Mục tiêu chính của chương này là giải thích rõ tầm quan trọng của việc phân tích thuật toán cùng với mối liên hệ và sự ảnh hưởng qua lại giữa dữ liệu và thuật toán. Để thực hiện được điều này, chúng ta sẽ bắt đầu từ những khái niệm và định nghĩa cơ bản về dữ liệu, thuật toán sau đó mở rộng sang những vấn đề quan trọng hơn độ phức tạp thuật toán, độ phức tạp chương trình. Cuối cùng, chúng ta sẽ xem xét đến quy trình giải quyết một vấn đề trong khoa học máy tính bằng thuật toán.

1. Kiểu và cấu trúc dữ liệu

Trước khi định nghĩa chính xác các khái niệm về kiểu dữ liệu (*data types*), biến (*variables*) ta xem xét lại với những gì ta đã từng biết trước đây trong toán học. Chẳng hạn khi ta giải phương trình:

$$x^2-2y-2=1$$

Tiếp cận bằng toán học ta nói nghiệm của phương trình trên là tập các cặp (x, y) sao cho $x^2-2y-2=1$. Ví dụ cặp (1, -1) là một nghiệm của phương trình. Tiếp cận bằng tin học ta sẽ thấy phương trình trên có hai tên là x và y. Nếu x và y có giá trị tương ứng là *1*, *-1* thì đó là một nghiệm của phương trình. Trong khoa học máy tính cũng gọi x và y là hai biến và các bộ giá trị của x và y được gọi là dữ liệu.

Hai biến x và y có thể nhận giá trị trong các miền khác nhau. Để giải được phương trình ta cần phải xác định được miền giá trị của hai biến x và y. Ví dụ, x, y xác định trong miền các số nguyên (10, 20, 30,..), số thực (0.23, 0.55,…) hoặc (0, 1). Để xác định miền giá trị của các biến, trong khoa học máy tính sử dụng một từ khóa đại diện cho một tập các giá trị còn được gọi là một kiểu dữ liệu (*a data type*). Ta sẽ bắt đầu bằng cách tổng quát hóa những khái niệm cơ bản này theo cách tiếp cận của khoa học máy tính.

1.1 Kiểu dữ liệu

Kiểu dữ liệu (*a data type*) là một tên hay từ khóa dùng để chỉ tập các đối tượng dữ liệu cùng các phép toán trên nó. Ví dụ trong C++, từ khóa *int* dùng để chỉ tập các số nguyên có độ lớn biểu diễn bằng 2 byte (tùy thuộc vào các compiler) cùng với các phép toán số học, các phép toán so sánh, các phép toán cấp bít, các phép toán dịch chuyển bit. Từ khóa *float* dùng để chỉ tập các số thực có độ chính xác đơn có độ lớn được biểu diễn bằng 4 byte (tùy thuộc vào các compiler) cùng với các phép toán số học, các phép toán so sánh. Không có phép lấy phần dư, các phép toán thao tác cấp bít với kiểu dữ liệu *float*. Kiểu dữ liệu được

chia thành hai loại kiểu dữ liệu cơ bản hay còn gọi là kiểu dữ liệu nguyên thủy và các kiểu dữ liệu do người dùng định nghĩa.

Kiểu dữ liệu nguyên thủy (*primitive data types*) các kiểu dữ liệu được định nghĩa bởi hệ thống (*system defined data type*) được gọi là các kiểu dữ liệu nguyên thủy. Thông thường, các ngôn ngữ lập trình cung cấp ba kiểu dữ liệu nguyên thủy đó là ký tự (*character*), số (*numeric*), và kiểu logic (*bool*). Kiểu dữ liệu ký tự được chia thành hai loại ký tự ASCII (*char*) và ký tự unicode (*wchar_t*). Kiểu dữ liệu số cũng được chia thành hai loại: số kiểu số nguyên (*integer*) và kiểu số thực (*real*). Kiểu số nguyên được chia thành ba loại: số nguyên nhỏ (*int*), số nguyên lớn (*long*), số nguyên rất lớn (*long long*). Kiểu số thực được chia làm hai loại: số thực có độ chính xác đơn (*float*) và số thực có độ chính xác kép (*double*). Dữ liệu kiểu bool chỉ định nghĩa bộ hai giá trị đúng (*true*) và sai (*false*).

