TỪ PASCAL ĐẾN C/C++

MỤC LỤC

PHẦN 1. GIỚI THIỆU VỀ C/C++

1. Lịch sử hình thành

Ngôn ngữ lập trình C do Dennis Ritchie nghĩ ra khi ông làm việc tại AT&T Bell Laboratories vào năm 1972.

C là một ngôn ngữ mạnh và có tính linh hoạt, nó đã nhanh chóng được sử dụng một cách rộng rãi, vượt ra khỏi phạm vi của Bell Labs. Các lập trình viên ở khắp mọi nơi bắt đầu sử dụng nó để viết tất cả các loại chương trình.

C++ được biết đến như là ngôn ngữ mới bao trùm lên C và do Bjarne Stroustrup sáng tác năm 1980 cũng tại phòng thí nghiệm Bell tại bang New Jersey, Mỹ. Ban đầu được ông đặt tên cho nó là "C with classes" (C với các lớp). Tuy nhiên đến năm 1983 thì ông đổi tên thành C++, trong đó ++ là toán tử tăng thêm 1 của C.

C++ được biết đến như là ngôn ngữ lập trình hướng sự vật hay hướng đối tượng - OOP (Object Oriented Programming).

2. Đặc điểm

C là một ngôn ngữ mạnh và linh hoạt. C được sử dụng trong nhiều dự án khác nhau, như viết hệ điều hành, chương trình xử lý văn bản, đồ hoạ, bảng tính, và thậm chí cả chương trình dịch cho các ngôn ngữ khác.

C có sẵn rất nhiều các trình biên dịch (compiler) và các thư viện được viết sẵn khác.

C là một ngôn ngữ khả chuyển (portable language). Nghĩa là một chương trình viết bằng C cho một hệ máy tính (ví dụ như IBM PC) có thể được dịch và chạy trên hệ máy tính khác (chẳng hạn như DEC VAX) chỉ với rất ít các sử đổi. Tính khả chuyển đã được bởi chuẩn ANSI cho C.

C chỉ gồm một số ít từ khoá (keywords) làm nền tảng để xây dựng các các chức năng của ngôn ngữ.

C là ngôn ngữ lập trình theo modul. Mã chương trình C có thể (và nên) được viết thành các thủ tục gọi là function. Những function này có thể được sử dụng lại trong các ứng dụng (application) và chương trình khác nhau. Tuy nhiên C không cho phép khai báo hàm trong hàm.

C++ bao trùm lên C nên mọi đặc điểm của C đều có trong C++. Ngoài ra, C++ còn có một số đặc điểm khác như:

C++ là ngôn ngữ hướng đối tượng.

C++ là ngôn ngữ định kiểu rất mạnh.

C++ cung cấp cách truyền tham số bằng tham chiếu cho hàm.

C++ cung cấp cơ cấu thư viện để người lập trình có thể tự tạo thêm hàm thông dụng vào thư viện và có thể tái sử dụng sau này.

C++ cung cấp một cơ chế đa dạng hóa tên hàm và toán tử.

C++ cung cấp các class là loại cấu trúc mới đóng gói chung cho cả dữ liệu lẫn các hàm trong một chủ thể được bảo vệ một cách chặt chẽ.

Cái tên C++ nhấn mạnh sự tiến hóa tự nhiên của các thay đổi từ C. C+ là tên của một ngôn ngữ lập trình cũ và không liên quan gì đến C hay C++.

3. Một số ví dụ đơn giản

Ví dụ 1. Hiển thị câu hỏi; người dùng đưa vào thông báo trả lời và máy sẽ đọc vào biến `response`. Sau đó, khối mã của câu lệnh `if` sẽ phân nhánh quyết định hiển thị trả lời.

```
#include <iostream>
using namespace std;
int main()
 {
   int response;
   cout << "Are you feeling well? (1=Yes, 2=No)" << flush;
   cin >> response;
   if (response == 1) {
     cout << "I am glad that you are fine.";
   }
   else {
     cout << "Oh, I am so sorry.";
   }
 }
```

Ví dụ 2. Tìm ước chung lớn nhất của hai số nguyên a và b theo cách đệ quy

```
#include <iostream>
using namespace std;
int gcd(int a, int b) {
   return b == 0 ? a : gcd(b, a % b);
}
int main() {
   int num1, num2;
   cout << "Enter first number: "; cin >> num1;
   cout << "Enter second number: "; cin >> num2;
   cout << "Greatest Common Divisor: " << gcd(abs(num1), abs(num2)) <<
   endl;
}
```

Hoặc theo cách không chính quy

```
#include <iostream>
using namespace std;
int main() {
   int num1, num2;
   cout << "Enter first number: "; cin >> num1;
   cout << "Enter second number: "; cin >> num2;
   cout << "Greatest Common Divisor: " << __gcd(abs(num1), abs(num2)) <<
```

```
    endl;
}
```

PHẦN 2. CÀI ĐẶT IDE HỖ TRỢ C/C++

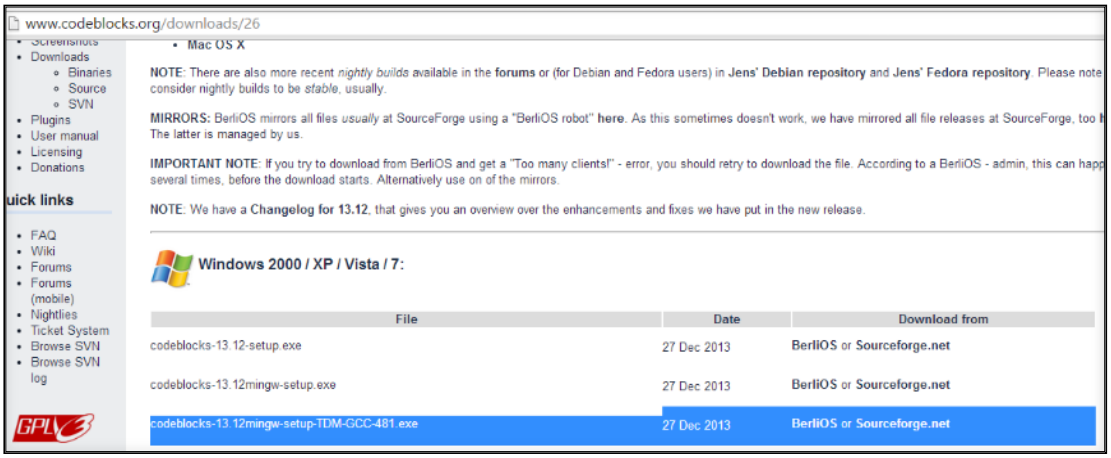
IDE phổ biến thông dụng hiện nay dành cho C/C++ trên môi trường thông dụng Windows là CodeBlocks. Cá nhân tôi thích học sinh trong đội tuyển sử dụng trên môi trường Linux, Ubuntu chẳng hạn.

Nhiều lập trình viên có thể dùng IDE khác với nhiều mục đích và hỗ trợ nhiều nguôn ngữ, chẳng hạn: NetBeans IDE hỗ trợ khá nhiều: Java, HTML5, PHP, C/C++ và rất nhiều. Tất nhiên bạn vẫn có thể dùng Notepad: code mọi thứ. (^_^). Tôi không khuyến khích sử dụng Turbo C hoặc Dev C++ vì nhiều lý do.

