là tổng kiểm tra của số cuối cùng phải chia hết cho 10; trong Nói cách khác, nó sẽ kết thúc bằng 0.

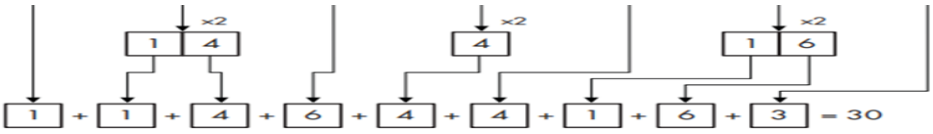


Trong phần (b)

- Xác thực số 1762483.



- Kiểm tra chữ số. Nhân đôi mỗi chữ số thứ hai: bắt đầu bằng chữ số ở bên phải và thêm giá trị của tất cả các chữ số, bao gồm cả chữ số kiểm tra, để xác định tổng kiểm tra. Vì tổng kiểm tra chia hết cho 10, số này xác thực.



Breaking Down the Problem (phân tích vấn đề)

Breaking Down the Problem (BTD) là một quá trình trong quá trình giải quyết vấn đề, trong đó vấn đề lớn và phức tạp được chia thành các phần nhỏ hơn và dễ giải quyết hơn. Quá trình này giúp làm rõ mục tiêu và tạo ra kế hoạch hành động cụ thể. Hữu ích để giải quyết bất kỳ vấn đề nào có vẻ quá sức hoặc khó giải quyết.

Việc phân tích vấn đề giúp mô phỏng và hiểu rõ hơn về cấu trúc, các yếu tố và mối quan hệ của vấn đề, từ đó tạo nên một hình dung toàn diện và khả thi về cách giải quyết vấn đề.

Quá trình phân tích vấn đề thường bao gồm các bước sau:

1. Xác định vấn đề: Định rõ vấn đề cần giải quyết và mục tiêu cụ thể mà bạn muốn đạt được.

2. Phân chia thành phần vấn đề: Chia vấn đề lớn thành các phần nhỏ hơn, thường gọi là các phần tử con, tiến hành phân tích từng phần tử đó.

3. Đối với mỗi phần tử con: Xác định thứ tự mà bạn sẽ giải quyết từng phần tử con, tiến hành phân tích chi tiết hơn để hiểu các yếu tố, quan hệ và khả năng giải quyết.

4. Xác định kế hoạch hành động: Dựa trên phân tích của các phần tử con, xác định các bước cụ thể, tài nguyên và thời gian để giải quyết từng phần của vấn đề.

⇨ Breaking Down the Problem giúp giảm độ phức tạp của vấn đề ban đầu bằng cách chia thành các phần nhỏ hơn, dễ quản lý hơn và tìm ra các cách giải quyết hiệu quả cho từng phần. Điều này thường làm cho quá trình giải quyết vấn đề trở nên rõ ràng hơn và tăng khả năng thành công.

Ví dụ:

Trong hình cho thấy một chương trình tính tổng của tất cả các số chẵn từ 1 đến 100.

```
int sum=0;
for(int i=1;i<=100;i++){
if( i % 2 == 0) {
sum += i;
}
}
cout<< " Tong cua tat ca cac so chan tu 1 den 100 la:" <<sum<< endl;
return 0;
}
```

Phân tích

- Dòng 1: Khởi tạo biến sum là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 2: Lặp qua các số từ 1 đến 100, gán mỗi số cho biến i.
- Dòng 3: Kiểm tra xem số hiện tại i có phải là số chẵn hay không bằng cách sử dụng toán tử modulo (%). Nếu số dư của phép chia i cho 2 bằng 0, thì số đó là số chẵn.
- Dòng 4: Nếu i là số chẵn, thì bình phương số đó bằng i và cộng vào tổng sum bằng cách sử dụng toán tử cộng gán (+=).
- Dòng 7: Hiển thị tổng cuối cùng bằng cách sử dụng hàm cout.
- Dòng 8: Trả về giá trị 0 để biểu thị rằng chương trình đã thực thi thành công.

KQ: Tong=2550

Ví dụ 1: Trong hình là một chương trình nhận một số có một chữ số và chuyển đổi nó thành một số có hai chữ số.

```
int digit;
cout << "Enter a single digit number, 0-9: ";
cin >> digit;
❶ int doubledDigit = digit * 2;
int sum;
❷ if (doubledDigit >= 10) sum = ❸1 + doubledDigit % 10;
else sum = doubledDigit;
❹ cout << "Sum of digits in doubled number: " << sum << "\n";
```

Phân tích

- Dòng 1: Khai báo biến digit là một số nguyên.
- Dòng 2: Hiển thị lời nhắc nhập.
- Dòng 3: Nhận đầu vào từ người dùng.
- Dòng 4: Nhân số với 2.
- Dòng 5: Khai báo biến sum là một số nguyên.
- Dòng 6: Kiểm tra xem số nhân đôi có lớn hơn hoặc bằng 10 hay không.
- Dòng 7: Nếu số nhân đôi lớn hơn hoặc bằng 10, thì tổng các chữ số là 1 cộng với số dư của số nhân đôi chia cho 10.
- Dòng 8: Nếu số nhân đôi nhỏ hơn 10, thì tổng các chữ số là chính số nhân đôi.
- Dòng 9: Hiển thị tổng các chữ số trong số nhân đôi.

TD: Nếu người dùng nhập số 4, thì chương trình sẽ in ra số 8, vì 4 x 2 = 8 , sum 1+8% 10 = 9.

Ví dụ 2:

Trong hình cho thấy một hàm có tên doubledigitvalue(). Hàm này nhận một số nguyên digit làm đầu vào và trả về một số nguyên là tổng của số nhân đôi của digit.

```
int doubleDigitValue(int digit) {
   int doubledDigit = digit * 2;
   int sum;
   if (doubledDigit > 10) sum = 1 + doubledDigit % 10;
   else sum = doubledDigit;
   return sum;
}
```

Phân tích

- Dòng 1: Khai báo hàm doubleDigitValue (). Hàm này nhận một số nguyên digit làm đầu vào và trả về một số nguyên.
- Dòng 2: Nhân số với 2.
- Dòng 3: Khai báo biến sum là một số nguyên.
- Dòng 4: Kiểm tra xem số nhân đôi có lớn hơn 10 hay không.

- Dòng 5: Nếu số nhân đôi lớn hơn 10, thì tổng các chữ số là 1 cộng với số dư của số nhân đôi chia cho 10.
- Dòng 6: Nếu số nhân đôi nhỏ hơn 10, thì tổng các chữ số là chính số nhân đôi.
- Dòng 7: Trả về tổng các chữ số.

TD: Nếu người dùng nhập số 4, thì chương trình sẽ in ra số 8, vì 4 x 2 = 8 và sum 1+8% 10 = 9.

Ví dụ 3:

Trong hình cho thấy một chương trình nhận một số có một chữ số từ người dùng và kiểm tra xem tổng các chữ số của số đó có bằng 55 hay không.

```
   char digit;
   cout << "Enter a one-digit number: ";
❶  digit = cin.get();
   int sum = digit;
   cout << "Is the sum of digits " << sum << "? \n";

❷  Enter a one-digit number: 7
   Is the sum of digits 55?
```

Phân tích

- Dòng 1: Khai báo biến digit là một ký tự.
- Dòng 2: Hiển thị lời nhắc nhập.
- Dòng 3: Nhận đầu vào từ người dùng.
- Dòng 4: Tính tổng các chữ số của số nhập vào.
- Dòng 5: Kiểm tra xem tổng các chữ số có bằng 55 hay không.
- Dòng 6: Nếu tổng các chữ số bằng 55, thì hiển thị kết quả.
- Dòng 7: Nếu tổng các chữ số không bằng 55, thì hiển thị kết quả.

TD: Nếu người dùng nhập số 4, thì chương trình sẽ in ra số 4, sum=4 và 4 không bằng 55 => Tổng các chữ số không bằng 55

2.PROBLEM: CONVERT CHARACTER DIGIT TO INTEGER

Viết một chương trình đọc một ký tự đại diện cho một chữ số (từ 0-9). Chuyển đổi ký tự thành số nguyên tương đương trong phạm vi 0–9, sau đó xuất ra số nguyên để chứng minh kết quả.

Trong các bài toán về hình dạng của phần trước, chúng ta đã có một biến với
Một phạm vi giá trị mà chúng tôi muốn chuyển đổi thành một phạm vi giá trị khác. Chúng ta tạo một bảng với các cột cho các giá trị ban đầu và các giá trị mong muốn và

Sau đó kiểm tra sự khác biệt giữa hai. Đây là một vấn đề tương tự,
và chúng ta có thể sử dụng lại ý tưởng bảng, như trong Bảng 2-2.

Có một số cách tiếp cận để giải quyết vấn đề này, nhưng phương pháp đơn giản nhất liên quan đến việc sử dụng các giá trị ASCII của các ký tự. ASCII (American Standard Code for Information Interchange) là một tiêu chuẩn mã hóa ký tự gán một giá trị số duy nhất cho mỗi ký tự. Đối với chữ số, các giá trị ASCII dao động từ 48 đến 57, với '0' có giá trị là 48 và '9' có giá trị là 57.

Để chuyển đổi một ký tự chữ số thành số nguyên, bạn có thể chỉ cần trừ giá trị ASCII của '0' khỏi giá trị ASCII của ký tự. Điều này thực sự loại bỏ độ lệch 48, dẫn đến giá trị số nguyên tương ứng.

Table 2-2: Character Codes and Desired Integer Values

Character	Character Code	Desired Integer	Difference
0	48	0	48
1	49	1	48
2	50	2	48
3	51	3	48
4	52	4	48
5	53	5	48
6	54	6	48
7	55	7	48
8	56	8	48
9	57	9	48

Bảng mã ký tự là bảng quy định một số nguyên cho mỗi ký tự. Trong bảng mã ASCII, ký tự 0 được quy định bởi số nguyên 48. Giá trị số nguyên mong muốn là giá trị số nguyên của ký tự 0 trong bảng mã ASCII. Trong trường hợp này, giá trị số nguyên mong muốn là 48.

Vậy bảng 2.2 “Ý nghĩa của bảng mã ký tự và giá trị số nguyên mong muốn” có nghĩa là:

- Bảng mã ký tự quy định một số nguyên cho mỗi ký tự.
- Giá trị số nguyên mong muốn là giá trị số nguyên của ký tự được yêu cầu trong bảng mã.

Trong bảng Dòng đầu tiên của bảng mã ký tự cho biết rằng ký tự 0 được quy định bởi số nguyên 48. Điều này có nghĩa là khi chúng ta gặp ký tự 0 trong chương trình của mình, chúng ta có thể sử dụng số nguyên 48 để đại diện cho ký tự đó.

Ví dụ 1:

Trong hình là code tính tổng các chữ số của một số một chữ số.

```
char digit;
cout << "Enter a one-digit number: ";
cin >> digit;
int sum = digit - '0';
cout << "Is the sum of digits " << sum << "? \n";
```

Phân tích

- Dòng thứ 1: Khai báo biến digit kiểu char.
- Dòng thứ 2: In ra màn hình lời nhắc người dùng nhập một số một chữ số.
- Dòng thứ 3: Sử dụng toán tử >> để đọc một ký tự từ bàn phím và lưu trữ nó vào biến digit.
- Dòng thứ 4: Chuyển ký tự digit thành số nguyên bằng cách trừ đi ký tự '0'.
- Dòng thứ 5: In ra màn hình tổng các chữ số.
- Dòng thứ 6: Trả về giá trị 0 để kết thúc chương trình.

TD: + Nhap so 1 chu so: 4

+ Tong cac chu so la: 4

3.PROBLEM: LUHN CHECKSUM VALIDATION, FIXED LENGTH

LUHN CHECKSUM VALIDATION, FIXED LENGTH là một phương pháp xác minh tính hợp lệ của một số, chẳng hạn như số thẻ tín dụng hoặc số ID. Phương pháp này dựa trên thuật toán Luhn, là một công thức kiểm tra chẵn lẻ đơn giản thường được sử dụng trong ngành tài chính.

FIXED LENGTH nghĩa là số ký tự trong chuỗi cần được kiểm tra là một giá trị cố định. Điều này có nghĩa là chuỗi đầu vào phải có một số ký tự cụ thể và không được phép thêm hoặc bớt bất kỳ ký tự nào.

Ví dụ, số thẻ tín dụng thường có 16 chữ số. Nếu số cần được xác minh không có đúng số chữ số, thì nó sẽ tự động được coi là không hợp lệ.

Thuật toán Luhn bản thân là một thuật toán tương đối đơn giản có thể dễ dàng triển khai trong phần mềm.

Xác minh kiểm tra chẵn lẻ Luhn với độ dài cố định là một phương pháp rất hiệu quả để xác minh tính hợp lệ của các số. Đây là một thuật toán đơn giản và đáng tin cậy, dễ triển khai và sử dụng. Đây cũng là một phương pháp rất phổ biến để xác minh tính hợp lệ của các số tài chính, chẳng hạn như số thẻ tín dụng và số ID.

- Chương trình lấy số nhận dạng (bao gồm cả chữ số kiểm tra của nó) có độ dài sáu và xác định xem số đó có hợp lệ theo công thức Luhn hay không. Chương trình phải xử lý từng

ký tự trước khi đọc ký tự tiếp theo. Như trước đây, chúng ta có thể giảm hơn nữa để bắt đầu dễ dàng nhất có thể. Điều gì sẽ xảy ra nếu chúng ta thay đổi công thức để không có chữ số nào được nhân đôi? Sau đó, chương trình chỉ phải đọc các chữ số và tính tổng chúng.

