Chương này trình bày một số ví dụ mà sẽ được dùng trong các chương tiếp theo nhằm minh họa cho các phương pháp kiểm thử. Các ví dụ này gồm: bài toán tam giác và hàm NextDate tương đối phức tạp về mặt lôgic. Các ví dụ này liên quan đến một số vấn đề mà người kiểm thử sẽ gặp trong quá trình kiểm thử. Khi bàn về kiểm thử tích hợp và kiểm thử hệ thống trong chương 10, ta sẽ dùng ví dụ về một phiên bản đơn giản của máy rút tiền tự động (ATM).

Trong chương này các ví dụ mức đơn vị, cài đặt bằng ngôn ngữ C, sẽ được trình bày cho mục đích kiểm thử cấu trúc. Các mô tả mức hệ thống của máy ATM dưới dạng một tập các sơ đồ dòng dữ liệu và máy hữu hạn trạng thái sẽ được trình bày trong các chương tiếp theo.

1. Bài toán tam giác

Kể từ ngày được công bố lần đầu dưới dạng một ví dụ của kiểm thử cách đây 30 năm [Gru73], bài toán tam giác đã được nhắc tới trong nhiều bài báo và sách về kiểm thử, chẳng hạn trong các tài liệu [Gru73, BL75, Mye75, S.82, AJ83, AJ84, Mal87, Bil88].

1.1 Phát biểu bài toán

Bài toán tam giác nhận ba số nguyên làm các dữ liệu đầu vào; các dữ liệu này là số đo các cạnh của một tam giác. Đầu ra của chương trình là loại của tam giác xác định bởi ba cạnh ứng với các số đo này: tam giác đều, tam giác cân, tam giác thường, hoặc không là tam giác. Ta sẽ dùng các từ tiếng Anh làm dữ liệu đầu ra tương ứng cho các loại này như lấy từ ví dụ nguyên thủy: Equilateral, Isosceles, Scalene, hoặc NotATriangle. Bài toán này đôi khi được mở rộng với đầu ra thứ năm là tam giác vuông (right triangle). Trong các bài tập, ta sẽ dùng bài toán mở rộng như vậy.

1.2 Nhận xét

Một trong các lý do làm bài toán này được sử dụng rất phổ biến có thể là vì nó tiêu biểu cho việc định nghĩa không đầy đủ làm phương hại đến việc trao đổi thông tin giữa khách hàng, người phát triển và người kiểm thử. Đặc tả này giả thiết rằng người phát triển biết các chi tiết về tam giác, đặc biệt tính chất sau của tam giác: tổng của hai cạnh bất kỳ cần thực sự lớn hơn cạnh còn lại. Nếu *a, b* và *c* ký hiệu cho ba cạnh của tam giác thì tính chất trên được biểu diễn chính xác bằng ba bất đẳng thức toán học $a < b + c$, $b < a + c$ và $c < a + b$. Nếu bất kỳ một trong ba bất đẳng thức này không được thỏa mãn thì *a, b* và *c* không tạo thành ba cạnh của một tam giác. Nếu cả ba cạnh đều bằng nhau, chúng tạo thành tam giác đều, nếu chỉ có một cặp cạnh bằng nhau, chúng tạo thành tam giác cân và nếu không có cặp cạnh nào bằng nhau thì chúng là độ dài ba cạnh của một tam giác thường. Một người kiểm thử giỏi có thể làm rõ ý nghĩa bài toán này hơn nữa bằng việc đặt giới hạn cho các độ dài của các cạnh. Ví dụ, câu trả lời nào cho trường hợp khi đưa vào

chương trình ba số −5, −4 và −3? Ta sẽ đòi hỏi là các cạnh phải ít nhất là bằng 1, và khi đó ta cũng có thể khai báo giới hạn của cận trên, chẳng hạn 20000.