Dưới đây là phần cài đặt CodeBlocks, tính đến nay đã là phiên bản 13.12

Bước 1. Download

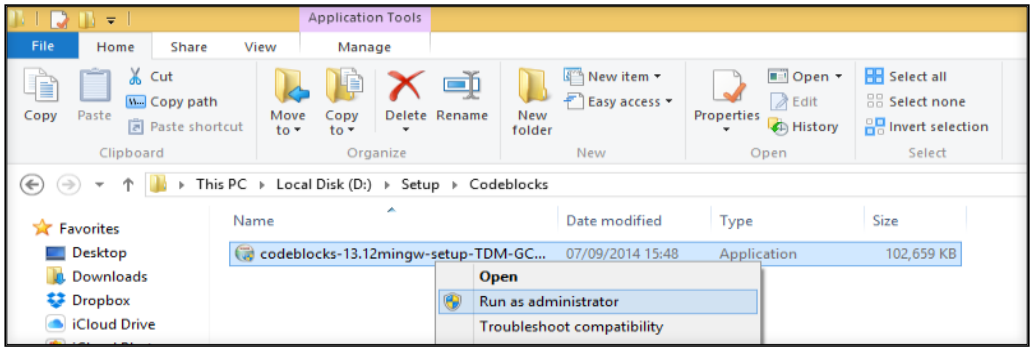
- Truy cập vào địa chỉ: http://www.codeblocks.org/downloads/26
- Tải về tệp tin: codeblocks-13.12mingw-setup-TDM-GCC-481.exe tại host BerliOS hoặc của host Sourceforge.net



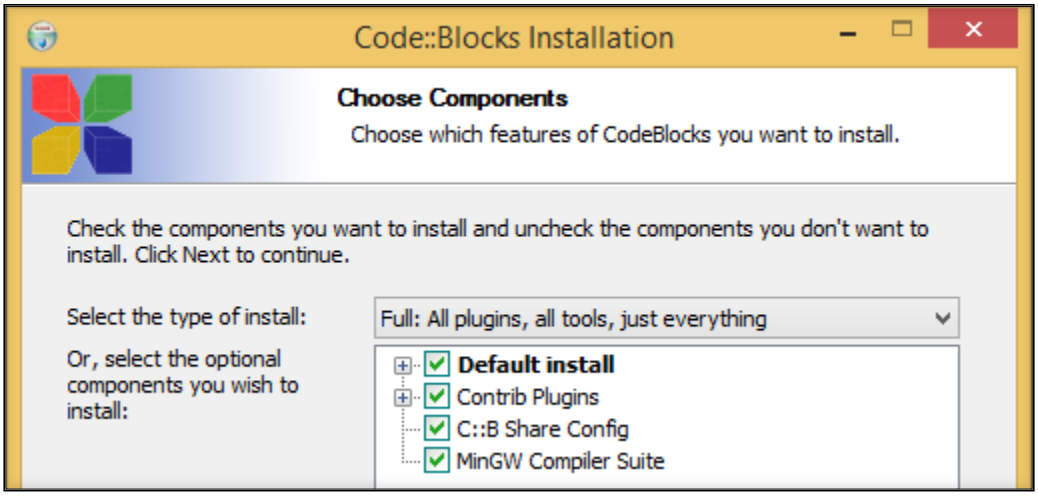
Hình 1: Tải CodeBlocks IDE

Bước 2. Cài đặt

- Cài đặt thông thường như một phần mềm trên môi trường windows, chú ý là với windows Vista trở lên, quý thầy cô cần cài đặt với quyền hạn của Administrator



Hình 2: Cài đặt CodeBlocks với quyền quản trị trên windows 8.1 64 bit

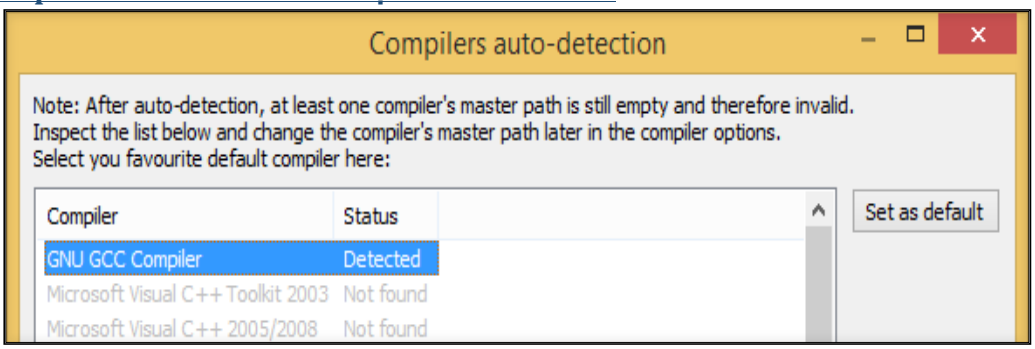


Hình 3: Kiểu cài đặt: Full: All plugins

Bước 3. Khởi động IDE

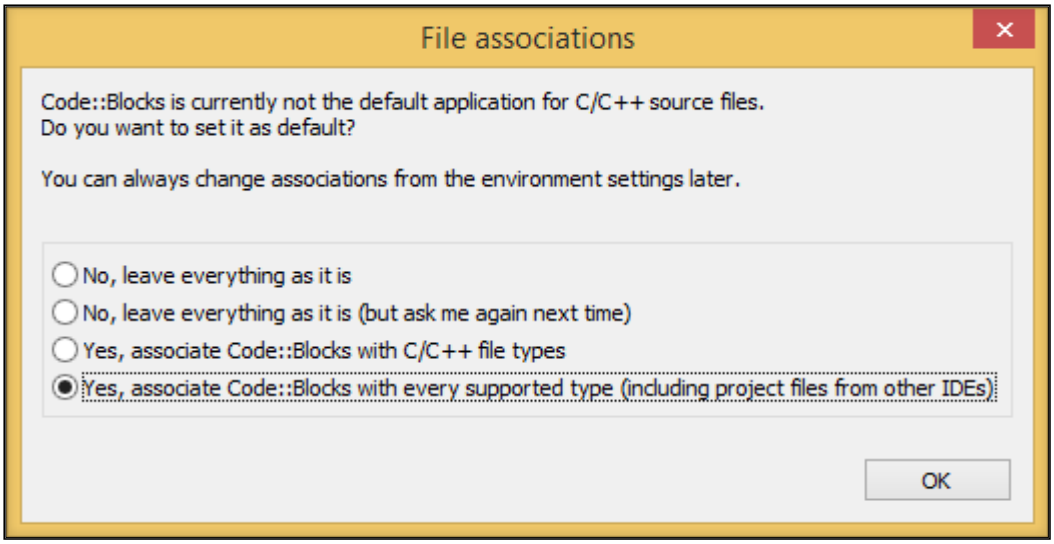
Hình 4: Khởi động CodeBlocks từ màn hình Desktop