Ví dụ:

```
string id;
cout<< "Nhap so nhan dang:";
cin>>id;
if(id.length() != 6){
cout<< "So khong hop le!!! So phai co do dai la 6."<<end;
return 1;
}
int checksum = 0;
for(int i= 0; i<id.length();i++){
checksum += id[i] - '0';
}
if(checksum % 10 !=0){
cout<< "So khong hop le.";
}
else{
cout<< " So hop le.";
}
return 0;
}
```

Phân tích

- Dòng thứ 1: Khai báo biến id kiểu string.
- Dòng 2: Hiển thị lời nhắc nhập.
- Dòng 3: Nhận đầu vào từ người dùng.
- Dòng 4-7: Kiểm tra độ dài của số nhận dạng. Nếu id có độ dài không phải là 6, chương trình sẽ in ra thông báo "Số không hợp lệ!!! Số phải có độ dài là 6." và kết thúc. Ngược lại sẽ tính tổng các chữ số của id.
- Dòng 8: Khởi tạo biến checksum (tổng) là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 9-10: Tính tổng các chữ số của id.
- Dòng 12-18: kiểm tra xem tổng các chữ số có chia hết cho 10 hay không. Nếu tổng các chữ số chia hết cho 10, chương trình sẽ in ra thông báo "Số hợp lệ!". Ngược lại, chương trình sẽ in ra thông báo "Số không hợp lệ.".
- Dòng 19: Trả về giá trị 0 để biểu thị rằng chương trình đã thực thi thành công.

LUHN CHECKSUM VALIDATION, FIXED LENGTH là một phương pháp hiệu quả để xác minh tính hợp lệ của các số. Phương pháp này dựa trên thuật toán Luhn đơn giản và đáng tin cậy.

4.PROBLEM: SIMPLE CHECKSUM VALIDATION, FIXED LENGTH

- SIMPLE CHECKSUM VALIDATION, FIXED LENGTH (Xác minh tổng kiểm tra đơn giản, độ dài cố định) là một phương pháp xác minh tính hợp lệ của một số, chẳng hạn như số thẻ tín dụng hoặc số ID. Phương pháp này dựa trên việc sử dụng tổng kiểm tra, là một phép tính đơn giản được thực hiện trên các chữ số của số để xác định xem số có hợp lệ hay không.

FIXED LENGTH có nghĩa là số cần được xác minh phải có một số chữ số cụ thể. Ví dụ, số thẻ tín dụng thường có 16 chữ số. Nếu số cần được xác minh không có đúng số chữ số, thì nó sẽ tự động được coi là không hợp lệ.

Có nhiều phương pháp xác minh tổng kiểm tra khác nhau, nhưng một phương pháp phổ biến là ***méthode de la somme des chiffres pondérée*** (phương pháp tổng các chữ số có trọng số). Phương pháp này dựa trên việc gán trọng số cho từng chữ số trong số và sau đó cộng các chữ số được trọng số lại. Kết quả của phép cộng này sau đó được so sánh với một số kiểm tra, là một chữ số được tính toán dựa trên các chữ số của số. Nếu kết quả của phép cộng bằng với số kiểm tra, thì số được coi là hợp lệ. Nếu không, số được coi là không hợp lệ.

- Chương trình lấy số nhận dạng (bao gồm cả chữ số kiểm tra của nó) độ dài sáu và xác định xem số có hợp lệ theo một công thức đơn giản trong đó Các giá trị của mỗi chữ số được tính tổng và kết quả được kiểm tra để xem nó có phải là Divisi - không bằng 10. Chương trình phải xử lý từng ký tự trước khi đọc ký tự tiếp theo. Bởi vì chúng ta biết cách đọc một chữ số riêng lẻ như một ký tự, chúng ta có thể Giải quyết vấn đề tổng kiểm tra đơn giản, độ dài cố định này khá dễ dàng. Chúng tôi chỉ cần Để đọc sáu chữ số, tính tổng chúng và xác định xem tổng có chia hết hay không bởi 10.

```
char digit;
int checksum = 0;
cout << "Enter a six-digit number: ";
for (int position = 1; position <= 6; position ++) {
   cin >> digit;
   checksum += digit - '0';
}
cout << "Checksum is " << checksum << ". \n";
if (checksum % 10 == 0) {
   cout << "Checksum is divisible by 10. Valid. \n";
} else {
   cout << "Checksum is not divisible by 10. Invalid. \n";
}
```

Phân tích:

- Dòng thứ 1: Khai báo biến digit kiểu char.

- Dòng 2: Khởi tạo biến checksum là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 3: Hiển thị lời nhắc nhập.
- Dòng 4: Chạy vòng lặp for sử dụng vòng lặp for để duyệt qua từng chữ số của số sáu chữ số.
- Dòng 5: Nhận đầu vào từ người dùng.
- Dòng 6: Tính tổng các chữ số của id.
- Dòng 9: Kiểm tra xem tổng các chữ số có chia hết cho 10 hay không. Nếu tổng các chữ số chia hết cho 10, chương trình sẽ in ra thông báo "Tổng kiểm tra chia hết cho 10. Hợp lệ". Ngược lại, chương trình sẽ in ra thông báo " Tổng kiểm tra chia hết cho 10. Không hợp lệ.".

Từ đây, chúng ta cần thêm logic cho công thức xác thực Luhn thực tế, có nghĩa là nhân đôi mọi chữ số khác bắt đầu từ chữ số thứ hai từ bên phải.
Logic cho công thức xác thực Luhn thực tế như sau:

- Bắt đầu từ chữ số thứ hai từ bên phải, nhân từng chữ số lẻ với 2.
- Nếu kết quả nhân là số có hai chữ số, thì cộng tổng của hai chữ số đó.
- Cộng tổng các chữ số vừa nhân và không nhân.
- Nếu tổng chia hết cho 10, thì số đó hợp lệ.

Vì chúng tôi hiện đang giới hạn bản thân ở các số có sáu chữ số, Chúng ta cần nhân đôi các chữ số ở các vị trí một, ba và năm, tính từ bên trái. Nói cách khác, chúng tôi nhân đôi chữ số nếu vị trí là lẻ. Chúng ta có thể xác định các vị trí lẻ và chẵn bằng toán tử modulo (%) vì định nghĩa của một số chẵn là nó chia đều cho hai. Vì vậy, nếu kết quả Trong số vị trí biểu thức % 2 là 1, vị trí là lẻ và chúng ta nên nhân đôi.
Đó là Điều quan trọng cần nhớ là nhân đôi ở đây có nghĩa là nhân đôi chữ số riêng lẻ và cũng tính tổng các chữ số của số được nhân đôi nếu nhân đôi kết quả là số 10 trở lên. Đây là nơi chức năng trước đây của chúng tôi thực sự Giúp. Khi chúng ta cần nhân đôi một chữ số theo công thức Luhn, chúng ta Chỉ cần gửi nó đến hàm của chúng tôi và sử dụng kết quả. Đặt điều này lại với nhau, chỉ Thay đổi mã bên trong vòng lặp For từ danh sách trước:

```
for (int position  = 1; position  <= 6; position++) {
   cin >> digit;
   if (position % 2 == 0) checksum += digit - '0';
      else checksum += doubleDigitValue(digit - '0');
}
```

Phân tích

- Dòng 1: Chạy vòng lặp for sử dụng vòng lặp for để duyệt qua từng chữ số của số sáu chữ số.
- Dòng 2: Nhận đầu vào từ người dùng.
- Dòng 3-4: Kiểm tra Nếu vị trí của chữ số đó là chẵn, thì giá trị của chữ số đó được cộng trực tiếp vào checksum. Ngược lại, nếu vị trí là lẻ, thì giá trị của chữ số sau khi nhân đôi (theo thuật toán Luhn) được cộng vào checksum.

Hàm doubleDigitValue (): thường được sử dụng trong các ứng dụng cần nhân một chữ số với 2 và cộng tổng của hai chữ số đó nếu cần thiết. Một ví dụ điển hình là ứng dụng xác thực số thẻ tín dụng theo công thức Luhn.

Chúng tôi đã hoàn thành rất nhiều về vấn đề này cho đến nay, nhưng vẫn còn một vài bước trước khi chúng tôi có thể viết mã cho các số iden-tification có độ dài tùy ý. Để cuối cùng giải quyết vấn đề này, chúng ta cần phải phân chia và chinh phục. Giả sử tôi yêu cầu bạn sửa đổi mã trước đó cho các số có 10 hoặc 16 chữ số. Điều đó sẽ rất tầm thường — bạn chỉ phải thay đổi 6 được sử dụng làm cận trên của vòng lặp thành một giá trị khác. Nhưng giả sử tôi yêu cầu bạn xác thực các số có bảy chữ số. Điều đó sẽ đòi hỏi một sửa đổi bổ sung nhỏ bởi vì nếu số chữ số là lẻ và chúng tôi đang nhân đôi mỗi chữ số bắt đầu từ chữ số thứ hai bên phải, chữ số đầu tiên bên trái không còn được nhân đôi. Trong trường hợp này, bạn cần tăng gấp đôi các vị trí chẵn: 2, 4, 6, v.v. Tạm gác vấn đề đó sang một bên, chúng ta hãy tìm hiểu cách xử lý bất kỳ số có độ dài chẵn nào. Vấn đề đầu tiên chúng tôi phải đối mặt là xác định khi nào chúng tôi đã kết thúc số. Nếu người dùng nhập một số có nhiều chữ số và nhấn ENTER và chúng tôi đang đọc ký tự đầu vào theo ký tự, ký tự nào được đọc sau chữ số cuối cùng? Điều này thực sự thay đổi dựa trên hệ điều hành, nhưng chúng tôi sẽ chỉ viết một thử nghiệm:

```
cout << "Enter a number: ";
char digit;
while (true) {
   digit = cin.get();
   cout << int(digit) << " ";
}
```

Phân tích:

- Dòng 1: Hiển thị lời nhắc nhập 1 số.
- Dòng 2: Khai báo biến digit kiểu char.
- Dòng 3: Chạy vòng lặp while(true).
- Dòng 4: Gán giá trị của ký tự được nhập từ đầu vào tiêu chuẩn cho biến digit.
- Dòng 5: Chuyển đổi ký tự chữ số thành số nguyên và in ra.

Vòng lặp này chạy mãi mãi, nhưng nó thực hiện công việc. Tôi gõ số 1234 và nhấn ENTER. Kết quả là 49 50 51 52 10 (dựa trên ASCII; điều này sẽ thay đổi tùy theo hệ điều hành). Vì vậy, 10 là những gì tôi đang tìm kiếm. Với thông tin đó trong tay, chúng ta có thể thay thế vòng lặp for trong code trước đó bằng một vòng lặp while:

```
   char digit;
   int checksum = 0;
❶  int position = 1;
   cout << "Enter a number with an even number of digits: ";
❷  digit = cin.get();
   while ❸(digit != 10) {
     ❹if (position % 2 == 0) checksum += digit - '0';
       else checksum += 2 * (digit - '0');
     ❺digit = cin.get();
     ❻position++;
   }
   cout << "Checksum is " << checksum << ". \n";
   if (checksum % 10 == 0) {
      cout << "Checksum is divisible by 10. Valid. \n";
   } else {
      cout << "Checksum is not divisible by 10. Invalid. \n";
   }
```

Trong đoạn code này, position không còn là biến control trong một vòng lặp for, vì vậy chúng ta phải khởi tạo (1) và tăng nó một cách riêng biệt (6). Vòng lặp bây giờ được điều khiển bởi biểu thức có điều kiện (3), kiểm tra giá trị mã ký tự báo hiệu kết thúc dòng. Bởi vì chúng ta cần một giá trị để kiểm tra lần đầu tiên Chúng ta đi qua vòng lặp, chúng ta đọc giá trị đầu tiên trước khi vòng lặp bắt đầu (2)và sau đó đọc từng giá trị tiếp theo bên trong vòng lặp (5), sau mã xử lý. Một lần nữa, mã này sẽ xử lý một số độ dài chẵn bất kỳ. Để xử lý một số có độ dài lẻ bất kỳ, chúng ta chỉ cần sửa đổi mã xử lý, đảo ngược logic của điều kiện câu lệnh if (4) để nhân đôi các số ở các vị trí chẵn, thay vì các vị trí lẻ. Điều đó, ít nhất, làm cạn kiệt mọi khả năng. Độ dài của số nhận dạng phải là số lẻ hoặc số chẵn. Nếu chúng ta biết trước độ dài, chúng ta sẽ biết nên nhân đôi các vị trí lẻ hay các vị trí chẵn trong số. Tuy nhiên, chúng tôi không có thông tin đó cho đến khi chúng tôi đi đến cuối số. Liệu một giải pháp có bất khả thi với những ràng buộc này? Nếu chúng ta biết cách giải bài toán cho một số chữ số lẻ Và đối với một số chữ số chẵn nhưng không biết có bao nhiêu chữ số trong số cho đến khi chúng ta đọc nó hoàn toàn, làm thế nào chúng ta có thể giải quyết vấn đề này? Bạn có thể đã thấy câu trả lời cho vấn đề này. Nếu không, không phải vì câu trả lời khó mà vì nó được ẩn giấu trong các chi tiết. Những gì chúng ta có thể sử dụng ở đây là một sự tương tự, nhưng chúng ta chưa thấy một tình huống tương tự cho đến nay. Thay vào đó, chúng ta sẽ tạo ra sự tương tự của riêng mình. Chúng ta hãy đưa ra một vấn đề rõ ràng về chính tình huống này và xem liệu việc nhìn chằm chằm vào vấn đề có giúp chúng ta tìm ra giải pháp hay không. Xóa tâm trí của bạn về những định kiến dựa trên công việc cho đến nay, và đọc vấn đề sau đây.