Đương nhiên, hai từ khóa khác nhau đại diện cho hai kiểu dữ liệu khác nhau. Quan sát này chỉ mang tính hình thức vì ta có thể quan sát được bằng mắt. Sự khác biệt bên trong giữa các kiểu dữ liệu là không gian bộ nhớ dùng để biểu diễn kiểu và các phép toán dành cho mỗi biến thuộc kiểu. Không gian nhớ dành cho kiểu phụ thuộc vào compiler của ngôn ngữ lập trình và hệ thống máy tính ta đang sử dụng. Chẳng hạn, kiểu dữ liệu *int* một số compiler dùng 2 byte biểu diễn, một số compiler dùng 4 byte để biểu diễn. Các phép toán lấy phần dư (*modulo*), dịch chuyển bít (*bit operations*) định nghĩa cho các số *int*, *long* nhưng không định nghĩa cho các số *float* và *double*. Để xác định độ lớn của kiểu ta có thể sử dụng hàm *sizeof* (*tên kiểu*). Ví dụ dưới đây dùng để xác định không gian nhớ dành cho kiểu.

```
//Ví dụ 1.1. Xác định kích cỡ bộ nhớ biểu diễn kiểu
#include <iostream>
using namespace std;
int main(void){
        cout<<"KÍCH CỠ KIỂU CƠ BẢN"<<endl;
        cout<<"Kích cỡ kiểu bool:"<<sizeof(bool)<<endl;
        cout<<" Kích cỡ kiểu char:"<<sizeof(char)<<endl;
        cout<<" Kích cỡ kiểu wchar_t:"<<sizeof(wchar_t)<<endl;
        cout<<" Kích cỡ kiểu int:"<<sizeof(int)<<endl;
        cout<<" Kích cỡ kiểu long:"<<sizeof(long)<<endl;
        cout<<" Kích cỡ kiểu long long:"<<sizeof(long long)<<endl;
        cout<<" Kích cỡ kiểu float:"<<sizeof(float)<<endl;
        cout<<" Kích cỡ kiểu double:"<<sizeof(double)<<endl;
}
```

Kiểu dữ liệu do người dùng định nghĩa (*user defined data types*) là các kiểu dữ liệu được do người dùng xây dựng bằng cách tổ hợp các kiểu dữ liệu nguyên thủy theo một nguyên tắc nào đó. Chẳng hạn, kiểu mảng (*array*) là dãy có thứ tự các phần tử (*các biến*) có cùng chung một kiểu dữ liệu được tổ chức liên tục nhau trong bộ nhớ. Kiểu xâu ký tự (string) là một mảng mỗi phần tử là một ký tự và có ký tự kết thúc là '\0'. Như vậy, các kiểu dữ liệu không thuộc các kiểu dữ liệu nguyên thủy như mảng, cấu trúc, file đều được xem là các kiểu dữ liệu do người dùng định nghĩa.

Cấu trúc dữ liệu (*data structure*) là phương pháp biểu diễn các đối tượng ở thế giới thực thành một đối tượng dữ liệu được tổ chức và lưu trữ trong máy tính để có thể sử lý một cách hiệu quả. Theo nghĩa này, mảng (*array*), danh sách liên kết (*linked list*), ngăn xếp (*stack*), hàng đợi (*queue*), cây (*tree*), đồ thị (*graph*)… đều được gọi là các cấu trúc dữ liệu. Dựa vào biểu diễn của các cấu trúc dữ liệu, khoa học máy tính chia các cấu trúc dữ liệu thành hai loại: các cấu trúc dữ liệu tuyến tính (*linear data structures*) và các cấu trúc dữ liệu không tuyến tính (*non-linear data structures*). Một cấu trúc dữ liệu được gọi là tuyến tính nếu việc truy cập các phần tử được thực hiện tuần tự nhưng không nhất thiết được tổ chức liên tục. Điều này có nghĩa, các cấu trúc dữ liệu mảng, danh sách liên kết đơn, danh sách liên kết kép đều là các cấu trúc dữ liệu tuyến tính. Một cấu trúc dữ liệu được gọi là

không tuyến tính nếu các phần tử của nó được tổ chức và truy cập không tuần tự. Theo nghĩa này, các cấu trúc dữ liệu cây, graph đều là các cấu trúc dữ liệu không tuyến tính.