Bước 4. Compilers là GNU GCC và chọn Set As default



Hình 5: Chọn Compilers là GNU GCC và chọn Set As default

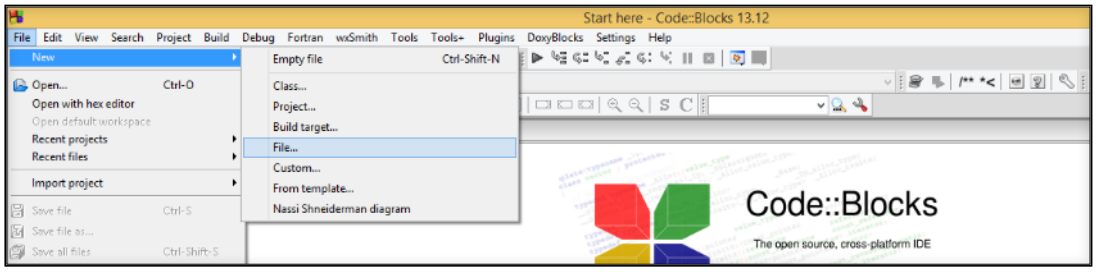
Bước 5. Chọn every supported type



Hình 6: Lựa chọn thứ 4 Every Supported Type

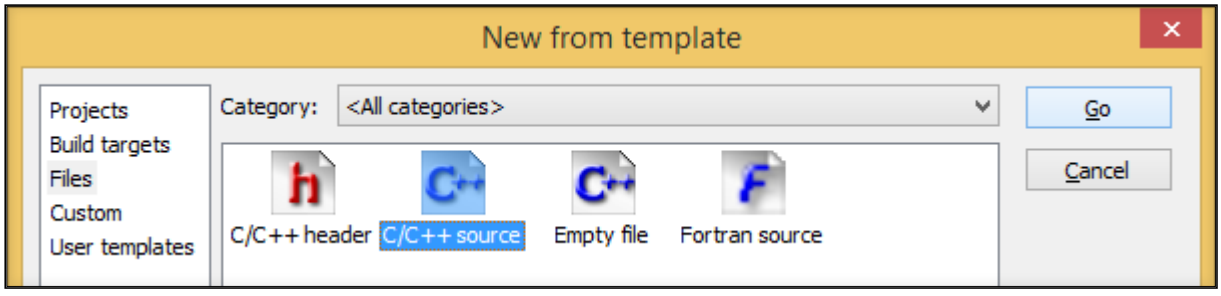
Bước 6. Viết chương trình đầu tiên

- Chọn File -> New -> File . . .



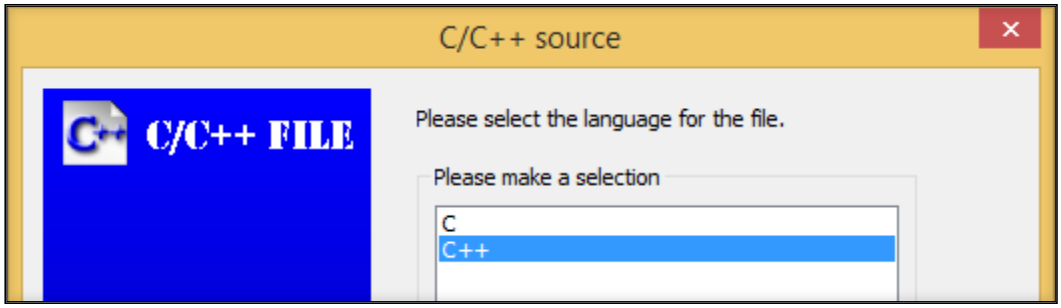
Hình 7: Tạo file mới để viết chương trình

- Chọn C/C++ -> Go



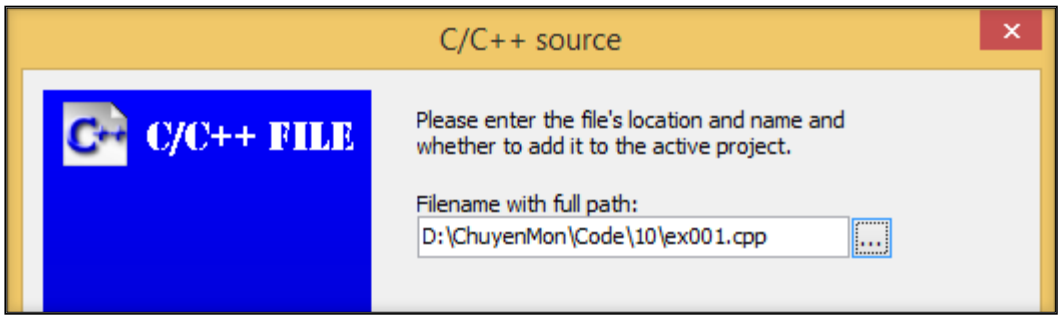
Hình 8: Chọn kiểu C/C++ source

- Chọn tiếp C++ để viết mã nguồn



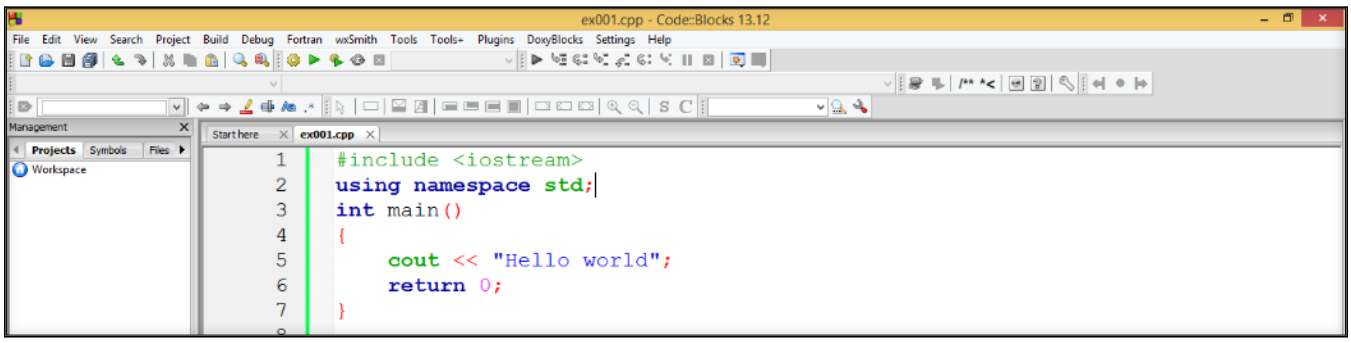
Hình 9: Chọn C++

- Đặt đường dẫn cho file:



Hình 10: Đặt đường dẫn để lưu file nguồn

- Viết mã nguồn:



Hình 11: Mã nguồn Hello World

Chạy chương trình: Bấm phím F9

Hình 12: Cửa sổ chạy chương trình

PHẦN 3. MỘT SỐ CÚ PHÁP GIỮA C/C++ VÀ PASCAL

1. QUY ƯỚC VỀ TỪ VỰNG

Pascal	C/C++
- Không phân biệt chữ in hoa, chữ in thường; - *somename, SOMENAME, Somename, SomeName* là các từ giống nhau.	- Phân biệt chữ in hoa, chữ in thường; - *somename, SOMENAME, Somename, SomeName* là các từ khác nhau.