5.PROBLEM: POSITIVE OR NEGATIVE

Viết một chương trình đọc 10 số nguyên từ người dùng. Sau khi tất cả các số đã được nhập, người dùng có thể yêu cầu hiển thị số lượng số dương hoặc số lượng số âm. Đây là một vấn đề đơn giản, một vấn đề dường như không có bất kỳ sự phức tạp nào cả. Chúng ta chỉ cần một biến đếm các số dương và một biến khác đếm các số âm. Khi người dùng chỉ định yêu cầu ở cuối chương trình, chúng ta chỉ cần tham khảo biến thích hợp cho phản hồi:

```
int number;
int positiveCount = 0;
int negativeCount = 0;
for (int i = 1; i <= 10; i++) {
   cin >> number;
   if (number > 0) positiveCount++;
   if (number < 0) negativeCount++;
}
char response;
cout << "Do you want the (p)ositive or (n)egative count? ";
cin >> response;
if (response == 'p')
   cout << "Positive count is " << positiveCount << "\n";
if (response == 'n')
    out << "Negative count is " << negativeCount << "\n";
```

Phân tích

Chương trình này thực hiện việc đếm số số dương và số âm trong một dãy 10 số.

- Dòng 1: Khai báo biến number kiểu int.
- Dòng 2: Khởi tạo biến số dương là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 3: Khởi tạo biến số dương là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 4: Thực hiện vòng lặp từ 1 đến 10.
- Dòng 5: Nhận đầu vào từ người dùng.
- Dòng 6-7: Trong mỗi lần lặp, Nếu number lớn hơn 0 số dương đếm tăng 1 đơn vị, nếu number nhỏ hơn 0 số âm đếm tăng 1 đơn vị.
- Dòng 8: Khai báo biến response kiểu char, biến nhận phản hồi của người dùng.
- Dòng 9: Hiển thị câu hỏi cho người dùng.
- Dòng 10: Nhận phản hồi của người dùng.
- Dòng 11-14: Nếu người dùng chọn số dương thì chương trình sẽ hiển thị số dương hoặc người dùng chọn số âm thì chương trình sẽ hiển thị số âm tương ứng với phản hồi của người dùng.

Điều này cho thấy phương pháp chúng ta cần sử dụng cho bài toán tổng kiểm tra Luhn: Theo dõi tổng kiểm tra đang chạy cả hai chiều, như thể số nhận dạng num-ber là một độ dài lẻ và một lần nữa như thể nó là độ dài chẵn. Khi chúng ta đến cuối số và khám phá ra độ dài thực sự, chúng ta sẽ có tổng kiểm tra chính xác trong biến này hay biến khác.

6.PUTTING THE PIECES TOGETHER

Bây giờ chúng tôi đã kiểm tra mọi thứ trong danh sách "việc cần làm" ban đầu của chúng tôi. Đã đến lúc kết hợp mọi thứ lại với nhau và giải quyết vấn đề này. Bởi vì chúng tôi đã giải quyết tất cả các vấn đề phụ một cách riêng biệt, chúng tôi biết chính xác những gì chúng tôi cần làm và có thể sử dụng các chương trình trước đó của chúng tôi làm tài liệu tham khảo để tạo ra kết quả cuối cùng một cách nhanh chóng:

```
char digit;
int oddLengthChecksum = 0;
int evenLengthChecksum = 0;
int position = 1;
cout << "Enter a number: ";
digit = cin.get();
while (digit != 10) {
    if (position % 2 == 0) {
       oddLengthChecksum += doubleDigitValue(digit - '0');
       evenLengthChecksum += digit - '0';
    } else {
       oddLengthChecksum += digit - '0';
       evenLengthChecksum += doubleDigitValue(digit - '0');
    }
    digit = cin.get();
    position++;
}
int checksum;
if ((position - 1) % 2 == 0) checksum = evenLengthChecksum;
else checksum = oddLengthChecksum;
cout << "Checksum is " << checksum << ". \n";
if (checksum % 10 == 0) {
   cout << "Checksum is divisible by 10. Valid. \n";
} else {
   cout << "Checksum is not divisible by 10. Invalid. \n";
}
```

Phân tích

- Dòng 1: Khai báo biến digit kiểu char.
- Dòng 2: Khởi tạo biến tổng các số lẻ là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 3: Khởi tạo biến tổng các số chẵn là một số nguyên và gán giá trị ban đầu là 0.
- Dòng 4: Khởi tạo biến position là một số nguyên và gán giá trị ban đầu là 1.
- Dòng 5-6: Hiển thị lời nhắc nhập 1 số.
- Dòng 7-14: Thực hiện vòng lặp nhập số. Trong mỗi lần lặp, chương trình sẽ xác định số chẵn hay lẻ và tính tổng số tương ứng.

- Dòng 15: Tăng vị trí hiện tại lên 1 đơn vị.
- Dòng 17: Khai báo biến checksum kiểu int.
- Dòng 18-20: Tính tổng số cuối cùng, thỏa điều kiện xuất ra tổng các số chẵn, ngược lại là tổng các số lẻ.
- Dòng 21-24: Kiểm tra xem tổng các chữ số có chia hết cho 10 hay không. Nếu tổng các chữ số chia hết cho 10, chương trình sẽ in ra thông báo "Tổng kiểm tra chia hết cho 10. Hợp lệ". Ngược lại, chương trình sẽ in ra thông báo " Tổng kiểm tra chia hết cho 10. Không hợp lệ.".

Lưu ý rằng khi chúng ta kiểm tra xem độ dài của số đầu vào là lẻ hay chẵn (, chúng ta trừ 1 khỏi vị trí. Chúng tôi làm điều này bởi vì char-acter cuối cùng mà chúng tôi đọc trong vòng lặp sẽ là kết thúc kết thúc, không phải là chữ số cuối cùng của số. Chúng ta cũng có thể viết biểu thức kiểm tra là (vị trí % 2 == 1), nhưng điều đó khó hiểu hơn khi đọc. Nói cách khác, tốt hơn là nói "nếu vị trí - 1 là chẵn, hãy sử dụng tổng kiểm tra chẵn" hơn là "nếu vị trí là lẻ, hãy sử dụng tổng kiểm tra chẵn" và phải nhớ tại sao điều đó có ý nghĩa. Đây là danh sách mã dài nhất mà chúng tôi đã xem xét cho đến nay, nhưng tôi không cần phải chú thích mọi thứ trong mã và mô tả cách mỗi phần hoạt động vì bạn đã thấy từng phần riêng biệt. Đây là sức mạnh của việc có một kế hoạch. Tuy nhiên, điều quan trọng cần lưu ý là kế hoạch của tôi không nhất thiết phải là kế hoạch của bạn. Các vấn đề tôi thấy trong mô tả ban đầu về vấn đề và các bước tôi đã thực hiện để giải quyết các vấn đề đó có thể khác với những gì bạn sẽ thấy và xong. Nền tảng của bạn là một lập trình viên và các vấn đề bạn đã hoàn thành thành công xác định phần nào của vấn đề là tầm thường hoặc khó khăn và do đó những bước bạn cần thực hiện để giải quyết vấn đề. Có thể đã có một điểm trong phần trước mà tôi đã đi đường vòng trông giống như một đường vòng không cần thiết để tìm ra điều gì đó đã rõ ràng đối với bạn. Ngược lại, có thể đã có một điểm mà tôi nhanh chóng bỏ qua một số điều khó khăn đối với bạn.

Ngoài ra, nếu bạn đã tự mình làm việc này, bạn có thể đã đưa ra một chương trình thành công không kém trông khá khác với chương trình của tôi. Không có một giải pháp "đúng" nào cho một vấn đề, vì bất kỳ chương trình nào đáp ứng tất cả các ràng buộc đều được tính là một giải pháp và đối với bất kỳ giải pháp nào, không có cách nào "đúng" để đạt được nó. Nhìn thấy tất cả các bước mà chúng tôi đã thực hiện để đạt được giải pháp, cùng với sự ngắn gọn tương đối của mã cuối cùng, bạn có thể bị cám dỗ để cố gắng cắt giảm các bước trong quy trình giải quyết vấn đề của riêng bạn. Tôi sẽ cảnh báo chống lại sự thúc đẩy này. Luôn luôn tốt hơn để thực hiện nhiều bước hơn là cố gắng làm quá nhiều cùng một lúc, ngay cả khi một số bước có vẻ tầm thường. Hãy nhớ những mục tiêu là gì trong việc giải quyết vấn đề. Tất nhiên, mục tiêu chính là tìm ra một chương trình giải quyết vấn đề đã nêu Câu đố thuần túy và đáp ứng tất cả các ràng buộc. Mục tiêu thứ hai là tìm chương trình đó trong khoảng thời gian tối thiểu. Giảm thiểu số bước không phải là mục tiêu và không ai phải biết bạn đã thực hiện bao nhiêu bước. Cân nhắc cố gắng lên đến đỉnh của một ngọn đồi dốc có con đường nông nhưng dài và quanh co. Bỏ qua con đường và leo lên đồi trực tiếp từ chân núi lên đỉnh chắc chắn sẽ đòi hỏi ít bước hơn so với đi theo con đường - nhưng nó có nhanh hơn không? Kết quả có khả năng nhất của việc leo thẳng là bạn bỏ cuộc và gục ngã. Cũng nên nhớ quy tắc cuối cùng của tôi để giải quyết vấn đề: Tránh thất vọng. Bạn càng cố gắng làm nhiều việc trong mỗi bước, bạn càng mời gọi sự thất vọng tiềm ẩn. Ngay cả khi bạn lùi lại một bước khó khăn và chia nó thành các bước phụ, thiệt hại sẽ được thực hiện bởi vì về mặt tâm lý, bạn sẽ cảm thấy như bạn đang đi lùi thay vì tiến bộ. Khi tôi huấn luyện

các lập trình viên mới bắt đầu theo cách tiếp cận từng bước, đôi khi tôi có một sinh viên phàn nàn, "Này, bước đó quá dễ dàng." Tôi trả lời: "Anh đang giải thích về điều gì vậy?" Nếu bạn đã thực hiện một vấn đề ban đầu trông có vẻ khó khăn và chia nó thành những mảnh nhỏ đến mức mỗi phần đều tầm thường để hoàn thành, tôi nói: Xin chúc mừng! Đó chỉ là những gì bạn nên hy vọng.

CHƯƠNG III.TÌM HIỂU VỀ TRACKING STATE:

Một trong những thách thức của câu đó lập trình Pure Puzzles là Tracking State, nghĩa là theo dõi trạng thái của chương trình và trong các ngôn ngữ lập trình thông thường ta biết đến. Chúng ta có thể sử dụng biến để lưu trữ trạng thái của chương trình.Tuy nhiên vì trong Pure Puzzles, chúng ta không thể sử dụng biến vì nó sẽ tạo ra side effect (tác dụng phụ).

1.SIDE EFFECT: là một thuật ngữ chỉ tác dụng phụ của một hàm hoặc biểu thức. Tác động phụ có thể là thay đổi giá trị của một biến, xuất dữ liệu ra màn hình, hoặc thực hiện một tác vụ nào đó trên hệ thống.

Các tác dụng phụ này khiến cho chương trình trở nên khó hiểu và khó bảo trì. Để tránh các tác dụng phụ, các lập trình viên cố gắng viết các hàm và biểu thức có tác dụng phụ tối thiểu.

Các side effect này bao gồm:

- Tạo ra một kết quả phụ, chẳng hạn như in ra một số tệp, mở một số tệp
- Thay đổi các giá trị của một biến bên ngoài hàm hoặc biểu thức
- Và gọi thêm một hàm khác có thể gây ra thêm side effect

a.Hàm Print (): Gây ra side effect bằng cách in ra một chuỗi văn bản hoặc một giá trị.Đây là side effect bên ngoài vì nó đã tác động đến môi trường bên ngoài hàm biểu thức

Ví dụ:

NGÔN NGỮ PYTHON:

Hàm Print () in chuỗi văn bản ra màn hình.

```
print("Hello world")
```

Kết quả:

```
Hello world
```

Hàm Print () in một giá trị ra màn hình

```
a=5
print(a)
```

Kết quả:

```
5
```

Lưu ý: Ngoài ra, **hàm Print ()** còn có thể gây ra side effect bằng cách gọi một hàm khác, có thể gây ra side effect.

Ví dụ: gọi **hàm Print ()** sau rồi sẽ gọi hàm **Open ()**, có thể gây ra side effect bằng cách mở một tệp:

```
print(open("file.txt", "r"))
```

Kết quả:

```
<_io.TextI0Wrapper name='file.txt' mode= 'r' encoding= 'utf-8'>>
```

Trong ví dụ này, **hàm Print ()** đã gọi hàm **open (),** có thể gây ra side effect bằng cách mở tệp "file.txt". Điều này là một side effect, vì nó đã tác động đến môi trường bên ngoài hàm hoặc biểu thức.

b. Hàm Open ( ):gây ra hiện tượng side effect bằng cách mở một tệp. Đây là một side effect bên ngoài, vì nó tác động đến môi trường bên ngoài hàm hoặc biểu thức.

Ví dụ:

Ví dụ, **hàm Open ()** sau sẽ mở tệp "file.txt" và trả về đối tượng tệp:

```
f=open("file.txt", "r")
```

Trong ví dụ, **hàm Open ()** đã mở tệp "file.txt". Điều này là một side effect, vì nó đã tác động đến môi trường bên ngoài hàm hoặc biểu thức.Ngoài ra hàm **Open ()** cũng có thể gây ra side effect bằng cách gọi một hàm khác.Ví dụ, hàm **Open ()** sau đó gọi hàm **Print ()**.

```
f=open("file.txt", "r")

print(f.read())
```

Kết quả:

```
This is a file.
```

Trong ví dụ này, hàm **Open ()** đã mở tệp "file.txt". Sau đó, hàm **Print ()** đã được gọi để in ra nội dung của tệp. Điều này là một side effect, vì nó đã tác động đến môi trường bên ngoài hàm hoặc biểu thức.