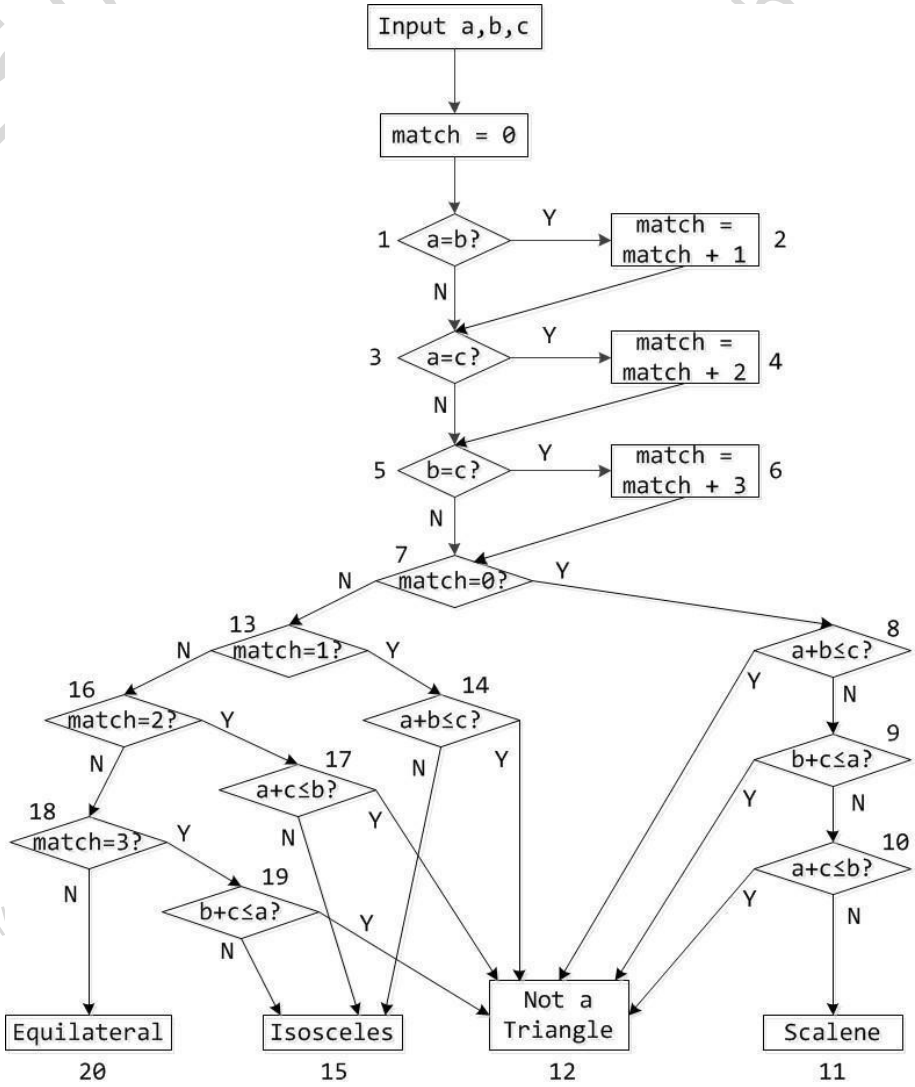
1.3 Cài đặt truyền thống

Cài đặt truyền thống của ví dụ cổ điển này có kiểu tựa FORTRAN [S.82]. Tuy nhiên, chúng tôi chuyển cài đặt của ví dụ này sang ngôn ngữ C để thống nhất với các ví dụ khác trong giáo trình này. Sơ đồ khối của ví dụ này được biểu thị trong hình 2.1. Các số của khối trong sơ đồ này tương ứng với các chú giải trong chương trình sau đây. Một cài đặt có cấu trúc hơn sẽ được cho trong mục 2.1.4.

Biến match được dùng để ghi nhận sự bằng nhau giữa các cặp cạnh. Nếu hai cạnh bằng nhau, chẳng hạn $a = c$, thì chỉ cần so sánh $a + c$ với b (do $b > 0$, $a + b > c$ sẽ phải thỏa mãn vì $a = c$). Nhờ quan sát này, chúng ta có thể rút gọn số các so sánh cần làm. Cái giá phải trả cho tính hiệu quả này chỉ là sự rõ ràng và dễ kiểm thử!.