2. CHÚ THÍCH

Pascal	C/C++

`(* Đây là chú thích *)`	`/* Đây là chú thích */`
	Hoặc theo kiểu C++ chú thích 1 dòng:
	`// Đây là chú thích 1`
	`// Đây là chú thích 2`

3. HẰNG VÀ KHAI BÁO

Các quy tắc cho hằng số (số nguyên, số thực) về cơ bản là giống nhau mặc dù trong C có linh hoạt hơn.

Pascal	C/C++

`'c'`
`'This is a string'`
`(* Khai báo *)`
`const`
`size = 100;`
`pi = 3.14159;`
`pi = 3.14159;`
`first = 'A';`
`filename = 'XYZ.DAT';`
	`'c'`
	`"This is a string"`
	`// Khai báo`
	
	`#define size 100`
	`#define pi 3.14159`
	`#define first 'A'`
	`#define filename "XYZ.DAT"`
	
	`// Hoặc theo Ansi C và C++`
	`const int size = 100;`
	`const float pi = 3.14159;`
	`const char first = 'A';`
	`const char filename[] = "XYZ.DAT";`

4. KHAI BÁO

Khai báo biến thông thường

Pascal	C/C++

`var`
`i: integer;`
`r: real;`
`b: boolean;`
	
	`int i;`
	`float r;`
	`int b;`

c: char; j, k, l: integer;	char c; int j, k, l;

Mở rộng:

- Với biến có giá trị TRUE hoặc FALSE trong Pascal, có thể được khai báo trong C++ mới theo cách như sau: `bool b;`
- C không sử dụng một từ giống như là `var` để khai báo biến.
- C không sử dụng một loại từ riêng biệt dành cho các giá trị boolean. Nó sử dụng int với giá trị 0 tương đương False và 1 tương đương True trong Pascal.
- Với hai kiểu nguyên là `char` và `int` có thể có dấu âm hoặc không có. Do vậy nếu dùng dưới dạng kiểu nguyên chỉ chấp nhận các số không âm thì phải dùng từ khóa `unsigned`. Ví dụ: `unsigned int i;` và với kiểu `char` thì là `unsigned char c;`

Khai báo mảng:

Pascal	C/C++
var a: array[0..9] of integer; b: array[0..4, 0..9] of real; c: array[1..10] of integer;	 int a[10]; float b[5][10]; int c[10];

Mở rộng:

- Mảng trong C/C++ luôn bắt đầu từ 0 tới kích thước khai báo -1

Bản ghi

Pascal	C/C++
var student: record id: integer; name: packed array[1..10] of char; gpa: real end;	struct { int id; char name[11]; float gpa; } student;

Biến nhận giá trị xác định từ trước

Pascal	C/C++
var flavor: (chocolate, vanilla);	 enum {chocolate, vanilla} flavor;

Đoạn con

Pascal	C/C++

var
squares: set of 1..9;
 	
squares := [];
...
squares := squares + [2]
	- Không có kiểu dữ liệu đoạn con, nhưng có thể tìm giải pháp thay thế như dưới đây:
	int squares;
	...
	squares = 0;
	...
	squares |= 1 << 2;

Mảng hằng ký tự

Pascal	C/C++
const vowels = ['A','a','E','e','I','O']; ... if c in vowels then ...	 #define VOWELS "AaEeIO" ... if (strchr(VOWELS, c))

Mở rộng:

- Một biến trong C/C++ có thể được nhận giá trị khởi tạo trong khi khai báo như ví dụ dưới đây:

```
int i = 3;
char c = 'A';
float a[3] = { 1.0, 2.0, 3.0 };
struct
  {
     char name[11];
     float gpa;
  } student = { "AARDVARK", 4.0 };
```

5. KHAI BÁO KIỂU DỮ LIỆU NGƯỜI DÙNG

Pascal	C/C++

type
    realarray = array[0..9] of real;
    fla = (choco, vanilla);
var
    a: realarray;
    flavor: fla;
	
	typedef float realarray[10];
	enum fla {choco, vanilla};
	
	realarray a;
	enum fla flavor;

6. CON TRỎ

Pascal	C/C++

var
   p: ^integer;
   first: ^student;
	
	int *p;
	student *first;
	Hoặc
	struct student *first;

type
    stuptr = ^node;
var
    first: stuptr;
	
	typedef student *stuptr;
	
	stuptr first;

type
    nodeptr = ^node;
    node = record
        info: integer;
        link: nodeptr
	typedef struct node * nodeptr;
	typedef
	    struct node
	    {
	        int info;

end; var first: nodeptr;	nodeptr link; } node; nodeptr first;
Với khai báo như trên thì có thể sử dụng như ví dụ sau:	

p^ := p^ + 1;
new(first);
...
first := first^.link;
...
dispose(first);
	*p = *p + 1;
	first = (nodeptr) malloc(sizeof(node));
	...
	first = first -> link;
	...
	free(first);
	
	// hoặc ngắn gọn hơn với C++
	
	first = new node;
	...
	delete first;

7. HÀM VÀ THỦ TỤC

Pascal	C/C++

function f(x, y: integer; z: real): real;
 	
var
q: integer;
begin
q := sqr(x) + y;
f := q – z
end;
	float f(int x, int y, float z)
	
	{
	int q;
	
	q = x*x + y;
	return q - z;
	}

procedure p(x: integer);
var
temp: integer;
begin
...
end;
	void p(int x)
	{
	int temp;
	...
	return 0;
	}

Giả sử, có:
function f: integer;
...
procedure p;
...
Thì có thể sử dụng theo cách sau:
x := f;
p;
	Giả sử, có:
	int f()
	...
	void p()
	...
	Thì có thể sử dụng theo cách sau:
	x = f();
	p();

procedure p(var x: integer); begin	void p(int &x) {

x := 17; end;	x = 17; }

type
realarray = array[0..9] of real;
procedure p(var a: realarray);
begin
a[1] := a[2] + a[3]
end;
	
	void p(float a[])
	{
	a[1] = a[2] + a[3];
	}

program ...
...
begin
...
end;
	
	main()
	{
	...
	}

8. PHÉP TOÁN VÀ BIỂU THỨC

Pascal	C/C++
x * y x / y i div j i mod j	x * y x / y i / j i % j
x + y x - y	x + y x – y
x < y x > y x <= y x >= y	x < y x > y x <= y x >= y
x = y x <> y	x == y x != y
b and c b or c	b && c b || c
x := y x := x + y x := x – y x := x * y i := i div j x := x / y i := i mod j	x = y x += y x -= y x *= y i /= j x /= y i %= j

Mở rộng:

Một số phép toán chỉ được dành cho C++

Pascal	C/C++
new(p) dispose(p)	p = new t delete p

9. THỰC THI LỆNH

Pascal	C/C++
x := y + z	x = y + z;

begin
  x := y + z;
  w := x
end
	{
	  x = y + z;
	  w = x;
	}

10. RẼ NHÁNH VÀ LẶP

Pascal	C/C++

if x < 0 then
  x := -x
if x > y then
  max := x
else
  max := y
	if (x < 0)
	  x = -x;
	if (x > y)
	  max = x;
	else
	  max = y;

while x < y do
  x := 2 * x
	while (x < y)
	  x = 2 * x;

repeat
  x := 2 * x;
  y := y – 1
until x >= y
	do
	  {
	  x = 2 * x;
	  y--;
	  }
	while (x<y);

for i := 1 to n do
  x[i] := 0
	for (i = 1 ; i <= n ; i++)
	  x[i] = 0;

case i of
 	
  1: write('one');
  2: write('two');
  3: write('three');
  4: begin
  write('four');
  i := 3
  end
 	
otherwise
  write('Bad value')
 	
end;
	switch (i)
	{
	  case 1: printf("one");
	  break;
	  case 2: printf("two");
	  break;
	  case 3: printf("three");
	  break;
	  case 4: printf("four");
	  i = 3;
	  break;
	  default: printf("Bad value");
	}

11. VÀO – RA

Pascal	C

var
  c: char;
  i: integer;
  r: real;
  s: packed array[1..10] of char;
...
read(c);
read(i);
read(r);
readln;
readln(c, i, r);
	#include <stdio.h>
	
	char c;
	int i;
	float r;
	char s[10];
	...
	scanf("%c", & c);
	scanf("%d", & i);
	scanf("%f", & r);
	while (getchar() != '\n') ;
	scanf("%c%d%f", & c, & i, &r);

write(c);
write(i);
write(r);
write(s);
writeln;
	printf("%c", c);
	printf("%d", i);
	printf("%f", r);
	printf("%s", s);
	printf("\n");

writeln('c=',c,' i=',i,' r=', r, s);	printf("c=%c i=%d r=%f%s\n",c, i, r, s);

Mở rộng

Có thể dùng theo kiểu của C++ như sau:

Pascal	C++

read(c);
read(i);
read(r);
readln;
readln(c, i, r);
	#include <iostream.h>
	cin >> c;
	cin >> i;
	cin >> r;
	while (cin.get() != '\n') ;
	cin >> c, i, r;

write(c);
write(i);
write(r);
write(s);
writeln;
	cout << c;
	cout << i;
	cout << r;
	cout << s;
	cout << endl;

writeln('c=',c,' i=',i,' r=', r, s);	cout << "c=" << c << " i=" << i << " r=" << r << s << endl;

Vào – ra đối với tệp

Pascal	C++
var fi, fo: text; ... readln(fi, c, i);	#include <fstream.h> ifstream fi; ofstream fo; ... fi >> c >> i >> r;

write(fo,'c=',c,' i=',i);	while (fi.get() != '\n') ; fo << "c=" << c << " i=" << i;

12. TIỀN XỬ LÝ CỦA C/C++

Trong khoa học máy tính, tiền xử lý là một chương trình xử lý các dữ liệu đầu vào thành các đầu ra. Các đầu ra này tiếp tục được sử dụng là đầu vào của một chương trình khác. Các đầu ra được coi là dạng tiền xử lý của dữ liệu đầu vào, thường được sử dụng bởi các chương trình tiếp theo như các trình biên dịch

Trong Pascal, tiền xử lý thông thường ở mức độ khá đơn giản, nhưng với C/C++ thì rất linh hoạt.

Pascal	C++
const size = 10;	#define size 10

Mở rộng:

```
#define iszero(e)       (e == 0)
#define equal(x, y)     (x == y)
#define error(f, m)     if (f) printf(m)
#define VAX
```

Trong khuôn khổ một chuyên đề thì không thể trình bày hết tất cả mọi thứ về tiền xử lý trong C/C++, nhưng học sinh trong giai đoạn chuyển đổi từ Pascal sang C/C++ chỉ cần nắm được chỉ thị `#define`, còn những thứ khác xem như là tự học để nâng cao kiến thức.

PHẦN 4. STL

STL là viết tắt của Standard Template Library, một thư viện template cho C++ với những cấu trúc dữ liệu cũng như giải thuật được xây dựng tổng quát mà vẫn tận dụng được hiệu năng và tốc độ của C.

Một số khả năng của STL:

- Quản lí mảng với vector
- Xây dựng sẵn các cấu trúc dữ liệu cơ bản (stack, queue, map, set…),
- Cung cấp các thuật toán cơ bản: tìm min, max, tính tổng, sắp xếp (với nhiều thuật toán khác nhau), thay thế các phần tử, tìm kiếm (tìm kiếm thường và tìm kiếm nhị phân), trộn.

Chúng ta sẽ tìm hiểu về một trong số nhiều STL thông dụng trong C++ để nâng cao hiệu năng so với Pascal đó là vector. Tôi sẽ tiếp tục cập nhật các STL khác sau khi trại hè 2015 kết thúc.

STL Vector

4.1. Giới thiệu về vector:

Vector có thể xem như là một mảng động (dynamic array): một mảng có thể thay đổi kích thước.

Giống như mảng, vector sử dụng các vùng nhớ liên tiếp để lưu trữ các phần tử, đều đó có nghĩa là các phần tử có thể sử dụng con trỏ để truy cập các phần tử. Tuy nhiên, không giống như mảng thông thước, kích thước của vector có thể thay đổi một cách linh hoạt.

4.2. Khai báo thư viện vector ở đầu chương trình:

```
#include <vector>
```

4.3. Khai báo vector 1 chiều

Ví dụ 1: Khai báo vector 1 chiều, rỗng, dữ liệu kiểu `int` có tên là `firstVector`:

```
vector <int> firstVector;
```

Ví dụ 2: Tạo vector có 4 phần tử `int` và 4 phần tử này đều có giá trị 100 có tên là `secondVector`:

```
vector <int> secondVector (4,100);
```

Ví dụ 3: Khai báo một vector kiểu int có tên là `thirdVector` sao chép từ đầu đến cuối `secondVector`:

```
vector <int> thirdVector (secondVector.begin(),secondVector.end()) ;

/*
Lưu ý: begin(), end() là các phương thức cung cấp sẵn của đối tượng vector
dùng để truy xuất phần tử đấu tiên và cuối cùng của vector.

*/
```

Ví dụ 4: Tạo vector kiểu `int` tên là `fourVector` và sao chép tất cả phần tử `thirdVector`

```
vector <int> fourList (thirdList) ;
```

4.4. Khai báo vector 2 chiều

Ví dụ 5: Tạo vector 2 chiều, phần tử rỗng, kiểu int có tên là v:

```
vector < vector <int> > v;
```

Ví dụ 6: khai báo vector kích thước 5×10 :

```
vector < vector <int> > v (5, 10) ;
```

Ví dụ 7: Khai báo vector 2 chiều, có 5 vector 1 chiều đều rỗng:

```
vector < vector <int> > v (5) ;
```

4.5. Các phương thức thành viên

Đầu tiên ta cần hiểu rằng: vector là một đối tượng (Object) được xây dựng sẵn. Do đó, các hàm thành viên của vector tất nhiên được gọi là phương thức (method). Vậy, để truy xuất các phương thức của vector, ta dùng cấu trúc:

```
Object.Method(parameter, ...)
```

Ví dụ: Sử dụng phương thức `push_back(paramether)` của vector để thêm 10 vào cuối vector:

```
#include <vector>

int main() {
            vector<int> myVector ;
            myVector.push_back(10) ;
}
```