Nhìn chung thì hàm **Open ()** có thể gây ra side effect bằng cách mở một tệp, hoặc gọi một hàm khác cũng có thể gây ra side effect

Dưới đây là một số ví dụ side effect của hàm **Open ()**:

- Mở một tệp có thể khiến tệp đó bị khóa, ngăn người dùng khác truy cập vào tệp đó
- Mở một tệp có thể chiếm tài nguyên của hệ thống, chẳng hạn như là bộ nhớ CPU
- Mở một tệp có thể gây ra lỗi, nếu tệp không tồn tại hoặc bị hỏng

Do đó cần phải cẩn thận khi sử dụng hàm **Open ()**, để tránh xảy ra các side effect không mong muốn

c.Hàm Assign (): Gây ra side effect bằng cách thay đổi giá trị của một biến. Đây là một side effect bên trong, vì nó chỉ tác động đến trạng thái bên trong hàm hoặc biểu thức.

Ví dụ: **Hàm Assign** sẽ thay đổi giá trị của biến a thành 5:

```
a=5
```

Trong ví dụ này thì **hàm Assign** đã biến giá trị biến a thành 10. Điều này là một side effect vì nó đã tác động đến bên trong hàm hoặc biểu thức.

Tuy vậy,**hàm Assign** cũng có thể gây ra side effect bên ngoài, nếu biến được gán là một đối tượng có side effect.

Ví dụ:

Trong ví dụ này, **hàm Assign** đã gán đối tượng tệp được mở bởi hàm **Open ()** cho biến a. Điều này là một side effect bên ngoài, vì nó đã mở tệp "file.txt".

```
a=open("file.txt", "r")
```

Nhìn chung hàm Assign được có thể gây ra side effect bên trong hoặc ngoài tùy thuộc vào loại biểu thức được gán.

Dưới đây là một số ví dụ side effect của hàm Assign:

- Thay đổi giá trị của một biến có thể ảnh hưởng tới kết quả của các biểu thức khác.
- Thay đổi giá trị của một biến có thể khiến chương trình hoạt động khác với mong đợi.
- Thay đổi giá trị của một biến có thể gây ra lỗi, nếu biến đó ta sẽ dụng sai cách.

Do đó, cẩn thận khi sử dụng **hàm Assign**, để tránh gây ra các side không mong muốn.

Và để giải quyết tất cả các vấn đề này, ta có thể sử dụng các kỹ thuật sau:

- Sử dụng hàm đệ quy để lưu trạng thái của chương trình trong các tham số của hàm.
- Sử các cấu trúc dữ liệu không thay đổi để lưu trữ trạng thái của chương trình
- Sử dụng các hàm reduce () hoặc fold () để chuyển đổi một chuỗi đầu vào thành một giá trị duy nhất

Sử dụng hàm đệ quy

Một cách để lưu trữ trạng thái của chương trình trong Pure Puzzles là sử dụng các hàm đệ quy. Trong một hàm đệ quy, chúng ta có thể truyền trạng thái của chương trình dưới dạng tham số cho hàm đệ quy

Ví dụ, đây là một hàm đệ quy để tính giai thừa của một số nguyên:

```
def factorial(n):
if n==0:
return 1
else:
```

```
return n*factorial(n-1)
```

Hàm này sử dụng tham số *n* để lưu trữ trạng thái của chương trình. Khi hàm được gọi lần đầu tiên,*n* là số nguyên mà ta muốn tính giai thừa. Khi hàm được gọi đệ quy, *n* là số trừ đi 1

Ví dụ:

- ***Palindrome***: Một số palindrome là số bạn đọc theo chiều xuôi và chiều ngược thì đều như nhau (VD:121,12321,3443,…)

Dưới đây là một hàm đệ quy để kiểm tra xem một chuỗi có là palindrome hay không :

```
def is_palindrome(s):
if len(s) <=1:
return True
else:
return s[0]==s[-1] and is_palindrome(s[1:-1])
```

Hàm này sử dụng tham số *s* để lưu trữ trạng thái của chương trình. Khi hàm được gọi lần đầu tiên, *s* là chuỗi mà chúng ta muốn kiểm tra. Khi hàm được gọi đệ quy, *s* là chuỗi mà chúng ta đã loại bỏ hai ký tự đầu và cuối

Sử dụng các cấu trúc dữ liệu không thay đổi

Một cách khác để lưu trữ trạng thái của chương trình trong Pure Puzzles là sử dụng các cấu trúc dữ liệu không thay đổi. Các cấu trúc dữ liệu không thay đổi là các cấu trúc dữ liệu mà chúng ta không thể thay đổi giá trị sau khi chúng được tạo ra
Ví dụ:

Dưới đây là một cấu trúc dữ liệu không thay đổi để lưu trữ trạng thái của một trò chơi cờ vua:

```
class ChessBoard:
def _init_(self,board):
self.board=board
def _repr_(self):
return str(self.board)
```

Cấu trúc dữ liệu này lưu trữ trạng thái của trò chơi cờ vua dưới dạng một mảng 2 chiều. Mảng này không thể thay đổi sau khi cấu trúc dữ liệu được tạo ra.

CHƯƠNG 4.PROBLEMS TRACKING STATE

1.PROBLEM: DECODE A MESSAGE

-Là bài tập giải mã một thông điệp đã được mã hóa dưới dạng chuỗi các số nguyên, mỗi số được phân tách bằng dấu phẩy.Các số nguyên này đại diện cho các chữ cái trong bảng chữ cái tiếng Anh, số 1 thì là A, số 2 là B , số 3 là C,… và cứ tiếp tục như vậy cho đến hết bảng chữ cái.

Tuy nhiên thì các giải mã thông điệp không hề đơn giản như ta nghĩ. Có 3 chế độ giải mã. Chế độ ban đầu là chữ hoa,chế độ thứ 2 là chữ thường, chứ độ thứ 3 là dấu chấm câu. Mỗi khi số dư của phép chia các số nguyên cho 27 bằng 0 thì chế độ giải mã sẽ chuyển sang chế độ giải mã tiếp theo, trong chuỗi theo quy luật: CHỮ HOA → chữ thường → dấu chấm câu → CHỮ HOA. Nếu chế độ giải mã hiện tại là dấu chấm câu thì ta phải chia các số nguyên cho 9 thay vì 27.

Và để giải mã thông điệp, ta cần các bước như sau:

1. Khởi tạo chế độ giải mã thành chữ hoa.
2. Đọc số nguyên tiếp theo từ chuỗi.
3. Nếu số nguyên là 0 thì chuyển sang chế độ giải mã tiếp theo .
4. Nếu số nguyên là số khác 0, hãy giải mã số nguyên theo chế độ giải mã hiện tại.
5. Xuất ra ký tự được giải mã
6. Lặp lại các bước 2-5 cho đến khi hết chuỗi.

Dưới đây là bảng các dấu chấm câu được gán cho từng số nguyên trong chế độ dấu chấm câu:

Table 2-3: Punctuation Decoding Mode

Number	Symbol
1	!
2	?
3	,
4	.
5	(space)
6	;
7	"
8	'

Ví dụ:

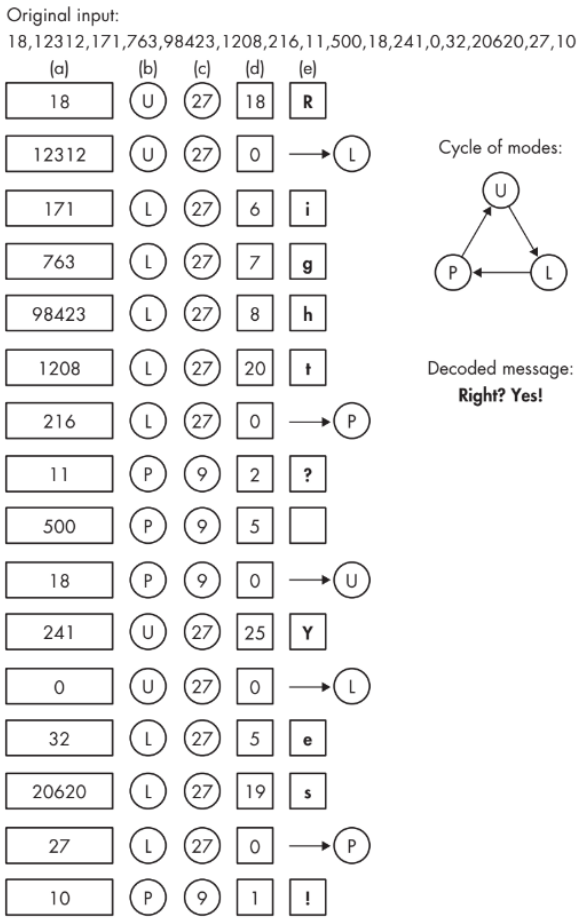


Figure 2-4: Sample processing for the "Decode a Message" problem

Các bước xử lý sẽ được tiến hành từ trên xuống. Cột (a) hiển thị số đầu vào hiện tại, cột (b) là chế độ hiện tại, chuyển từ chữ hoa (U) sang chữ thường (L) đến dấu chấm câu(P). Cột (c) hiển

thị số chia cho chế độ hiện tại.Cột (d) là số dư của phép chia hiện tại số chia ở cột (C) và số đầu vào ở cột (a). Kết quả được hiển thị trong cột (e). Nếu kết quả ở (d) là bằng 0 thì chuyển sang chế độ tiếp theo ở trong chu kỳ.

Dưới đây là bảng phân tích và quá trình giải m:

Chế độ giải mã: Chữ hoa

Số đầu vào: 18

Số dư: 18

Ký tự được giải mã: R

Chế độ giải mã: Chữ hoa

Số đầu vào: 12312

Số dư: 0

Chế độ giải mã chuyển sang chữ thường

Chế độ giải mã: Chữ thường

Số đầu vào: 171

Số dư: 6

Ký tự được giải mã: i

Chế độ giải mã: Chữ thường

Số đầu vào: 763

Số dư: 7

Ký tự được giải mã: g

Chế độ giải mã: Chữ thường

Số đầu vào: 98423

Số dư: 8

Ký được được giải mã: h

Chế độ giải mã: Chữ thường

Số đầu vào: 1208

Số dư: 20

Ký tự được giải mã: t

Chế độ giải mã: Chữ thường

Số đầu vào: 216

Số dư: 0

Chế độ giải mã chuyển sang dấu chấm câu

Chế độ giải mã: Dấu chấm câu

Số đầu vào: 11

Số dư: 2

Ký tự được giải mã: ?

Chế độ giải mã: Dấu chấm câu

Số đầu vào: 500

Số dư: 5

Ký tự được giải mã: (Space)

Chế độ giải mã: Dấu chấm câu

Số đầu vào: 18

Số dư: 0

Chế độ giải mã chuyển sang chữ hoa

Chế độ giải mã: Chữ hoa

Số đầu vào: 241

Số dư 25

Ký tự được giải mã: Y

Chế độ giải mã: Chữ hoa

Số đầu vào: 0

Số dư: 0

Chế độ giải mã chuyển sang chữ thường

Chế độ giải mã: Chữ thường

Số đầu vào: 32

Số dư: 5

Ký tự được giải mã: e

Chế độ giải mã: Chữ thường

Số đầu vào: 20620

Số dư: 19

Ký tự được giải mã: s

Chế độ giải mã: Chữ thường

Số đầu vào: 27

Số dư: 0

Chế độ giải mã chuyển sang dấu chấm câu

Chế độ giải mã: Dấu chấm câu

Số đầu vào: 10

Số dư: 1

Ký tự được giải mã: !

Tin nhắn được giải mã: Right? Yes!

a.Storing Code For Later Reuse

Tại sao việc tái sử dụng mã lại quan trọng?

Việc sử dụng lại mã rất quan trọng vì một số lý do.:

- Nó có thể giúp bạn tiết kiệm thời gian và công sức. Nếu bạn đã viết một đoạn mã thực hiện một tác vụ cụ thể thì bạn không cần phải viết lại từ đầu.
- Việc sử dụng lại mã có thể giúp cải thiện chất lượng mã của bạn. Nếu bạn đang sử dụng một đoạn mã đã được kiểm tra và sửa lỗi, bạn sẽ ít có khả năng gặp phải các lỗi mới hơn.
- Việc sử dụng lại mã có thể giúp mã của bạn dễ bảo trì hơn. Nếu bạn sử dụng mã mô-đun và được ghi chép đầy đủ thì sau này bạn sẽ dễ hiểu và cập nhật mã hơn.

Cách lưu trữ mã để sử dụng lại sau này?

Một cách tiếp cận phổ biến là sử dụng hệ thống kiểm soát phiên bản như Git hoặc SVN. Hệ thống kiểm soát phiên bản cho phép bạn theo dõi các thay đổi đối với mã của mình theo thời gian và tạo các nhánh và thẻ. Điều này giúp bạn dễ dàng hoàn nguyên về phiên bản mã trước đó nếu cần.

Một cách tiếp cận khác để lưu trữ mã để sử dụng lại sau này là sử dụng trình quản lý đoạn mã. Trình quản lý đoạn mã cho phép bạn lưu trữ và sắp xếp các đoạn mã để sử dụng lại sau này. Chúng cũng có thể được sử dụng để chèn các đoạn mã vào IDE hoặc trình soạn thảo của bạn.

Cuối cùng, bạn cũng có thể chỉ cần lưu trữ mã của mình trong các tệp văn bản thuần túy. Đây là một cách tiếp cận đơn giản và linh hoạt, nhưng có thể khó sắp xếp và theo dõi mã nếu bạn có nhiều mã.

Mẹo lưu trữ mã để sử dụng lại sau này?

Sử dụng hệ thống kiểm soát phiên bản hoặc trình quản lý đoạn mã. Điều này sẽ giúp việc tìm và sử dụng mã của bạn sau này dễ dàng hơn.

Sắp xếp mã của bạn một cách hợp lý. Điều này sẽ giúp bạn dễ dàng tìm thấy mã bạn cần khi cần.