Trong các chương tiếp theo, ta sẽ thấy lợi thế của phiên bản này khi bàn đến các đường đi thực thi được của chương trình. Đó là lý do tốt nhất để giữ lại bản này.

```
int main(){
int          a, b, c, match;
printf("Enter 3 sides (a, b, c) of a triangle \n"); printf("a = ");
scanf("%d",&a);
printf("b = ");
scanf("%d",&b);
printf("c = ");
scanf("%d",&c);
printf ("Side A is %d\n", a); printf ("Side B is %d\n", b);
```



Hình 2.1: Sơ đồ khối cho cài đặt chương trình tam giác truyền thống.

```
printf ("Side C is %d\n", c); match = 0;
if(a == b) {1}
match = match + 1;          {2}
if(a == c) {3}


match = match + 2;          {4}
if(b == c) {5}
match = match + 3;          {6}
if(match == 0)      {7}
```

```
if((a+b) <= c)      {8}
printf("Not a Triangle");   {12.1} else if((b+c) <= a)          {9}
printf("Not a Triangle");          {12.2}   else if((a+c) <= b)                {10}
printf("Not a Triangle");   {12.3}
else printf("Triangle is Scalene");{11}
else
if(match == 1)      {13}
if((a+c)  <=  b)     {14}
printf("Not a Triangle");    {12.4} else printf("Triangle is Isosceles");        {15.1}
else
if(match == 2)      {16}
if((a+c)  <=  b)     {17}
printf("Not a Triangle");          {12.5}
else  printf("Triangle's  Isoscel.");{15.2}
else if(match == 3)                        {18} if((b+c) <= a)       {19}
printf("Not a Triangle");   {12.6} else
printf("Triangle's Isoscel.");{15.3}
else printf("Triangle's Equilat.");  {20}
return 0;
}//the end.
```

Lưu ý là có sáu cách để đi đến nút "Not A Triangle" (12.1 – 12.6) và có ba cách để đi đến nút "Isosceles" (15.1 – 15.3).

1.4 Cài đặt có cấu trúc

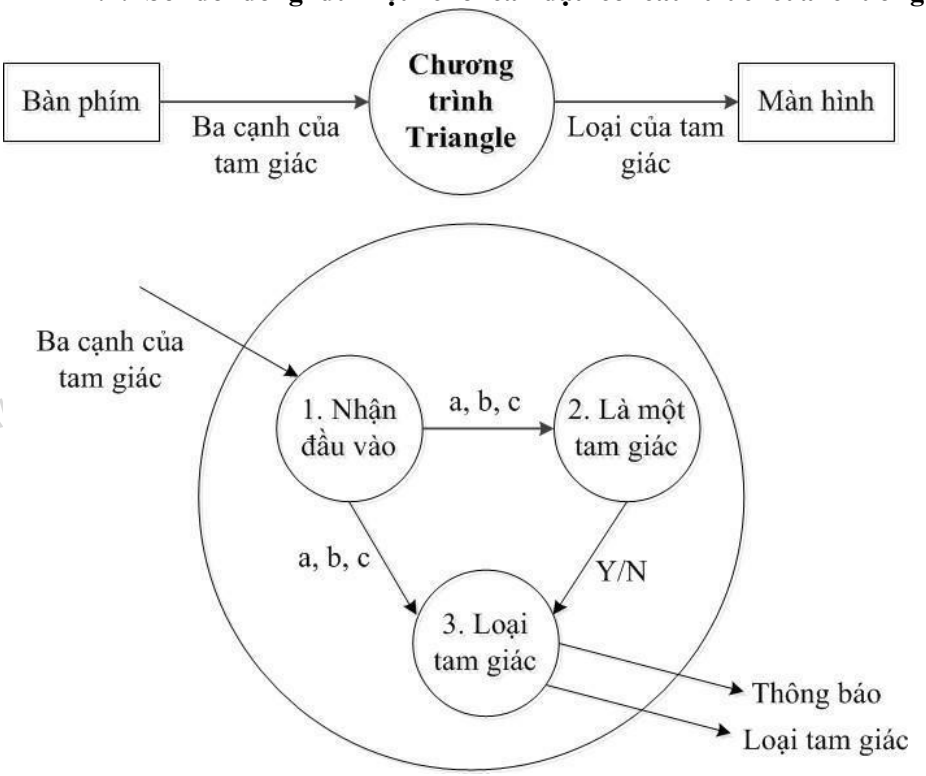
Hình 2.2 là một mô tả sơ đồ dòng dữ liệu của chương trình tam giác. Ta có thể cài đặt bằng một chương trình chính và bốn thủ tục. Vì ta sẽ dùng ví dụ này cho việc kiểm thử đơn vị, bốn thủ tục đã được kết hợp thành một chương trình C. Các dòng chú giải liên kết các đoạn mã với việc phân rã cho trong hình 2.2.

```
int main(){
int a, b, c, IsATriangle;

//Function 1: Get Input
```

```
printf("Enter 3 sides (integers) of a triangle"); printf("a = ");
scanf("%d",&a);
printf("b = ");
scanf("%d",&b);
printf("c = ");
scanf("%d",&c);
printf ("Side A is %d\n", a); printf ("Side B is %d\n", b); printf ("Side C is %d\n", c);

//Function 2: Is A Triangle?
if((a < b + c) && (b < a + c) && (c < a + b)) IsATriangle = 1;
else IsATriangle = 0;

//Function 3: Determine Triangle Type if(IsATriangle)
if((a == b) && (b == c)) printf("Triangle is Equilateral");
else if((a != b) && (a != c) && (b != c))

printf("Triangle is Scalene"); else printf("Triangle is Isosceles");
else printf("Not a Triangle");

return 0;
}//the end
```