Một số phương thức và ý nghĩa

Phương thức	Ý nghĩa
`size()`	Trả về số lượng phần tử
`empty()`	Trả về true(1) nếu vector rỗng, ngược lại là false (0)
`operator [int i]`	Truy cập giá trị phần tử thứ i của vector, tương tự như đối mảng thông thường
`at(int i)`	Truy cập phần tử thứ i của vector
`front()`	Truy cập phần tử đầu tiên của vetor
`back()`	Truy cập phần tử cuối cùng của vector
`push_back(const x)`	Thêm phần tử có giá trị x vào cuối vector
`pop_back()`	Loại bỏ phần tử cuối ra khỏi vector
`insert (iterator positon,const x)` `insert (iterator positon,int n, const x)`	Chèn phần tử có giá trị x vào trước vị trí position.Chèn n phần tử có giá trị x vào trước vị trí position.
`insert (iterator positon,iterator a, itertator b)`	Chèn vào trước vị trí position tất cả các phần tử trong nửa khoảng [a,b) của một vector khác.
`erase (iterator position)`	Xóa phần tử ở vị trí position
`erase (iterator first, iterator last)`	Xóa tất cả các phần tử trong nửa khoảng

	[first,last), tức là từ phần tử thứ first đến phần tử thứ (last-1)
`swap(vector v)`	Hoán đổi các phần tử của vector hiện hành và vector v
`clear()`	Xóa vector

4.6. Một số ví dụ áp dụng.

Ví dụ 1: Tạo một vector có kích thước là 10 và in ra số lượng phần tử của vector này.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector (10)  ;
    cout << "The size of vector is: " << myVector.size()<< endl;
    return 0;
}
```

Ví dụ 2: Ta có thể dùng phương thức `size()` để truy xuất các phần tử của vector khi chưa biết kích thước của vector:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector;
    // create a vector
    for(int i = 0 ; i<10; i++) {
        myVector.push_back(i) ;
    }
    // print vector
    for(int i = 0 ; i<myVector.size() ; i++) {
       cout<<myVector[i]<<" "  ;
    }
    return 0;
}
```

Ví dụ 3: Kiểm tra vector có rỗng không và in ra thông báo tương ứng.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector ;
   if( myVector.empty()) {
        cout<<"Vector is empty! "<<endl ;
   }
```

```
   else {
        cout<<"Vector is not empty! "<<endl ;
   }
    return 0;
}
```

Ví dụ 4: Tạo một vector 10 phần tử với giá các phần tử lần lượt là từ 0 đến 9. In ra các phần tử của vector:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector ;
    for(int i = 0 ; i<10 ; i++) {
        myVector.push_back(i) ;
    }
    for(int i = 0 ; i<10 ; i++) {
       cout<<myVector[i] << " "  ;
    }
     return 0;
}
```

Ví dụ 5: Tạo vector có 10 phần tử bao gồm 0, 10, 20, 30 , 40, 50, 60, 70, 80, 90. Sau đó, sử dụng phương thức at() để in giá trị các phần tử.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector;
    // create a vector
    for(int i = 0 ; i<10; i++) {
        myVector.push_back(i*10) ; // 0 10 20 30 40 50 60 70 80 90
    }
    // print vector
    for(int i = 0 ; i<myVector.size() ; i++) {
       cout << myVector.at(i) << " "  ;
    }
    return 0;
}
```

Ví dụ 6: Tạo vector có 5 phần tử có giá trị lần lượt là a,b,c,d,e. Thay đổi phần tử đầu tiền thành ‘A’ và in ra kết quả.

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    vector<char> myVector;
    // create a vector
    for(int i = 0 ; i<5; i++) {
        myVector.push_back('a'+ i ) ; // a b c d e
    }
    // change vector
    cout<<"Before change, front is: "<<myVector.front()<<endl ;
    myVector.front() = 'A' ;
     cout << "After change, front is: "<<myVector.front()<<endl ;
    return 0;
}
```

Ví dụ 7: Viết chương trình cho người dùng nhập vào kích thước vector và nhập vào giá trị các phần tử. In ra các phần tử trong vector.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector;
    int n; // size of vector
    cout << "Size: " ;
    cin >> n ;
    // create vector
    int tempNumber  ;
    for(int i = 0 ; i<n; i++) {
        cin >> tempNumber ;
        myVector.push_back(tempNumber ) ;
    }
    // print vector
    for(int i = 0 ; i<n; i++) {
       cout<<myVector[i]<<" " ;
    }

    return 0;
}
```

Ví dụ 8: Viết chương trình cho người dùng nhập vào kích thước vector và nhập vào giá trị các phần tử. Sau đó, loại bỏ đi một nửa các phần tử ở cuối vector. In ra các phần tử trong vector.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector;
    int n; // size of vector
```

```
    cout<<"Size: " ;
    cin>>n ;
    // create vector
    int tempNumber  ;
    for(int i = 0 ; i<n; i++) {
        cin>>tempNumber ;
        myVector.push_back(tempNumber ) ;
    }
    // delete element
     for(int i = 0 ; i<n/2; i++) {
        myVector.pop_back() ;
    }
    // print vector
    for(int i = 0 ; i<myVector.size(); i++) {
       cout<<myVector[i]<<" " ;
    }

    return 0;
}
```

Ví dụ 9: Tạo vector có 5 phần tử đều có giá trị là 100. Chèn phần tử có giá trị 200 vào vị trí thứ 2 của vector.

```
// inserting into a vector
#include <iostream>
#include <vector>
using namespace std ;
int main ()
{
    // create vector
  vector<int> myVector (5,100); // 100 100 100 100 100
    // create iterator
  vector<int>::iterator it;
  it = myVector.begin();
  // insert 200 into 2rd position of vector
  myVector.insert(it+1,200) ; // 100 200 100 100 100 100
  // print vector
  for(int i = 0 ; i < myVector.size() ; i++) {
    cout<<myVector[i]<<" " ;
  }
  return 0;
}
```

Ví dụ 10: Tạo một vector có 10 phần tử có giá trị từ 1 đến 10. Xóa phần tử thứ 6 và in ra các phần tử còn lại.

```
#include <iostream>
#include <vector>
using namespace std ;
int main () {
```

```
  vector<int> myvector;

  // set some values (from 1 to 10)
  for (int i=0; i<10; i++) {
      myvector.push_back(i+1); // 1 2 3 4 5 6 7 8 9 10
  }
  // erase the 6th element
  myvector.erase (myvector.begin()+5);
 // print vector
  for (int i=0; i<myvector.size(); ++i) {
        cout <<myvector[i]<<" " ;
  }
  return 0;
}
```

PHẦN 5. MỘT SỐ VÍ DỤ CÀI ĐẶT ĐỒ THỊ BẰNG C++

1. Tìm kiếm theo chiều rộng - BFS

```
#include <iostream>
#include <queue>
#include <vector>
const int MAX_VERTEX = 100;
bool seen[MAX_VERTEX];
std::vector<std::vector<int>> adj(MAX_VERTEX);
void visit(int u) {
    std::cout << u << "\n";
}
void bfs(int s) {
    std::queue<int> q;
    q.push(s);
    seen[s] = true;
    while (!q.empty()) {
        int u = q.front();
        visit(u);
        q.pop();
        for (auto v : adj[u]) {
            if (!seen[v]) {
                seen[v] = true;
                q.push(v);
            }
        }
    }
}
/**
 * For undirected graph
 */
void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}
int main() {
    add_edge(0, 1);
    add_edge(0, 2);
    add_edge(0, 3);
    add_edge(1, 4);
    add_edge(1, 5);
    add_edge(2, 3);
    bfs(0);
    return 0;
}
```