Ghi lại mã của bạn. Điều này sẽ giúp bạn hiểu và sử dụng mã của mình sau này dễ dàng hơn.

Sử dụng nhận xét để giải thích mã của bạn làm gì và tại sao. Điều này cũng sẽ giúp bạn hiểu và sử dụng mã của mình sau này dễ dàng hơn.

Lưu các chương trình trung gian. Điều này có thể hữu ích nếu sau này bạn cần quay lại dự án và thấy rằng bạn cần sử dụng lại một số mã bạn đã viết cho các chương trình trung gian ...

Bạn có thể chia sẻ mã của mình với người khác, điều này có thể giúp họ viết phần mềm tốt hơn. Nhìn chung, đoạn văn lập luận rằng việc lưu trữ mã để sử dụng lại sau này là một thực tiễn quan

trọng đối với tất cả các nhà phát triển phần mềm. Bằng cách đó, bạn có thể tiết kiệm thời gian, cải thiện kỹ năng viết mã của mình và giúp người khác viết phần mềm tốt hơn.

Mở rộng phương pháp chuyển đổi các ký tự có một chữ số thành số nguyên để xử lý các số có nhiều chữ số?

Đọc các chữ số riêng lẻ dưới dạng ký tự. Điều này có thể được thực hiện bằng cách sử dụng vòng lặp hoặc hàm đệ quy. Khi các chữ số riêng lẻ đã được đọc, chúng cần được chuyển đổi thành số nguyên.

```
int overallNumber = digit1 * 10 + digit2;
```

Trong này digit1 là hàng chục và digit2 là hàng đơn vị. Ta lấy ví dụ số 35 sẽ bằng = 3*10+5.

Tương tự các số có 3 chữ số = a*100+b*10+c.

Chương trình đầy đủ:

```
cout << "Enter a two-digit number: ";
char digitChar1 = cin.get();
char digitChar2 = cin.get();
int digit1 = digitChar1 - '0';
int digit2 = digitChar2 - '0';
int overallNumber = digit1 * 10 + digit2;
cout << "That number as an integer: " << overallNumber << "\n";
```

Điều đó có nghĩa chúng sẽ xuất ra cùng một số có 2 chữ số mà ta đã nhập vào. Chương trình sử dụng hai biến khác nhau để giữ 2 ký tự đầu vào.Nếu ta mở rộng lên các số có 3 chữ số và 4 chữ số ta sẽ cần thêm nhiều biến. Sẽ trở nên lộn xộn và khó sửa chữa.Giải pháp chung đó là giảm mã trước xuống chỉ còn 2 biến (1 int , 1 char).

```
cout << "Enter a two-digit number: ";
❶ char digitChar = cin.get();
❷ int overallNumber = (digitChar - '0') * 10;


❸ digitChar = cin.get();
❹ overallNumber += (digitChar - '0');
cout << "That number as an integer: " << overallNumber << "\n";
```

Vậy nếu ta mở rộng lên số có 3 chữ số hoặc 4 chữ số thì sao. Vấn đề là mã trước sử dụng một hệ số nhân cụ thể cho từng chữ số, dựa trên vị trí của chữ số trong số. Ví dụ nhân chữ số ngoài cùng bên trái với 10 vì chữ số đó nằm ở vị trí hàng chục.

Ta thấy được cách tiếp cận này không hiệu quả đối với các số có 3 chữ số, vì chúng ta sẽ không biết số đó có bao nhiêu chữ số cho đến khi đạt tới vị trí cuối cùng của số đó. Chúng ta cũng không thể mặc định chữ số ngoài cùng bên trái là ở vị trị hàng trăm hoặc ở bất kỳ vị trí nào khác trước khi ta biết số đó có bao nhiêu chữ số.

2. PROBLEMS: READING A NUMBER WITH THREE OR FOUR DIGITS

- **Viết một chương trình đọc từng chữ số và chuyển nó thành số,chỉ sử dụng một biến char và int. Số đó sẽ có một trong hai số có 3 chữ số hoặc 4 chữ số.**

Dưới đây là chương trình :

```
#include<iostream>
using namespace std;
int chuyendoiso(string so){
int songuyen=0;
char chuso;
for(int i=0;i<so.length();i++){
chuso=so[i];
songuyen*=10;
songuyen +=chuso-'0';
}
return songuyen;
}
int main()
{
string so;
cout<< "Nhập số có 3 chữ số hoặc số có 4 chữ số :";
cin>>so;
int songuyen=chuyendoiso(so);
cout<< " Số dưới dạng đầy đủ là :"<< songuyen<<endl;
return 0;
}
```

Kết quả xuất ra :

```
Nhập số có 3 chữ số hoặc số có 4 chữ số :123
Số dưới dạng đầy đủ là : 123
```

Phân tích:

- Hàm **chuyendoiso()**,hàm này có một tham số là một chuỗi ký tự. Hàm sẽ trả về một giá trị số nguyên là giá trị của số được chuyển đổi.
  Hàm này hoạt động như sau:
  - Biến **songuyen** được khởi tạo bằng giá trị 0. Đây là giá trị cuối cùng của số được chuyển đổi.
  - Biến **chuso** được sử dụng để lưu trữ giá trị của từng ký tự trong chuỗi **so**.
  - **Vòng lặp for** duyệt qua từng ký tự của chuỗi **so**.
  - **Trong mỗi lần lặp, hàm sẽ thực hiện các bước sau:**
    - Gán giá trị của ký tự hiện tại cho biến **chuso**.
    - Nhân giá trị hiện tại của biến **songuyen** với 10.
    - Thêm giá trị của ký tự **chuso** trừ đi ký tự '0' vào biến **songuyen**.

    Ví dụ: nếu chuỗi **so** có giá trị là "123", thì hàm **chuyendoiso()** sẽ thực hiện các bước sau:

    ```
    chuso= '1';

    songuyen *=10;

    songuyen += 2- '0'; // songuyen = 1

    chuso = '2';

    songuyen *=10;

    songuyen += 2 - '0'; //songuyen = 12

    chuso = '3';

    songuyen *=10;

    songuyen += 3 - '0'; // songuyen = 123
    ```

    Như vậy, giá trị trả về hàm **chuyendoiso()** sẽ là 123.\

- Hàm **main()** , thực hiện các bước :
  - Khai báo biến **so** là kiểu chuỗi.
  - Hiển thị thông báo yêu cầu người dùng nhập một số có 3 hoặc 4 chữ số.
  - Đọc dữ liệu từ bàn phím và lưu trữ vào biến **so**.
  - Gọi hàm **chuyendoiso()** để chuyển đổi số từ dạng chuỗi sang dạng số nguyên.
  - Hiển thị giá trị của biến **songuyen** dưới dạng số nguyên.
  - Trả về giá trị 0.

  Ví dụ: Nếu người dùng nhập số "123" vào biến **so** , thì đoạn code này sẽ có kết quả như sau:

  ```
  Nhập số có 3 chữ số hoặc số có 4 chữ số :

  123

  Số dưới dạng đầy đủ là : 123
  ```

3.PROBLEMS: READING A NUMBER WITH THREE OR FOUR DIGITS, FURTHER SIMPLIFIED

Bài toán này là dạng đơn giản của bài toán trước. Sự khác biệt chính giữa hai vấn đề là phiên bản đơn giản hóa không cho phép người nhập một số có ít hơn 3 chữ số. Điều này giúp giải quyết vấn đề dễ dàng hơn

```
cout << "Enter a three-digit or four-digit number: ";
char digitChar = cin.get();
❶ int threeDigitNumber = (digitChar - '0') * 100;
❷ int fourDigitNumber = (digitChar - '0') * 1000;
digitChar = cin.get();
threeDigitNumber += (digitChar - '0') * 10;
fourDigitNumber += (digitChar - '0') * 100;
digitChar = cin.get();
threeDigitNumber += (digitChar - '0');
fourDigitNumber += (digitChar - '0') * 10;
digitChar = cin.get();
if ❸(digitChar == 10) {
    cout << "Numbered entered: " << threeDigitNumber << "\n";
} else {
   ❹fourDigitNumber += (digitChar - '0');
    cout << "Numbered entered: " << fourDigitNumber << "\n";
}
```

Phân tích

Mã này đầu tiên khai báo ba biến:

- **digitChar**: Biến char để lưu trữ ký tự đầu vào hiện tại.
- **threeDigitNumber**: Biến int để lưu trữ số có ba chữ số.
- **fourDigitNumber**: Biến int để lưu trữ số có bốn chữ số.
- Sau đó, mã sẽ nhắc người dùng nhập một số có ba hoặc bốn chữ số và lưu chữ số đầu tiên vào biến **digitChar**.
- Sau đó, mã sẽ nhân giá trị **digitChar** với 100 và gán nó cho biến **threeDigitNumber**. Nó cũng nhân giá trị **digitChar** với 1000 và gán nó cho biến **fourDigitNumber**. Điều này chuyển đổi một cách hiệu quả chữ số đầu tiên thành số nguyên và lưu trữ nó trong cả hai biến **threeDigitNumber và fourDigitNumber**.
- Sau đó, mã sẽ đọc chữ số thứ hai từ luồng đầu vào và lưu nó vào biến **digitChar**. Sau đó, nó nhân giá trị **digitChar** với 10 và thêm nó vào biến **threeDigitNumber**. Nó cũng nhân giá trị **digitChar** với 100 và thêm nó vào biến **fourDigitNumber**. Điều này chuyển đổi một cách hiệu quả chữ số thứ hai thành số nguyên và thêm nó vào cả hai biến **threeDigitNumber** và **fourDigitNumber**.
- Mã lặp lại quá trình này cho chữ số thứ ba.
- Sau khi đọc chữ số thứ ba, mã sẽ kiểm tra xem có chữ số thứ tư hay không bằng cách đọc ký tự tiếp theo từ luồng đầu vào. Nếu không có chữ số thứ tư, mã sẽ in giá trị của

biến **threeDigitNumber** ra bàn điều khiển. Mặt khác, nó đọc chữ số thứ tư, chuyển nó thành số nguyên và thêm nó vào biến **fourDigitNumber**. Cuối cùng, nó in giá trị của biến **fourDigitNumber** ra màn hình console.

```
cout << "Enter a three-digit or four-digit number: ";
char digitChar = cin.get();
int number = (digitChar - '0') * 100;
digitChar = cin.get();
number += (digitChar - '0') * 10;
digitChar = cin.get();
number += (digitChar - '0');
digitChar = cin.get();
if (digitChar == 10) {
   cout << "Numbered entered: " << number << "\n";
} else {
   number = number * 10 + (digitChar - '0');
   cout << "Numbered entered: " << number << "\n";
}
```

Phân tích

- Dòng đầu tiên in lời yêu cầu nhập số có ba hoặc bốn chữ số.
- Dòng mã thứ hai đọc một ký tự đầu vào lưu nó vào biến **digitChar**.
- Dòng mã thứ ba chuyển đổi ký tự **digitChar** thành số nguyên và nhân nó với 100. Lưu kết quả trong biến số.
- Dòng mã thứ tư một ký tự đầu vào khác và lưu vào biến **digitChar**
- Dòng mã thứ năm chuyển đổi thành số nguyên và nhân nó với 10. Lưu kết quả vào trong biến số.
- Dòng mã thứ sáu tiếp tục đọc một ký tự đầu vào khác vào lưu vào biến **digitChar**.
- Dòng mã thứ bảy chuyển đổi ký tự khác từ luồng đầu vào và thêm nó vào biến số.
- Dòng mã thứ tám đọc một ký tự đầu vào khác và lưu nó vào trong biến **digitChar**.
- Câu lệnh **if** lấy chữ số thứ tư của số từ người dùng (nếu có) và lưu nó vào biến **digitChar**. Kiểm tra xem chữ số thứ tư có phải ký tự mới hay không:
  + Nếu đúng thì số đó có ba chữ số. Sau đó sẽ in **"Numbered entered:"** tiếp theo sau là giá trị của biến số.
  + Nếu chữ số thứ tư không phải là ký tự dòng mới thì số đó dài bốn chữ số và biến số cần nhân với 10 để nhường chỗ cho chữ số thứ tư. Chữ số thứ tư sau đó được chuyển đổi thành số nguyên và được thêm vào biến số.Cuối cùng, sẽ in **" Numbered entered:"** theo sau là giá trị của biến số.

Mã này nhắc người dùng nhập số có ba hoặc 4 chữ số, đọc các chữ số từ luồng đầu vào, chuyển đổi chúng thành số nguyên và lưu chúng vào một biến.Cuối cùng, in giá trị của số đó ra màn hình.