Cấu trúc dữ liệu trừu tượng (*Abstract Data types: ADTs*) là phương pháp kết hợp giữa cấu trúc dữ liệu cùng với các phép toán trên dữ liệu cụ thể của cấu trúc dữ liệu. Như vậy, mỗi kiểu dữ liệu ADTs bao gồm hai thành phần:

- *Biểu diễn cấu trúc dữ liệu*.
- *Xây dựng các phép toán trên dữ liệu cụ thể của cấu trúc dữ liệu.*

Theo nghĩa này các cấu trúc dữ liệu danh sách liên kết (*linked list*), ngăn xếp (*stack*), hàng đợi (*queue*), hàng đợi ưu tiên (*priority queue*), cây nhị phân (*binary tree*), đồ thị (*graph*) đều là *ADTs*. Mỗi cấu trúc dữ liệu cụ thể cùng các thao tác trên nó sẽ được trình bày trong những chương tiếp theo của tài liệu.

Đối với mỗi cấu trúc dữ liệu trừu tượng, ta cần quan tâm và nắm bắt được những vấn đề sau:

- **Định nghĩa**: nhằm xác định rõ cấu trúc dữ liệu ADTs ta đang quan tâm đến là gì.
- **Biểu diễn**: nhằm định hình nên cấu trúc dữ liệu ADTs.
- **Thao tác (phép toán)**: những thao tác và phép toán nào được cài đặt trên cấu trúc dữ liệu ADTs.
- **Ứng dụng**: sử dụng cấu trúc dữ liệu ADTs để giải quyết lớp những bài toán nào trong khoa học máy tính.

1.2 Biến

Khi một cấu trúc dữ liệu được thiết lập thì thể hiện cụ thể của cấu trúc dữ liệu đó là các phần tử dữ liệu và ta thường gọi là biến. Biến trong tin học và biến trong toán học cũng có một số điểm khác biệt. Biến trong toán học hoàn toàn là một tên hình thức. Ngược lại, biến trong tin học có tên mang tính hình thức, nhưng tên này được lưu trữ tại một vị trí bộ nhớ xác định được gọi là địa chỉ của biến. Dựa vào địa chỉ của biến, giá trị của biến sẽ được lưu trữ tại địa chỉ ô nhớ dành cho biến trong khi xử lý dữ liệu.

Biến (*variables*): là một tên thuộc kiểu. Trong đó, mỗi tên biến dùng để chỉ bộ ba thành phần: tên (*name*), địa chỉ (*address*), giá trị (*value*). Chẳng hạn khi ta có khai báo *int a =10*; khi đó tên biến là *a*, địa chỉ của biến là *&a*, giá trị của biến là *10*. Trong các ngôn ngữ lập trình, biến cũng được chia thành hai loại: biến toàn cục (global variables) và biến cục bộ (local variables).

Một biến được gọi là biến toàn cục nếu nó được khai báo ngoài tất cả các hàm kể cả hàm main(). Không gian nhớ dành cho biến toàn cục sẽ được cấp phát cho biến ngay từ khi bắt đầu thực hiện chương trình cho đến khi kết thúc thực hiện chương trình. Phạm vi sử dụng biến mang tính toàn cục. Người lập trình được phép truy cập và thay đổi giá trị của biến toàn cục ở bất kể vị trí nào trong chương trình. Một biến được gọi là biến cục bộ nếu nó được khai báo trong thân của hàm kể cả hàm main(). Không gian nhớ dành cho biến toàn cục chỉ được cấp phát sau mỗi lần kích hoạt hàm. Phạm vi sử dụng biến cục bộ chỉ được phép trong nội bộ hàm. Chú ý, khi tên biến toàn cục trùng với tên biến cục bộ thì ưu tiên xử lý dành cho biến cục bộ. Ví dụ dưới đây sẽ minh họa cho biến toàn cục và biến cục bộ.

```
//Ví dụ 1.2. Biến toàn cục và biến cục bộ
#include <iostream> using
namespace std; int a = 10,
b = 20; int Swap (int &a,
int & b){
        int t = a; a=b; b = t;
}
int main(void){
         Swap(a, b); //Tại điểm này ta thao tác với hai biến toàn cục a, b
        cout<<"a = "<<a <<"  b = "<<b<<endl;
        int a = 5, b = 7; //Từ vị trí này trở đi
        Swap(a,b);//Ta hoàn toàn thao tác với hai biến a, b cục bộ
cout<<"a = "<<a <<"  b = "<<b<<endl;
}
```