Ứng dụng: Tìm đường đi ngắn nhất cho từng cặp đỉnh

```
#include <iostream>
#include <queue>
#include <vector>
#include <limits>
const int MAX_VERTEX = 100;
const int oo = std::numeric_limits<int>::max();
bool seen[MAX_VERTEX];
std::vector<std::vector<int>> adj(MAX_VERTEX);
std::vector<int> distance(MAX_VERTEX, oo);

void bfs(int s) {
    std::queue<int> q;
    q.push(s);
    seen[s] = true;
    distance[s] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto v : adj[u]) {
            if (!seen[v]) {
                // recompute distance to v
                distance[v] = distance[u] + 1;
                seen[v] = true;
                q.push(v);
            }
        }
    }
}
/**
 * For undirected graph
 */
void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}
void shortest_distance() {
    add_edge(0, 1);
    add_edge(0, 2);
    add_edge(0, 3);
    add_edge(1, 4);
    add_edge(4, 5);
    bfs(0);
    for (int v = 0; v < 6; ++v) {
```

```
        std::cout << "shortest distance from 0->" << v << ":
" << distance[v] << std::endl;
    }
}
int main() {
    shortest_distance();
    return 0;
}
```

2. Tìm kiếm theo chiều sâu:

Thuật toán kế tiếp là thuật toán tìm kiếm theo chiều sâu (DFS), nó cũng là một thuật toán dùng để ghé thăm các đỉnh của đồ thị. Điều khác biệt rõ ràng nhất với BFS là DFS hoạt động dựa trên sự vận hành của stack (FILO) thay vì queue (FIFO). Vì bản chất của DFS là dùng lời gọi đệ qui để gọi hàm, cho nên tôi sẽ implement DFS dùng đệ qui thay vì dùng stack.

```
#include <iostream>
#include <vector>
const int MAX_VERTEX = 100;
bool seen[MAX_VERTEX];
std::vector<std::vector<int>> adj(MAX_VERTEX);

void visit(int u) {
    std::cout << u << "\n";
}

void dfs(int u) {
    visit(u);
    seen[u] = true;
    for (auto v : adj[u]) {
        if (!seen[v]) {
            dfs(v);
        }
    }
}

/**
 * For undirected graph
 */
void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}

int main() {
    add_edge(0, 1);
```

```
    add_edge(0, 2);
    add_edge(0, 3);
    add_edge(1, 4);
    add_edge(1, 5);
    add_edge(2, 3);
    dfs(0);
    return 0;
}
```

Ứng dụng 1: Tìm thành phần liên thông của đồ thị (connected components)

```
#include <iostream>
#include <vector>

const int MAX_VERTEX = 100;
bool seen[MAX_VERTEX];
std::vector<std::vector<int>> adj(MAX_VERTEX);

void dfs(int u) {
    seen[u] = true;
    for (auto v : adj[u]) {
        if (!seen[v]) {
            dfs(v);
        }
    }
}

int count_connected_components(int no_vertex) {
    int cnt = 0;
    // for all vertex that has not been visited
    for (int v = 0; v < no_vertex; ++v) {
        if (!seen[v]) {
            // increase the number of components
            cnt++;
            // visit
            dfs(v);
        }
    }
    return cnt;
}

void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

```
int main() {
    add_edge(0, 1);
    add_edge(0, 2);
    add_edge(3, 4);
    add_edge(4, 5);
    std::cout << count_connected_components(6) << std::endl;
    return 0;
}
```

Ứng dụng 2: sắp xếp Topo

```
#include <iostream>
#include <vector>

const int MAX_VERTEX = 100;
bool seen[MAX_VERTEX];
std::vector<std::vector<int>> adj(MAX_VERTEX);
std::vector<int> topo_order;

void dfs(int u) {
    seen[u] = true;
    for (auto v : adj[u]) {
        if (!seen[v]) {
            dfs(v);
        }
    }
    // keep track the order of visited
    topo_order.push_back(u);
}

void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}

void toposort(int no_vertex) {
    for (int v = 0; v < no_vertex; ++v) {
        if (!seen[v]) {
            dfs(v);
        }
    }
    // print out topo order
    for (auto v : topo_order) {
        std::cout << v << " ";
    }
```

```
}

int main() {
    add_edge(0, 1);
    add_edge(0, 2);
    add_edge(3, 4);
    add_edge(4, 5);
    toposort(6);
    return 0;
}
```

3. Roy Warshall's Algorithm - Shortest Path Problem

Thuật toán tiếp theo có tên là Floyd Warshall dùng để tìm đường với chi phi thấp nhất từ 2 cặp đỉnh (u, v). Running time cho giải thuật này là $O(n^3)$ khá chậm so với Dijsktra, tuy nhiên nếu chúng ta cần query nhiều lần với những cặp đỉnh khác nhau thì việc tính toán trước cho toàn bộ các cặp đỉnh thì thuật toán này sẽ nhanh hơn Dijkstra rất nhiều.

Giả sử chúng ta đồ thị với V đỉnh, E cạnh, và M query thì:

- Floyd Warshall sẽ có running time $O(V^3) + M$
- Dijsktra Heap: $M * O(E * \log(V))$

```
#include <iostream>
#include <vector>
#include <limits>
using namespace std;
const int oo = std::numeric_limits<int>::max();
vector<std::vector<int>> cost;

void floyd_warshall(int no_vertex) {
    for (int bridge = 0; bridge < no_vertex; ++bridge) {
        for (int u = 0; u < no_vertex; ++u) {
            for (int v = 0; v < no_vertex; ++v) {
                cost[u][v] = std::min(cost[u][v], cost[u][bridge] + cost[bridge][v]);
            }
        }
    }
}

vector<vector<int>> generate_sample_cost() {
    return std::vector<std::vector<int>> {
       vector<int> {0, 2, 1, 3},
       vector<int> {1, 0, 4, 5},
       vector<int> {3, 1, 0, 3},
       vector<int> {1, 1, 1, 0},
    };
```

```
}

void query(int u, int v) {
    cout << "shortest path from " << u << "->" << v << ": " << cost[u][v] << "\n";
}

int main() {
    cost = generate_sample_cost();
    floyd_warshall(cost.size());
    query(0, 2);
    query(1, 2);
    return 0;
}
```

4. Prim's Algorithm - Minimum Spanning Tree

Thuật toán tiếp theo là thuật toán Prim, dùng để giải bài toán tìm cây khung nhỏ nhất trong đồ thị vô hướng. Thuật toán Prim là một dạng của thuật toán "tham lam" (greedy) vì nó luôn chọn cạnh với chi phí thấp nhất nhờ vào sự trợ giúp của min-heap.