Bây giờ hãy thử mở rộng mã này sang xử lý số năm chữ số.Sau khi xử lý tính toán các giá trị phù hợp cho bốn giá trị đầu tiên, chúng ta sẽ lặp lại quá trình tương tự như khi đọc số thứ tư.

```
cout << "Enter a number with three, four, or five digits: ";
char digitChar = cin.get();
int number = (digitChar - '0') * 100;
digitChar = cin.get();
number += (digitChar - '0') * 10;
digitChar = cin.get();
number += (digitChar - '0');
digitChar = cin.get();
if (digitChar == 10) {
   cout << "Numbered entered: " << number << "\n";
} else {
   number = number * 10 + (digitChar - '0');
   digitChar = cin.get();
   if (digitChar == 10) {
      cout << "Numbered entered: " << number << "\n";
   } else {
      number = number * 10 + (digitChar - '0');
      cout << "Numbered entered: " << number << "\n";
   }
}
```

Phân tích:

- Câu **digitChar=cin.get();** lệnh đọc số đầu tiên,thứ hai,thứ ba,… do người dùng nhập vào và lưu vào **digitChar**
- Câu lệnh **int number (digitChar- ‘ 0’)*100;** lệnh chuyển đổi ký tự thành giá trị số nguyên và nhân nó với 100. Điều này chuyển chữ số sang vị trí hàng trăm
- Câu lệnh **number +=(digitChar-‘0’)*10;** lệnh chuyển đổi ký tự thành giá trị số nguyên, công vào biến **number** và nhân kết quả với 10. Điều này chuyển chữ số đầu tiên sang vị trí hàng chục và cộng chữ số thứ hai đến vị trí hàng đơn vị
- Câu lệnh **number+=(digitChar-‘0’);** lệnh chuyển đổi ký tự thành giá trị số nguyên và thêm nó vào **number**. Điều này sẽ thêm chữ số thứ ba vào vị trí hàng đơn vị
- Câu lệnh **if(digitChar==10)** lệnh kiểm tra xem ký tự tiếp theo trong luồng đầu vào có phải là ký tự dòng mới hay không (mã ASCII 10). Nếu đúng vậy thì mã đã đọc một số có ba chữ số và biến **number** chứa giá trị chính xác. Câu **cout<<” Numbered enterd: ”<<number<<”\n”;** lệnh in số.
- Nếu ký tự tiếp theo sau chữ số thứ ba không phải là ký tự dòng mới thì mã đã đọc số có bốn hoặc năm chữ số. Câu **number=number*10+(digitChar-‘0’);** lệnh dịch các chữ số hiện có sang trái một vị và thêm chữ số thứ tư vào vị trí của các chữ số đó
- Câu **if(digitChar==10)** lệnh kiểm tra xem ký tự tiếp theo có phải là ký tự dòng mới hay không. Nếu đúng vậy thì mã đã đọc một số có bốn chữ số và biến **number** chứa giá trị chính xác. Câu **cout<<” Numbered entered: ”<<number<<”\n”;** lệnh dịch các chữ số hiện có sang trái vị trí và thêm chữ số thứ năm vào vị trí của các chữ số đó.

Tại thời điểm này chúng ta có thể dễ dàng mở rộng mã để có thể xử lý các số có sáu chữ số hoặc số có ít chữ số hơn.Nếu ký tự tiếp theo là một chữ số khác, nhân tổng số đang chạy với 10 trước khi thêm giá trị chữ số nguyên của ký tự. Với cách trên ta có thể viết một vòng lặp xử lý một số có độ dài bất kỳ.

```
   cout << "Enter a number with as many digits as you like: ";
❶  char digitChar = cin.get();
❷  int number = (digitChar - '0');
❸  digitChar = cin.get();
   while ❹(digitChar != 10) {
      ❺number = number * 10 + (digitChar - '0');
      ❻digitChar = cin.get();
   }
   ❼cout << "Numbered entered: " << number << "\n";
```

Phân tích:

- Câu lệnh **cout << "Enter a number with as many digits as you like:"** dòng mã này in một thông báo tới người dùng nhắc họ nhập số.
- Câu lệnh **char digitChar = cin.get();** dòng mã này đọc ký tự đầu tiên từ dữ liệu đầu vào của người dùng và lưu nó vào biến **digiChar**.
- Câu lệnh **int number = (digitChar - '0');** dòng mã này chuyển đổi ký tự **digiChar** thành giá trị nguyên của nó và lưu nó vào biến số. Ký tự '0' bị trừ khỏi chữ **digitChar** vì mã ASCII cho '0' là 48 và chúng tôi muốn giá trị nguyên của chữ số là 0.
- Câu lệnh **digitChar = cin.get();** dòng mã này đọc ký tự tiếp theo từ đầu vào của người dùng và lưu nó vào biến **digitChar**.
- Câu lệnh **while (digitChar!= 10){** dòng mã này bắt đầu một vòng lặp **while** và sẽ tiếp tục lặp cho đến khi ký tự **digitChar** không bằng 10. Mã ASCII cho ký tự dòng mới là 10, vì vậy vòng lặp này sẽ tiếp tục vòng lặp miễn là người dùng chưa nhập ký tự dòng mới.
- Câu lệnh **number = number * 10 + (digitChar - '0');** dòng mã này nhân giá trị hiện tại của **number** với 10 rồi cộng giá trị nguyên của ký tự **digitChar**. Điều này sẽ thêm chữ số **digitChar** vào cuối số một cách hiệu quả.
- Câu lệnh **digitChar = cin.get();** Dòng mã này đọc ký tự tiếp theo từ đầu vào của người dùng và lưu nó vào biến **digitChar**.
- Câu lệnh **cout << "Number entered: " << number << "\n";** dòng mã này in một thông báo cho người dùng biết họ đã nhập số nào. Biến số chứa giá trị nguyên của số mà người dùng đã nhập.

Cùng thảo luận về quá trình đọc một chuỗi số, được phân tách bằng dấu phẩy, từ dữ liệu nhập của người dùng. Nó gợi ý sửa đổi mã hiện có để xử lý tình huống này bằng cách kiểm tra cả dấu phẩy và ký tự cuối dòng. Vòng lặp bên trong sẽ dừng khi gặp một trong hai ký tự này, trong khi vòng lặp bên ngoài sẽ tiếp tục cho đến khi toàn bộ dữ liệu đầu vào được xử lý.

Các điểm chính :

- **Định dạng đầu vào**: Đầu vào bao gồm một dãy số được phân tách bằng dấu phẩy.
- **Chiến lược xử lý**: Mỗi số cần được đọc riêng lẻ và chuyển đổi thành số nguyên tương đương.
- **Cấu trúc vòng lặp**: Vòng lặp bên trong sẽ xử lý việc đọc và xử lý các số riêng lẻ, trong khi vòng lặp bên ngoài sẽ kiểm soát quá trình đầu vào tổng thể.
- **Điều kiện kết thúc**: Vòng lặp bên trong sẽ dừng khi gặp dấu phẩy hoặc ký tự cuối dòng. Vòng lặp bên ngoài chỉ dừng khi toàn bộ đầu vào đã được xử lý.
- **Sửa đổi mã**: Mã hiện tại cần được sửa đổi để phù hợp với yêu cầu xử lý và định dạng đầu vào mới.

```
❶ char digitChar;
do {
    digitChar = cin.get();
    int number = (digitChar - '0');
    digitChar = cin.get();
    while ((digitChar != 10) && (digitChar != ',')) {
        number = number * 10 + (digitChar - '0');
        digitChar = cin.get();
    }
    cout << "Numbered entered: " << number << "\n";
} while ❷(digitChar != 10);
```

Phân tích:

- Câu lệnh **char digitChar;** dòng mã này khai báo một biến ký tự có tên là digitChar.
- Câu lệnh **do {** dòng mã này bắt đầu một vòng lặp **do-while**. Vòng lặp sẽ tiếp tục thực hiện cho đến khi điều kiện ở cuối vòng lặp không đúng.
- Câu lệnh **digitChar = cin.get();** dòng mã này đọc một ký tự từ dữ liệu đầu vào của người dùng và lưu nó vào biến digitChar.
- Câu lệnh **int number = (digitChar - '0');** dòng mã này chuyển đổi ký tự **digitChar** thành giá trị nguyên của nó và lưu nó vào biến số. Ký tự **'0'** bị trừ khỏi **ditgitChar** vì mã ASCII cho **'0'** là 48 và chúng tôi muốn giá trị nguyên của chữ số là 0.
- Câu lệnh **digitChar = cin.get();** dòng mã này đọc một ký tự khác từ dữ liệu đầu vào của người dùng và lưu nó vào biến **digitChar**
- Câu lệnh **while ((digitChar != 10) && (digitChar != ',')) {** dòng mã này bắt đầu một vòng lặp while. Vòng lặp sẽ tiếp tục thực hiện miễn là điều kiện ở cuối vòng lặp là đúng. Điều kiện kiểm tra rằng **digitChar** không bằng ký tự dòng mới (10) hoặc ký tự dấu phẩy (44).
- Câu lệnh **number = number * 10 + (digitChar - '0');** dòng mã này nhân giá trị hiện tại của **number** với 10 rồi cộng giá trị nguyên của ký tự **digitChar**. Điều này sẽ thêm chữ số **digitChar** vào cuối số một cách hiệu quả.
- Câu lệnh **digitChar = cin.get();** dòng mã này đọc một ký tự khác từ dữ liệu đầu vào của người dùng và lưu nó vào biến **digitChar.**

- Câu lệnh **cout << "Numbered entered: " << number << "\n";** dòng mã này in một thông báo cho người dùng biết họ đã nhập số nào. Biến số chứa giá trị nguyên của số mà người dùng đã nhập.
- Câu lệnh } **while (digitChar != 10);** dòng mã này đóng vòng lặp do-while. Vòng lặp sẽ tiếp tục thực thi cho đến khi ký tự **digitChar** không bằng ký tự dòng mới (10).

Đây là ví dụ tuyệt vời về tầm quan trọng của các bước nhỏ mặc dù đây là một chương trình ngắn, nhưng bản chất bánh xe trong bánh xe của vòng lặp kép sẽ tạo ra mã phức tạp nếu chúng ta cố gắng viết mã này từ đầu. Tuy nhiên, điều đó thật đơn giản khi chúng ta đến mã này bằng cách thực hiện một bước so với chương trình trước đó. Khai báo digitChar (1) chuyển sang một dòng riêng biệt để khai báo nằm trong phạm vi xuyên suốt của mã.

Những điểm rút ra ở đây là :

- **Phát triển tăng dần**: Tiếp cận các vấn đề phức tạp thông qua các bước tăng dần làm cho quá trình trở nên ít khó khăn hơn và dễ quản lý hơn.
- **Tái sử dụng mã**: Việc xác định và sử dụng lại các đoạn mã hiện có có thể đơn giản hóa việc lập trình và giảm lỗi.
- **Vòng lặp để xử lý đầu vào**: Vòng lặp thường được sử dụng để xử lý dữ liệu đầu vào, cho phép thực hiện lặp lại dựa trên các điều kiện cụ thể.

Trong ngữ cảnh này, đoạn mã trình bày cách đọc một chuỗi số được phân tách bằng dấu phẩy. Vòng lặp bên trong xử lý việc đọc và chuyển đổi các số riêng lẻ, trong khi vòng lặp do-while bên ngoài kiểm soát quá trình nhập liệu tổng thể, kết thúc khi gặp ký tự cuối dòng.

Cách tiếp cận này chia nhiệm vụ thành các bước nhỏ hơn, dễ quản lý hơn một cách hiệu quả, giúp mã dễ hiểu hơn và dễ bảo trì hơn.

Với giải pháp đó, chúng ta có thể tập trung vào việc xử lý số riêng lẻ. Mục tiếp theo trong danh sách của chúng tôi là chuyển đổi số 1-26 thành chữ cái A-Z. Nếu nghĩ về nó đây thực sự là một đảo ngược của quá trình chúng ta sử dụng để chuyển đổi các ký tự chữ số riêng lẻ thành số nguyên tương đương của chúng. Nếu chúng ta trừ mã ký tự cho 0 để dịch từ phạm vi mã các ký tự 0-9 sang phạm vi số nguyên 0-9, chúng ta có thể thêm mã ký tự để dịch từ 1-26 sang A-Z.

Việc thêm mã kts tự cho ‘A’ thực sự là cách tiếp cận đúng để chuyển đổi số 1-26 thành chữ cái A-Z. Kỹ thuật này dịch chuyển một cách hiệu quả phạm vi giá trị số để tương ứng với mã ASCII cho chữ in hoa từ A-Z

Giải thích:

Mã ASCII cho ‘A’ là 65

Khi chúng ta thêm ‘A’ vào một số từ 1 đến 26, chúng ta sẽ nhận được mã ASCII cho chữ in hoa tương ứng.

Ví dụ: thêm ‘A’ vào 1 ta sẽ có 66 đây là mã ASCII cho ‘B’.

Tương tự việc thêm ‘A’ vào 2 sẽ cho ra 67, đây là mã ASCII của ‘C’,v.v.

Về bản chất, việc thêm ‘A’ vào một số 1-26 cho phép chúng ta chuyển đổi số đó thành chữ hoa tương ứng từ A-Z.

```
cout << "Enter a number 1-26: ";
int number;
cin >> number;
char outputCharacter;
outputCharacter = ❶number + 'A';
cout << "Equivalent symbol: " << outputCharacter << "\n";

Enter a number 1-26: 5
Equivalent letter: F
```

Phân tích:

- Câu lệnh **cout << "Enter a number 1-26: ";** dòng mã này in thông báo tới người dùng nhắc họ nhập số.
- Câu lệnh **int number;** Dòng mã này khai báo một biến số nguyên có tên là **number.**
- Câu lệnh **cin >> number;** dòng mã này đọc một số nguyên từ đầu vào của người dùng và lưu nó vào biến số.
- Câu lệnh **char outputCharacter;** dòng mã này khai báo một biến ký tự có tên là **outputCharacter**.
- Câu lệnh **outputCharacter = number + 'A';** dòng mã này chuyển đổi số nguyên thành chữ cái viết hoa tương ứng và lưu ký tự đó vào biến **outputCharacter**. Ký tự **'A'** được thêm vào số vì mã ASCII cho **'A'** là 65 và chúng tôi muốn mã ASCII của chữ cái nằm trong khoảng từ 65 đến 90.
- Câu lệnh **cout << "Equivalent symbol: " << outputCharacter << "\n";** dòng mã này in một thông báo cho người dùng cho biết chữ cái nào tương ứng với số họ đã nhập. Biến **outputCharacter** chứa chữ in hoa tương ứng với số.

Có gì đó không đúng. Chữ cái thứ năm trong bảng chữ cái là E chứ không phải F?