Lưu ý: Function 4 và Output Controller đã được kết hợp thành các lệnh trong Function 3.

Hình 2.2: Sơ đồ dòng dữ liệu cho cài đặt có cấu trúc của chương trình tam giác.



2. Hàm NextDate (ngày kế tiếp)

Độ phức tạp của chương trình tam giác nằm ở các mối quan hệ giữa dữ liệu đầu vào và dữ liệu đầu ra. Hàm NextDate nhằm minh họa một loại độ phức tạp khác: mối quan hệ giữa các biến đầu vào.

2.1 Phát biểu bài toán

NextDate là một hàm có ba biến biểu diễn ngày, tháng và năm là day, month và year. Hàm này trả về ngày kế tiếp của ngày đầu vào. Các biến day, month, year có các giá trị số thỏa mãn các ràng buộc: $1 \leq \text{day} \leq 31$, $1 \leq \text{month} \leq 12$, $1812 \leq \text{year} \leq 2012$.

2.2 Nhận xét

Có hai nguyên nhân tạo nên độ phức tạp của hàm NextDate: độ phức tạp nêu trong các ràng buộc trên đây của miền dữ liệu đầu vào, và quy tắc phân biệt giữa năm nhuận và năm không nhuận. Do trung bình một năm có 365, 2422 ngày, năm nhuận được dùng để giải quyết ngày "vượt trội". Nếu ta chấp thuận cứ bốn năm lại có một năm nhuận thì sẽ có một sai số nhỏ. Lịch Gregorian (đề xuất năm 1582 bởi Giáo hoàng Gregory) giải quyết vấn đề này bằng cách điều chỉnh các năm nhuận theo năm thế kỷ (những năm chia hết cho 100). Do đó, một năm là nhuận nếu nó chia hết cho 4 nhưng không là năm thế kỷ. Các năm thế kỷ là nhuận khi và chỉ khi nó là bội của 400 [Ing61, fS91]. Do đó các năm 1992, 1996 và 2000 là năm nhuận, nhưng năm 1900 lại không phải là năm nhuận.

Hàm NextDate cũng minh họa một khía cạnh của kiểm thử phần mềm. Ta thường thấy các ví dụ về luật Zipf - nói rằng 80% các hoạt động xảy ra tại chỉ 20% của không gian. Ta cũng thấy ở đây phần lớn mã nguồn (80% các hoạt động) được dành cho các năm nhuận (20% của không gian).