2. Thuật toán và một số vấn đề liên quan

Như đã trình bày trong Mục 1.1.1, cấu trúc dữ liệu là phương pháp biểu diễn các đối tượng ở thế giới thực thành một đối tượng trong máy tính. Còn thuật toán được hiểu là phương pháp xử lý các đối tượng dữ liệu đã được biểu diễn để đưa ra kết quả mong muốn. Ta có thể tổng quát hóa khái niệm thuật toán như sau.

Định nghĩa thuật toán (Algorithm): Thuật toán F giải bài toán P là dãy các thao tác sơ cấp $F_1, F_2,..,F_N$ trên tập dữ kiện đầu vào (*Input*) để đưa ra được kết quả ra (*Output*).

$F_1F_2..F_N(\ Input\ ) \cdot Ouput.$

- $F = F_1F_2..F_N$ được gọi là thuật toán giải bài toán P. Trong đó, mỗi F_i là các phép toán sơ cấp.

- *Input* được gọi là tập dữ kiện đầu vào hay tập thông tin đầu vào.
- *Output* là kết quả nhận được sau khi thực hiện thuật toán *F* trên tập *Input*.

Ví dụ. Thuật toán tìm USCLN(a, b).

```
int     USCLN ( int a,  int  b) {//đầu vào là số nguyên a, b
        while (b!=0 ) {//lặp trong khi b khác 0
                x = a % b; //lấy x là a mode b
                a = b; //đặt a bằng b
                b =x; //đặt b bằng x
        }
        return(a);//kết quả thực thi thuật toán
}
```

Những đặc trưng cơ bản của thuật toán:

- **Tính đơn định**. Ở mỗi bước của thuật toán, các thao tác sơ cấp phải hết sức rõ ràng, không tạo ra sự lộn xộn, nhập nhằng, đa nghĩa. Thực hiện đúng các bước của thuật toán trên tập dữ liệu đầu vào chỉ cho duy nhất một kết quả.
- **Tính dừng**. Thuật toán không được rơi vào quá trình vô hạn. Thuật toán phải dừng lại và cho kết quả sau một số hữu hạn các bước.
- **Tính đúng**. Sau khi thực hiện tất cả các bước của thuật toán theo đúng quy trình đã định, ta phải nhận được kết quả mong muốn với mọi bộ dữ liệu đầu vào. Kết quả đó được kiểm chứng bằng yêu cầu của bài toán.
- **Tính phổ dụng**. Thuật toán phải dễ sửa đổi để thích ứng được với bất kỳ bài toán nào trong lớp các bài toán cùng loại và có thể làm việc trên nhiều loại dữ liệu khác nhau.
- **Tính khả thi**. Thuật toán phải dễ hiểu, dễ cài đặt, thực hiện được trên máy tính với thời gian cho phép.

Đối với thuật toán ta cần quan tâm đến những vấn đề sau:

- **Biểu diễn thuật toán:** xác định ngôn ngữ để biểu diễn thuật toán.
- **Đánh giá độ phức tạp thuật toán**: ước lượng thời gian và không gian nhớ khi thực hiện thuật toán.
- **Kiểm nghiệm thuật toán**: kiểm nghiệm thuật toán với các bộ dữ liệu thực khác nhau.
- **Cài đặt thuật toán**: cài đặt thuật toán bằng ngôn ngữ lập trình cụ thể.