Chuyển đổi sang chữ hoa:

Mã ban đầu sử dụng công thức số + ‘A’ để chuyển đổi một số thành chữ in hoa tương ứng. Công thức này không chính xác vì Công thức này không chính xác vì nó giả định rằng phạm vi số bắt đầu từ 0, tương tự như việc chuyển đổi các chữ số ký tự thành số nguyên tương đương của chúng. Tuy nhiên, phạm vi số cho chữ in hoa bắt đầu từ 1, tương ứng với mã ASCII 65-90 cho 'A' đến 'Z'.

Công thức đã sửa là số + 'A' - 1. Công thức này trừ 1 từ công thức ban đầu để tính phần bù trong phạm vi số. Điều này đảm bảo rằng mã ASCII chính xác cho chữ in hoa được tạo ra.

Chuyển đổi sang chữ thường:

Việc chuyển đổi sang chữ thường cũng có cách tiếp cận tương tự. Công thức chuyển đổi một số thành chữ cái viết thường tương ứng là số + 'a' - 1. Công thức này sử dụng ký tự 'a' viết thường thay vì 'A' để tính đến sự khác biệt trong mã ASCII cho chữ hoa và chữ thường.

Chuyển đổi sang ký hiệu dấu chấm câu:

Việc chuyển đổi sang ký hiệu dấu chấm câu ít đơn giản hơn vì các ký hiệu dấu chấm câu không có thứ tự nhất quán trong ASCII hoặc các hệ thống mã ký tự khác. Có thể cần một cách tiếp cận phức tạp hơn liên quan đến bảng tra cứu hoặc logic tùy chỉnh.

```
cout << "Enter a number 1-8: ";
int number;
cin >> number;
char outputCharacter;
❶ switch (number) {
    case 1: outputCharacter = '!'; break;
    case 2: outputCharacter = '?'; break;
    case 3: outputCharacter = ','; break;
    case 4: outputCharacter = '.'; break;
    case 5: outputCharacter = ' '; break;
    case 6: outputCharacter = ';'; break;
    case 7: outputCharacter = '"'; break;
    case 8: outputCharacter = ❷'\''; break;
}
cout << "Equivalent symbol: " << outputCharacter << "\n";
```

Phân tích:

- Câu lệnh **cout << "Enter a number 1-8: ";** dòng mã này in thông báo tới người dùng nhắc họ nhập số.
- Câu lệnh **int number;** dòng mã này khai báo một biến số nguyên có tên là **number**.
- Câu lệnh **cin >> number;** dòng mã này đọc một số nguyên từ đầu vào của người dùng và lưu nó vào biến số.
- Câu lệnh **char outCharacter;** Dòng mã này khai báo một biến ký tự có tên là **outputCharacter**.
- Câu lệnh **switch (number) {** dòng mã này bắt đầu câu lệnh **switch**. Câu lệnh **switch** sẽ đánh giá giá trị của biến **number** và thực hiện trường hợp khớp với giá trị đó.
- Câu lệnh **case 1: outputCharacter = '!'; break;** dòng mã này dành cho trường hợp giá trị của số là 1. Nó gán ký tự ! đến biến **ouputCharacter**. Câu lệnh **break** được sử dụng để thoát khỏi câu lệnh **switch**.Tương tự cho các case 1,2,3,4,5,6,7,8.
- Câu lệnh **cout << "Equivalent symbol: " << outputCharacter << "\n";** dòng mã này in một thông báo cho người dùng cho biết ký hiệu nào tương ứng với số họ đã nhập. Biến **ouputCharacter** chứa ký hiệu tương ứng với số.

Lưu ý: Dấu gạch chéo ngược **case 8: outputCharacter = '\''; break;** dùng để thoát.

Sự cần thiết của một cơ chế để chuyển đổi giữa các chế độ chữ hoa, chữ thường và dấu câu trong mã. Ta nên sử dụng một bảng liệt kê để biểu diễn chế độ hiện tại thay vì sử dụng các giá trị nguyên tùy ý

Lý do sử dụng bảng liệt kê:

- **Tính dễ đọc**: Việc liệt kê làm cho mã dễ đọc hơn và dễ hiểu hơn vì tên của các giá trị liệt kê chỉ rõ ý nghĩa của chúng.
- **Khả năng bảo trì**: Việc sử dụng bảng liệt kê giúp dễ hiểu và sửa đổi mã hơn trong tương lai vì ý nghĩa của các giá trị chế độ được xác định rõ ràng.
- **Ngăn ngừa lỗi**: Việc liệt kê có thể giúp ngăn ngừa lỗi bằng cách hạn chế các giá trị mà biến chế độ có thể nhận, đảm bảo rằng chỉ sử dụng các trạng thái hợp lệ.

Trong C++, kiểu liệt kê enum là một kiểu dữ liệu được sử dụng để định nghĩa một tập hợp các giá trị có tên. Các giá trị này có thể được sử dụng để đại diện cho các trạng thái, mã lỗi hoặc bất kỳ giá trị nào khác có thể có số lượng hữu hạn các giá trị có thể.

Cú pháp của kiểu liệt kê enum:

enum<tên liệt kê>{<giá trị 1>,<giá trị 2>,…};

Trong đó:

<tên liệt kê> là tên của kiểu liệt kê.

<giá trị 1>,<giá trị 2>,… là các giá trị của kiểu liệt kê.

Ví dụ về kiểu liệt kê enum:

```
enum Color
{
RED,
GREEN,
BLUE
};
```

Kiểu liệt kê Color định nghĩa ba giá trị có tên RED, GREEN và BUE. Các giá trị này có thể được sử dụng để đại diện cho các màu sắc trong một ứng dụng

Ví dụ về cách sử dụng kiểu liệt kê enum trong C++

```
enum Color{
RED,
GREEN,
BLUE
};
```

```
int main(){
Color myColor = RED;
if(myColor==RED){
std::cout<< " Màu sắc là đỏ" << std::endl;
} else if(myColor==GREEN){
std::cout<< " Màu sắc là xanh lá" <<std::endl;
}else{
std::cout<< "Màu sắc là xanh dương"<<std::endl;
}
return 0;
}
```

Chương trình này sẽ in ra thông tin như sau :

```
Màu sắc là đỏ
```

Ưu điểm của kiểu liệt kê enum :

- Kiểu liệt kê enum có một số ưu điểm so với việc sử dụng các giá trị nguyên tố để đại diện cho các trạng thái hoặc mã lỗi:
    - **Khả năng đọc**: Các giá trị của kiểu liệt kê enum có tên rõ ràng, giúp cho mã dễ đọc và hiểu hơn.
    - **Khả năng bảo trì**: Các giá trị của kiểu liệt kê enum được định nghĩa rõ ràng, giúp cho việc bảo trì mã dễ dàng hơn.
    - **Ngăn ngừa lỗi**: Các giá trị của kiểu liệt kê enum được giới hạn trong một tập hợp các giá trị có thể, giúp ngăn ngừa lỗi.

Lưu ý khi sử dụng kiểu liệt kê enum:

- Kiểu liệt kê enum có một số điểm cần lưu ý khi sử dụng:
    - Kích thước: Các giá trị của kiểu liệt kê enum chiếm một lượng bộ nhớ cố định, bất kể số lượng giá trị của kiểu liệt kê enum.
    - Tương thích: Các giá trị của kiểu liệt kê enum không tương thích với các kiểu dữ liệu nguyên tố.

Đây là chương trình được tạo ra bởi ý tưởng này:

```
enum modeType {UPPERCASE, LOWERCASE, PUNCTUATION};
int number;
modeType mode = UPPERCASE;
cout << "Enter some numbers ending with -1: ";
do {
   cin >> number;
   cout << "Number read: " << number;
   switch (mode) {
      case UPPERCASE:
         number = number % 27;
         cout << ". Modulo 27: " << number << ". ";
         if (number == 0) {
            cout << "Switch to LOWERCASE";
            mode = LOWERCASE;
         }
         break;
      case LOWERCASE:
         number = number % 27;
         cout << ". Modulo 27: " << number << ". ";
         if (number == 0) {

                     cout << "Switch to PUNCTUATION";
                     mode = PUNCTUATION;
                  }
                  break;
               case PUNCTUATION:
                  number = number % 9;
                  cout << ". Modulo 9: " << number << ". ";
                  if (number == 0) {
                     cout << "Switch to UPPERCASE";
                     mode = UPPERCASE;
                  }
                  break;
            }
            cout << "\n";
         } while (number != -1);

         Enter some numbers ending with -1: 2 1 0 52 53 54 55 6 7 8 9 10 -1
         Number read: 2. Modulo 27: 2.
         Number read: 1. Modulo 27: 1.
         Number read: 0. Modulo 27: 0. Switch to LOWERCASE
         Number read: 52. Modulo 27: 25.
         Number read: 53. Modulo 27: 26.
         Number read: 54. Modulo 27: 0. Switch to PUNCTUATION
         Number read: 55. Modulo 9: 1.
         Number read: 6. Modulo 9: 6.
         Number read: 7. Modulo 9: 7.
         Number read: 8. Modulo 9: 8.
         Number read: 9. Modulo 9: 0. Switch to UPPERCASE
         Number read: 10. Modulo 27: 10.
         Number read: -1. Modulo 27: -1.
```

Phân tích:

- **Xác định chế độ liệt kê**: Mã này xác định một bảng liệt kê có tên **modeType** với ba giá trị: UPPERCASE, LOWERCASE và PUNCTUATION.Bảng liệt kê này thể hiện các chế độ có thể có để xử lý số đầu vào
- **Khởi tạo các biến**: Mã khai báo một biến số nguyên có tên là **number** để lưu trữ các số đầu vào và một biến có tên là **mode** thuộc loại **modeType** để theo dõi chế độ hiện tại. Biến **mode** được khởi tạo thành UPPERCASE, cho biết chế độ xử lý ban đầu là chữ hoa
- Câu lệnh **cout << "Enter some numbers ending with -1: ";** nhắc nhở người dùng nhập một dãy số kết thúc bằng -1.
- **Vòng xử lý chính**: Vòng lặp **do-while** lặp lại miễn là số đầu vào không phải là -1 và bên trong vòng lặp, các bước sau sẽ xảy ra:
    a. **Đọc số đầu vào**: Mã đọc một số nguyên từ đầu vào của người dùng và lưu nó vào biến **number**.
    b. **Hiển thị số đầu vào**: Mã in số đầu vào đã đọc ra console.
    c. **Xử lý đầu vào dựa trên chế độ**: Mã sử dụng câu lệnh **switch** để xác định quá trình xử lý dựa trên chế độ hiện tại (biến **mode):**
        1. **Chế độ CHỮ HOA**: Nếu chế độ hiện tại là CHỮ HOA, mã sẽ tính modulo 27 của số đầu vào, in kết quả và kiểm tra điều kiện chuyển đổi chế độ. Nếu kết quả modulo là 0, mã sẽ in thông báo cho biết chuyển đổi chế độ và đặt biến **mode** thành LOWER CASE.
        2. **Chế độ chữ thường**: Nếu chế độ hiện tại là chữ thường, mã sẽ tính modulo 27 của số đầu vào, in kết quả và kiểm tra điều kiện chuyển đổi chế độ. Nếu kết quả modulo là 0, mã sẽ in thông báo cho biết chuyển đổi chế độ và đặt biến **mode** thành dấu câu.
        3. **Chế độ dấu câu**: Nếu chế độ hiện tại là dấu câu, mã sẽ tính modulo 9 của số đầu vào, in kết quả và kiểm tra điều kiện chuyển đổi chế độ. Nếu kết quả modulo là 0, mã sẽ in thông báo cho biết chuyển đổi chế độ và đặt biến **mode** thành CHỮ HOA.
- **Kết thúc vòng lặp**: Vòng lặp **do-while** tiếp tục cho đến khi số đầu vào không phải là -1. Khi số đầu vào là -1, vòng lặp sẽ kết thúc.

Bước cuối cùng trong quá trình phát triển một chương trình hoàn chỉnh tích hợp cá danh sách mã riêng lẻ vào một giải pháp gắn kết. Có 2 tiếp cận khả thi cho sự tích hợp này

a. **Tích hợp tăng dần**: Cách tiếp cận này liên quan đến việc kết hợp các đoạn mã nhỏ hơn và xây dựng chúng dần dần. Tích hợp tăng dần là phương pháp thử nghiệm thường được dùng trong các dự án Agile* trong đó một mô-đun được thử nghiệm, sau đó được tích hợp với một mô-đun khác. Sự tích hợp đó được kiểm tra và sau đó một mô-đun hoặc thành phần khác sẽ được thêm vào. Thay vì tích hợp mọi thứ cùng một lúc và thử nghiệm, việc tích hợp được thực hiện tăng dần khi các phần bổ sung được thêm vào phần chính.

Mục tiêu của loại thử nghiệm này là nhận được phản hồi sớm hơn cho các nhà phát triển và giúp tách biệt các vấn đề. Nếu mô-đun A và B hoạt động tốt cùng nhau nhưng có lỗi xảy ra khi thêm mô-đun C thì điều đó sẽ giúp chỉ ra vấn đề có thể xảy ra ở đâu. Các vấn đề cơ bản có thể được tìm thấy sớm hơn và khắc phục mà không ảnh hưởng đến các mô-đun khác. Khi các lỗi được phát hiện sớm trong các cụm lắp ráp nhỏ hơn, việc sửa chữa sẽ hiệu quả hơn và ít tốn kém hơn. Cả nhà phát triển và người thử nghiệm đều có thể thực hiện thử nghiệm tích hợp gia tăng.

Nhược điểm là có thể tốn thời gian và lặp đi lặp lại. Tuy nhiên, nếu các thành phần được yêu cầu phải hoạt động độc lập cũng như tích hợp với nhau thì việc kiểm thử này là một bước không thể bỏ qua.

Điều bắt buộc là những người thực hiện tích hợp tăng dần phải có nền tảng vững chắc về các kỹ thuật thử nghiệm liên quan để phát triển và sử dụng chiến lược hiệu quả.