```
#include <algorithm>
#include <iostream>
#include <queue>
#include <vector>
#include <utility>

struct edge {
    int v;
    int cost;

    edge(int v, int cost)
        :v(v), cost(cost) {
    }

    // sort by cost first then vertex
    bool operator >(const edge &e) const {
        return (cost != e.cost ? cost > e.cost : v > e.v);
    }
};

// alias for min priority queue
typedef std::priority_queue<edge, std::vector<edge>, std::greater<edge>> min_heap;
// infinity
const int oo = std::numeric_limits<int>::max();
// maximum # of vertex
const int MAX_VERTEX = 100;
// mark array
bool seen[MAX_VERTEX];
// adjacency list
std::vector<std::vector<int>> adj(MAX_VERTEX);
// cost matrix, initially all cost are infinity
std::vector<std::vector<int>> cost(MAX_VERTEX, std::vector<int>(MAX_VERTEX, oo));
// minimum cost heap
```

```
min_heap pq;

/**
 * For all vertex that are adjacent to u,
 * if it's not visited, we add them to
 * our queue with its cost
 */
void relax(int u) {
    seen[u] = true;
    for (auto v : adj[u]) {
        if (!seen[v]) {
            pq.push(edge(v, cost[u][v]));
        }
    }
}

/**
 * Prim algorithm for finding the minimum
 * spanning tree of undirected graph
 */
int prim(int s) {
    int min_cost = 0;
    relax(s);
    while (!pq.empty()) {
        // get the minimum cost edge
        edge e = pq.top();
        pq.pop();
        // if we have seen v
        // visit v, and update new cost
        if (!seen[e.v]) {
            min_cost += e.cost;
            relax(e.v);
        }
    }
    return min_cost;
}

void add_edge(int u, int v, int c) {
    adj[u].push_back(v);
    adj[v].push_back(u);
    cost[u][v] = c;
    cost[v][u] = c;
}

void test_minimum_spanning_tree() {
    add_edge(0, 1, 3);
    add_edge(0, 2, 3);
    add_edge(0, 3, 1);
    add_edge(0, 4, 2);
    add_edge(1, 2, 2);
    add_edge(2, 3, 11);
    add_edge(3, 4, 9);
    std::cout << "min cost: " << prim(0) << std::endl;
}
```

```
int main() {
    test_minimum_spanning_tree();
    return 0;
}
```

5. Kruskal's Algorithm - Minimum Spanning Tree

Thuật toán này cũng là một dạng thuật toán tham lam dùng để giải bài toán cây khung nhỏ nhất (giống như Prim). Điểm khác biệt giữa Kruskal và Prim là Kruskal hoạt động dựa trên cấu trúc dữ liệu có tên là "Disjoint Set".

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <utility>

const int MAX_SIZE = 100;

class disjoint_set {
private:
    int components;
    int id[MAX_SIZE];
    int sizeof_ids[MAX_SIZE];
    int n;

public:
    disjoint_set(int n)
        :components(n), n(n) {
        for (int i = 0; i < components; ++i) {
            id[i] = i;
            sizeof_ids[i] = 1;
        }
    }

    void join(int p, int q) {
        int i = find(p);
        int j = find(q);
        if (i == j) {
            return;
        }
        if (sizeof_ids[i] < sizeof_ids[j]) {
            id[i] = j;
            sizeof_ids[j] += sizeof_ids[i];
            sizeof_ids[i] = 1;
        } else {
            id[j] = i;
            sizeof_ids[i] += sizeof_ids[j];
            sizeof_ids[j] = 1;
        }
        components--;
    }

    int find(int p) {
        if (p != id[p]) {
```

```
            id[p] = find(id[p]);
        }
        return id[p];
    }

    bool is_connected(int p, int q) {
        return find(p) == find(q);
    }

    int size() const {
        return components;
    }
};

struct edge {
    int u;
    int v;
    int cost;

    edge(int u, int v, int cost)
        :u(u), v(v), cost(cost) {
    }

    // sort by cost
    bool operator <(const edge &e) const {
        return cost < e.cost;
    }
};

std::vector<edge> edges;

int kruskal(int no_vertex) {
    int min_cost = 0;
    std::sort(edges.begin(), edges.end());
    disjoint_set ds(no_vertex);
    for (auto e : edges) {
        if (!ds.is_connected(e.u, e.v)) {
            min_cost += e.cost;
            ds.join(e.u, e.v);
        }
    }
    return min_cost;
}

void test_minimum_spanning_tree() {
    edges.push_back(edge(0, 1, 3));
    edges.push_back(edge(0, 2, 3));
    edges.push_back(edge(0, 3, 1));
    edges.push_back(edge(0, 4, 2));
    edges.push_back(edge(1, 2, 2));
    edges.push_back(edge(2, 3, 11));
    edges.push_back(edge(3, 4, 9));
    std::cout << "min cost: " << kruskal(5) << std::endl;
}
```

```
int main() {
    test_minimum_spanning_tree();
    return 0;
}
```

6. Dijkstra's Algorithm - Shortest Path Problem

Tiếp theo là giải thuật Dijkstra dùng để giải bài toán tìm đường đi ngắn nhất giữa các đỉnh trên đồ thị có trọng số. Cho đồ thị G = (V, E) thì Dijkstra có running time là O(E log(V)).

```
#include <iostream>
#include <vector>
#include <limits>
#include <iostream>
#include <queue>
#include <vector>
#include <limits>
#include <queue>
// priority queue
typedef std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int> >,
std::greater<std::pair<int, int>> > min_heap;
// cost and vertex
typedef std::pair<int, int> vertex;

const int oo = std::numeric_limits<int>::max();
const int MAX_VERTEX = 100;
std::vector<std::vector<int>> adj(MAX_VERTEX);
std::vector<std::vector<int>> cost(MAX_VERTEX, std::vector<int>(MAX_VERTEX, oo));
std::vector<int> dist(MAX_VERTEX, oo);

void dijkstra(int s) {
    min_heap pq;
    dist[s] = 0;
    pq.push(vertex(0, s));
    while (!pq.empty()) {
        int u = pq.top().second;
        int cost_to_u = pq.top().first;
        pq.pop();
        if (dist[u] == cost_to_u) {
            for (auto v : adj[u]) {
                if (dist[v] > dist[u] + cost[u][v]) {
                    dist[v] = dist[u] + cost[u][v];
                    pq.push(vertex(dist[v], v));
                }
            }
        }
    }
}

void add_edge(int u, int v, int c) {
    adj[u].push_back(v);
    cost[u][v] = c;
}

int main() {
```

```
        add_edge(0, 1, 13);
        add_edge(1, 2, 8);
        add_edge(2, 4, 1);
        add_edge(0, 2, 18);
        add_edge(0, 3, 20);
        add_edge(0, 4, 23);
        dijkstra(0);
        for (int v = 0; v <= 4; ++v) {
            std::cout << "shortest path from 0->" << v << ": " << dist[v] << "\n";
        }
        return 0;
    }
```

TÀI LIỆU THAM KHẢO

Website:

1. http://congdongcviet.com
2. http://www.cs.gordon.edu/
3. https://en.wikipedia.org
4. http://www.cplusplus.com/
5. http://www.tutorialspoint.com/
6. http://en.cppreference.com
7. http://windybook.com/