3. Biểu diễn thuật toán

Có ba ngôn ngữ chính để biểu diễn thuật toán: ngôn ngữ tự nhiên, ngôn ngữ máy tính và ngôn ngữ hình thức.

- **Ngôn ngữ tự nhiên** là phương tiện giao tiếp giữa con người với con người. Ta có thể sử dụng chính ngôn ngữ này vào việc biểu diễn thuật toán.
- **Ngôn ngữ máy tính** là phương tiện giao tiếp giữa máy tính và máy tính. Trong trường hợp này ta có thể sử dụng bất kỳ ngôn ngữ lập trình nào để biểu diễn thuật toán (C, Pascal, Java…).
- **Ngôn ngữ hình thức**. Ngôn ngữ hình thức là phương tiện giao tiếp trung gian giữa con người và hệ thống máy tính. Ví dụ ngôn ngữ sơ đồ khối, ngôn ngữ tựa tự nhiên, ngôn ngữ đặc tả. Đặc điểm chung của các loại ngôn ngữ hình thức là việc sử dụng nó rất gần gũi với ngôn ngữ tự nhiên, rất gần gũi với ngôn ngữ máy tính. Tuy nhiên, ngôn ngữ hình thức lại không phụ thuộc vào ngôn ngữ tự nhiên, không phụ thuộc vào ngôn ngữ máy tính. Chính vì lý do này, ngôn ngữ hình thức được sử dụng phổ biến trong biểu diễn thuật toán.

Ví dụ dưới đây sẽ minh họa cho các ngôn ngữ biểu diễn thuật toán.

//Ví dụ 1.3. Biểu diễn thuật toán bằng ngôn ngữ tự nhiên *Đầu vào (Input)*. Hai số tự nhiên a, b.

Đầu ra (Output). Số nguyên u lớn nhất để a và b đều chia hết cho u.

Thuật toán (Euclide Algorithm):

Bước 1. Đưa vào hai số tự nhiên a và b.

Bước 2. Nếu b· 0 thì chuyển đến bước 3, nếu b=0 thì thực hiện bước 4.

Bước 3. Đặt r = a mod b; a = b; b = r ; Quay quay trở lại bước 2.

Bước 4 (Output). Kết luận u=a là số nguyên cần tìm.

//**Ví dụ 1.4.** Biểu diễn thuật toán bằng ngôn ngữ máy tính (C++)

```
int USCLN( int a, int  b) {   while ( b != 0 ) {//lặp trong khi b khác 0
                r = a % b; //đặt r bằng phần dư của a/b
                a = b; // đặt a bằng b
                b = r; //đặt b bằng r
        }
        return(a);//trả lại giá trị a }
```

//**Ví dụ 1.5.** Biểu diễn thuật toán bằng ngôn ngữ hình thức *Thuật toán Euclide:*

Đầu vào (Input): a∙N, a∙N.
Đầu ra (Output): s = max { u ∙N : a mod u =0 and b mod u =0}.
Format : s = Euclide (a, b).
Actions:

```
        while (b ∙ 0 )  do //lặp trong khi b khác 0
                r = a mod b; //đặt r bằng a mod b
                a = b; //đổi giá trị của a thành b
                b = r;// đổi giá trị của b thành r
        endwhile;//kết thúc cấu trúc lặp while
        return(a);//giá trị trả về của hàm
Endactions.
```

Một số lưu ý trong khi biểu diễn thuật toán bằng ngôn ngữ hình thức:

- Khi biểu diễn bằng ngôn ngữ hình thức ta được phép sử dụng cả ngôn ngữ tự nhiên hoặc ngôn ngữ máy tính thông dụng. Mỗi bước thực hiện của thuật toán không cần mô tả quá chi tiết mà chỉ cần mô tả một cách hình thức miễn là đầy đủ thông tin để chuyển đổi thành ngôn ngữ lập trình.
- Đối với những thuật toán phức tạp nặng nề về tính toán, các công thức cần được mô tả một cách tường minh, có ghi chú rõ ràng.
- Đối với các thuật toán kinh điển thì ta cần phải thuộc. Không bắt buộc phải chứng minh lại độ phức tạp của các thuật toán kinh điển.