Ví dụ: Ta có thể bắt đầu bằng cách hợp nhất mã để đọc các số được phân tách bằng dấu phẩy với logic chuyển đổi chế độ từ danh sách gần đây nhất. Điều này cho phép thử nghiệm và gỡ lỗi.

```
#include<iostream>

enum Mode {UPPERCASE,LOWERCASE,PUNCTUATION};

int main() {

int number;

Mode mode = UPPERCASE;

char result;

bool done=false;

while (!done){

std::cout<< "Nhập số:";

std::cin>>number;

switch (mode){

case UPPERCASE:

result=(char)(number % 27 + 'A');

break;

case LOWERCASE:

result=(char)(number %27 + 'a');

break;

case PUNCTUATION:
```

```
result=(char)(number % 9+ '!');
break;
}
if (number == -1){
done=true;
} else {
std::cout<< "Kết quả:" <<result<<std:endl;
if(number%27==0){
mode=(Mode)((mode+1)%3);
}
}
}
return 0;
}
```

Ví dụ này bắt đầu việc khai báo hai biến **number** và **mode** để lưu trữ giá trị số được nhập vào và trạng thái hiện tại. Sau đó, chương trình bắt đầu vòng lặp đọc và xử lý đầu vào. Trong mỗi lần lặp, chương trình đọc giá trị số đầu vào và sử dụng nó để chuyển đổi sang chữ cái hoặc ký hiệu dấu câu.Kết quả được lưu trữ trong biến **result**.

Chương trình cũng kiểm tra trạng thái kết thúc. Nếu đầu vào là -1 vòng lặp sẽ kết thúc. Nếu không, chương trình in kết quả và kiểm tra điều kiện chuyển đổi trạng thái. Nếu giá trị số chia hết cho 27, chương trình sẽ chuyển đổi trạng thái thành trạng trạng thái tiếp theo.

Ví dụ này minh họa cách tích hợp tăng dần có thể được sử dụng để xây dựng một chương trình phức tạp. Chương trình bắt đầu bằng việc tích hợp mã để đọc và chuyển đổi giá trị số. Sau đó, nó tích hợp mã để chuyển đổi trạng thái. Cuối cùng, nó tích hợp mã để kiểm tra trạng thái kết thúc.

Để thực hiện tích hợp tăng dần, bạn có thể sử dụng các kỹ thuật sau:

- Tách mã thành các phần nhỏ, dễ quản lý.
- Thử nghiệm từng thành phần một trước khi tích hợp chúng vào chương trình.
- Sử dụng các biến và hàm để truyền dữ liệu giữa các thành phần.

Tích hợp tăng dần là một kỹ thuật hiệu quả để xây dựng phần mềm chất lượng cao. Nó giúp phát hiện và giải quyết lỗi sớm trong quá trình phát triển, đồng thời cải thiện khả năng bảo trì của mã.

Về bản chất , quá trình tích hợp bao gồm việc kết hợp cẩn thận các danh sách mã riêng lẻ để đảm bảo hoạt động liền mạch và đạt được chức năng mong muốn. Điều này có thể yêu cầu điều chỉnh và sửa đổi để đảm bảo tính tương thích và luồng dữ liệu giữa các thành phần khác nhau

b. Phân rã chức năng : Phân rã chức năng là quá trình thực hiện một quy trình phức tạp và chia nhỏ nó thành các phần nhỏ hơn, đơn giản hơn , cách tiếp cận này tập trung vào việc chuyển đổi danh sách mã riêng lẻ thành các hàm có thể sử dụng lại. Chương trình chính sau đó sẽ gọi các hàm này để thực hiện quá trình xử lý cần thiết. Điều này thúc đẩy tính mô đun hóa và khả năng bảo trì mã.

Ví dụ:

```
#include<iostream>
enum Mode {UPPERCASE,LOWERCASE,PUNCTUATION};
char to_char(int number,Mode mode){
switch (mode){
case UPPERCASE:
return (char)(number % 27 + 'A');
case LOWERCASE:
return (char)(number % 27 + 'a');
case PUNCTUATION:
return (char)(number % 9 + '!');
}
}
int main(){
int number;
Mode mode = UPPERCASE;
char result;
bool done = false;
while(!done){
std::cout<< " Nhập số:";
std::cin>>number;
result = to_char(number,mode);
if (number == -1){
done == true;
} else {
std::cout << "Kết quả:"<< result<< std::endl;
if(number % 27 == 0){
mode = (Mode)((mode + 1) % 3);
```

```
}
}
}
return 0;
}
```

Ví dụ này bắt đầu bằng việc khai báo một hàm **to_char ()** để chuyển đổi số thành chữ cái hoặc ký hiệu dấu câu. Hàm này nhận hai tham số là giá trị số cần chuyển đổi và trạng thái hiện tại.

Chương trình chính sau đó bắt đầu bằng việc đọc giá trị số từ đầu vào. Sau đó, gọi hàm **to_char()** để chuyển đổi giá trị số thành chữ cái hoặc ký hiệu dấu câu. Kết quả được lưu trữ trong biến **result**.

Chương trình cũng kiểm tra trạng thái kết thúc. Nếu đầu vào là -1, vòng lặp sẽ kết thúc. Nếu không, chương trình in kết quả và kiểm tra điều kiện chuyển đổi trạng thái. Nếu giá trị số chia hết cho 27, chương trình sẽ chuyển đổi trạng thái thành trạng thái tiếp theo.

Ví dụ này minh hoa cách phân rã chức năng có thể sử dụng để cải thiện khả năng đọc và hiểu của mã. Hàm **to_char ()** tách biệt logic chuyển đổi số thành chữ cái hoặc ký hiệu dấu câu khỏi logic chính của chương trình. Điều này giúp cho mã dễ đọc và dễ hiểu hơn.

Để thực hiện phân rã chức năng, bạn có thể dùng kỹ thuật sau:

- Xác định các khối logic trong mã.
- Tạo các hàm để thực hiện từng khối logic
- Thử nghiệm từng hàm một trước khi tích hợp chúng vào chương trình.

Phân rã chức năng là một kỹ thuật hiểu quả để cải thiện chất lượng mã.Nó giúp mã dễ đọc,dễ hiểu và dễ bảo trì hơn.

Lợi ích của việc phân rã chức năng là khi bạn bắt đầu viết mã, bạn đang làm việc trên các thành phần đơn giản mà bạn có thể làm việc với ứng dụng của mình. Do đó, việc phát triển và thử nghiệm các thành phần đó trở nên dễ dàng hơn nhiều(chưa kể bạn có thể thiết kế mã và dự án phù hợp với nhu cầu của mình tốt hơn).

Nhược điểm là đầu tư thời gian. Để thực hiện phân rã chức năng trên một hệ thống phức tạp cần nhiều thời gian hơn trước khi bắt đầu mã hóa.

Việc lựa chọn giữa tích hợp tăng dần và phân rã chắc năng phụ thuộc vào bối cảnh cụ thể và sở thích của người lập trình. Cả hai cách tiếp cận đều có thể dẫn đến một chương trình có cấu trúc tốt và có thể duy trì được

Thảo luận về một kỹ thuật lập trình được gọi là “sàng lọc lũy tiến” hoặc “sàng lọc từng bước”. Kỹ thuật này liên quan tới việc chia nhỏ, dễ quản lý hơn. Mỗi bước sau đó được giải quyết và các giải pháp được kết hợp để giải quyết vấn đề ban đầu.

c. Sàng lọc lũy tiến

Sàng lọc lũy tiến là cách viết mã hiệu quả hơn là cố gắng viết toàn bộ chương trình từ đầu. Điều này là do nó cho phép lập trình viên tập trung vào từng bước một mà không bị choáng ngợp bởi sự phức của toàn bộ vấn đề. Ngoài ra, nó giúp tránh mắc lỗi vì mỗi bước đều được kiểm tra và kiểm tra trước khi chuyển sang bước theo

Ví dụ viết chương trình in ra danh sách các số. Chương trình có thể được chia thành các bước sau:

- Viết mã để in số đầu tiên.
- Viết mã để in số thứ hai.
- Lặp lại bước 2 đến khi tất cả các số được in

Bằng cách làm theo các bước này, người lập trình có thể dần dần xây dựng giải pháp cho vấn đề mà không bị lạc vào sự phức tạp của toàn bộ chương trình.

Sàng lọc lũy tiến là một công cụ có giá trị đối với các lập trình viên, vì nó có thể giúp cải thiện chất lượng và khả năng bảo trì của mã.

Dưới đây là một số lợi ích của việc sử dụng sàng lọc lũy tiến:

- **Cải thiện chất lượng mã**: Bằng cách chia nhỏ vấn đề thành các bước nhỏ hơn, lập trình viên có thể tập trung vào việc viết mã rõ ràng, ngắn gọn và được kiểm tra tốt.
- **Giảm lỗi**: Tinh chỉnh lũy tiến giúp giảm lỗi bằng cách cho phép lập trình viên kiểm tra và kiểm tra từng bước trước khi chuyển sang bước tiếp theo.
- **Dễ bảo trì hơn**: Mã được viết bằng cách sử dụng tinh chỉnh lũy tiến sẽ dễ bảo trì hơn vì nó dễ hiểu và dễ sữa đổi hơn

Nhìn chung, sàng lọc lũy tiến là một kỹ thuật có giá trị có thể giúp các lập trình viên viết mã tốt hơn.

*Agile Project Management là một quy trình phát triển áp dụng cách tiếp cận lặp đi lặp lại để xây dựng phần mềm. Các nhóm sử dụng phương pháp Agile để lập kế hoạch phát hành sản phẩm, sau đó thực hiện nhiệm vụ trong các chu kỳ Sprint có giới hạn thời gian để liên tục tạo ra phần mềm mới

Các phương pháp lặp lại hoặc các bước tăng dần để hoàn thành một dự án. Các phương pháp lặp lại trong được sử dụng trong các dự án phát triển phần mềm để thúc đẩy tốc độ và khả năng thích ứng vì lợi ích của việc lặp lại là bạn có thể điều chỉnh khi thực hiện thay vì đi theo một đường dẫn tuyến tính.

Một trong những mục tiêu của cách tiếp cận linh hoạt hoặc lặp đi lặp lại là mang lại lợi ích trong suốt quá trình thay vì chỉ ở giai đoạn cuối. Về cốt lõi, các dự án linh hoạt phải thể hiện các giá trị và hành vi trọng tâm của sự tin cậy, linh hoạt, trao quyền và hợp tác.

TÀI LIỆU THAM KHẢO :

-“ Pure Puzzles”, tên sách tham khảo “ THINK LIKE A PROGRAMMER AN INTRODUCTION TO CREATIVE PROBLEMS SOLVING” tác giả V.ANTON SPRAUL.

-“LUHN”, https://viblo.asia/p/luhn-algorithm-phuong-phap-modulus-10-EoW4oxjBJml (Truy cập ngày 10/11/2023)

-“ Thuật toán LUHN ”, https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Luhn

(Truy cập ngày 10/11/2023)

-“Breaking down the problem, https://youtu.be/lY4OZBZNflo?si=JzxXfTqyMp_8tdJH

(Truy cập ngày 11/11/2023)

-“Giải bài tập LUHN”, https://youtu.be/wsphC8V36i0?si=VZgVz7ZpIsyfSXJF (Truy cập ngày 11/11/2023)

-“Incremental Integration Testing”,https://www.qamentor.com/testing-coverage/functional-testing/incremental-integration-testing/(Truy cập 4/12/2023)

-“What is Agile project management?”, https://www.apm.org.uk/resources/find-a-resource/agile-project-management/ (Truy cập ngày 4/12/2023)

-“9.0 Kiểu liệt kê (enum)”, https://cpp.daynhauhoc.com/9/0-kieu-liet-ke-enum/(Truy cập 2/12/2023)

-“Stepwise Refinement”, https://learnlearn.uk/alevelcs/stepwise-refinement/ (Truy cập 4/12/2023)

-Progressive Refinement, https://www.promodel.com/onlinehelp/promodel/80/C-03%20-%20Progressive%20Refinement.htm (Truy cập 4/12/2023)

-“What is Funtional Decomposition?”, answered Jun 3,2009 at 23:51 by Justin Niessner, https://stackoverflow.com/questions/947874/what-is-functional-decomposition (truy cập ngày 4/12/2023)

-“What is code reuse and why is it important”, https://www.opslevel.com/resources/what-is-code-reuse-and-why-is-it-important (Truy cập ngày 19/11/2023)

- “ How to reuse code using version control systems?”, https://www.linkedin.com/advice/1/how-can-you-reuse-code-using-version-control-systems, (Truy cập ngày 21/11/2023).

-“Sử dụng Subversion SVN để quản lý code trong project”, https://www.youtube.com/watch?v=e3lSjTEjcAk (Truy cập ngày 21/11/2023).

-“Storing code in GitHub”, https://qatechhub.com/storing-code-using-git-in-github-or-bigbucket/#:~:text=Storing%20code%20in%20GitHub%3A&text=Create%20a%20new%20repository%20by,The%20repository%20is%20created%20now (Truy cập ngày 21/11/2023).

-“Why does this work”, https://www.codecademy.com/forum_questions/503fd13491601700020144f2 (Truy cập ngày 10/11/2023).

-“Thuật toán tìm số Palindrome”, https://dothanhspyb.com/thuat-toan-tim-so-palindrome/ (Truy cập ngày 10/11/2023).

-“Recursion on string: What does the line ‘ return s[0]==s[-1] and isPal(s[1:-1]’do?”, https://stackoverflow.com/questions/53162619/recursion-on-string-what-does-the-line-return-s0-s-1-and-ispals1-1 (Truy cập ngày 10/11/2023)

-“Side effects”, “Side effect can cause order of evaluation issues”, “The sequencing of side effects”, https://www.learncpp.com/cpp-tutorial/increment-decrement-operators-and-side-effects/ (Truy cập ngày 9/11/2023)