



VIỆN ĐIỆN TỬ - VIỄN THÔNG

School of Electronics and telecommunications



KỸ THUẬT LẬP TRÌNH C/C++

Chương 2:

TỔNG QUAN VỀ C/C++

Giảng viên: TS. Nguyễn Thị Kim Thoa

Học kỳ: 20201

3898043

5660163

63758

6752245

7196671

154963

2867239

17

OAS

ODS

NAV

WAV

IDS

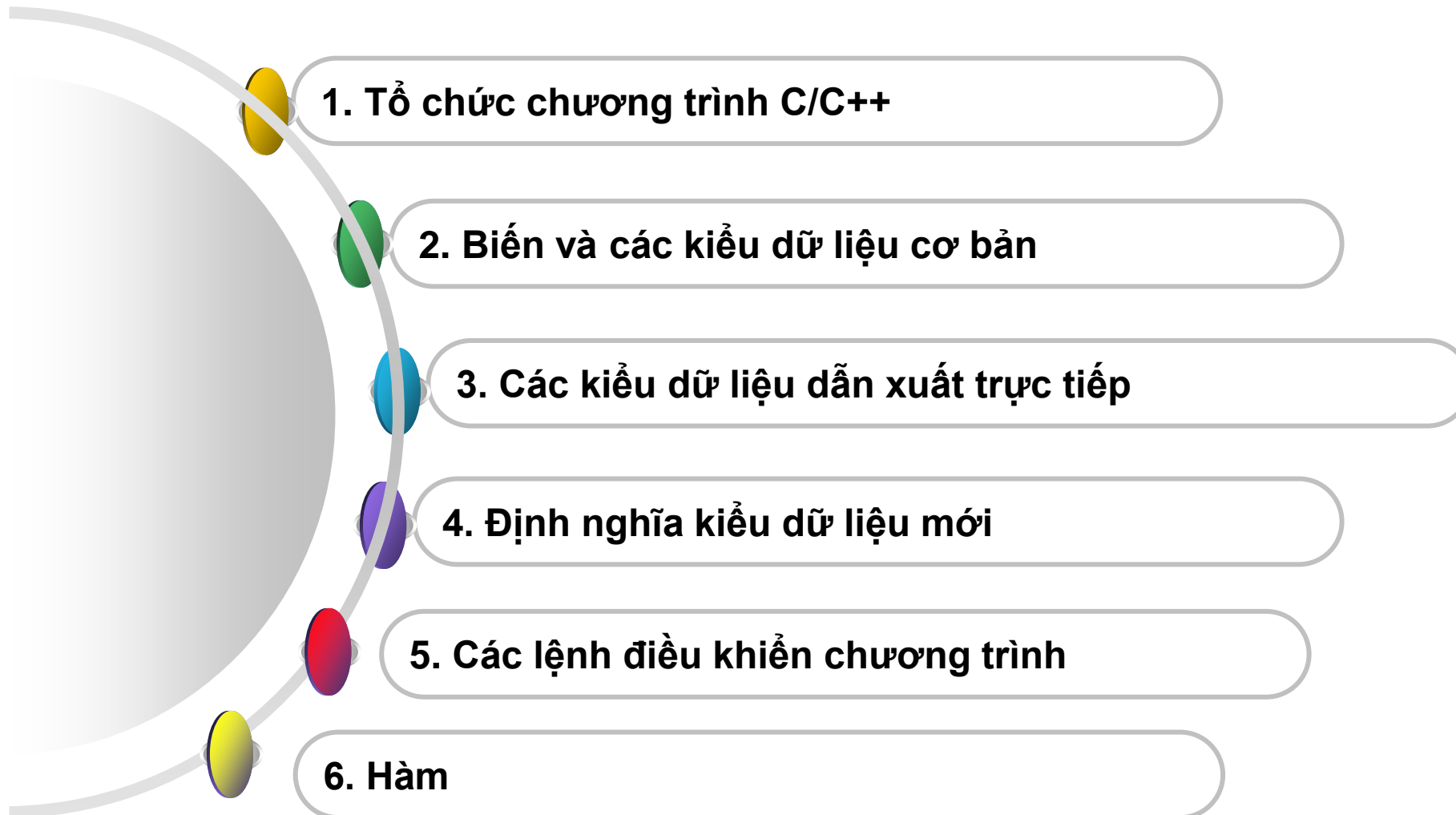
VUL

KAS

BAD



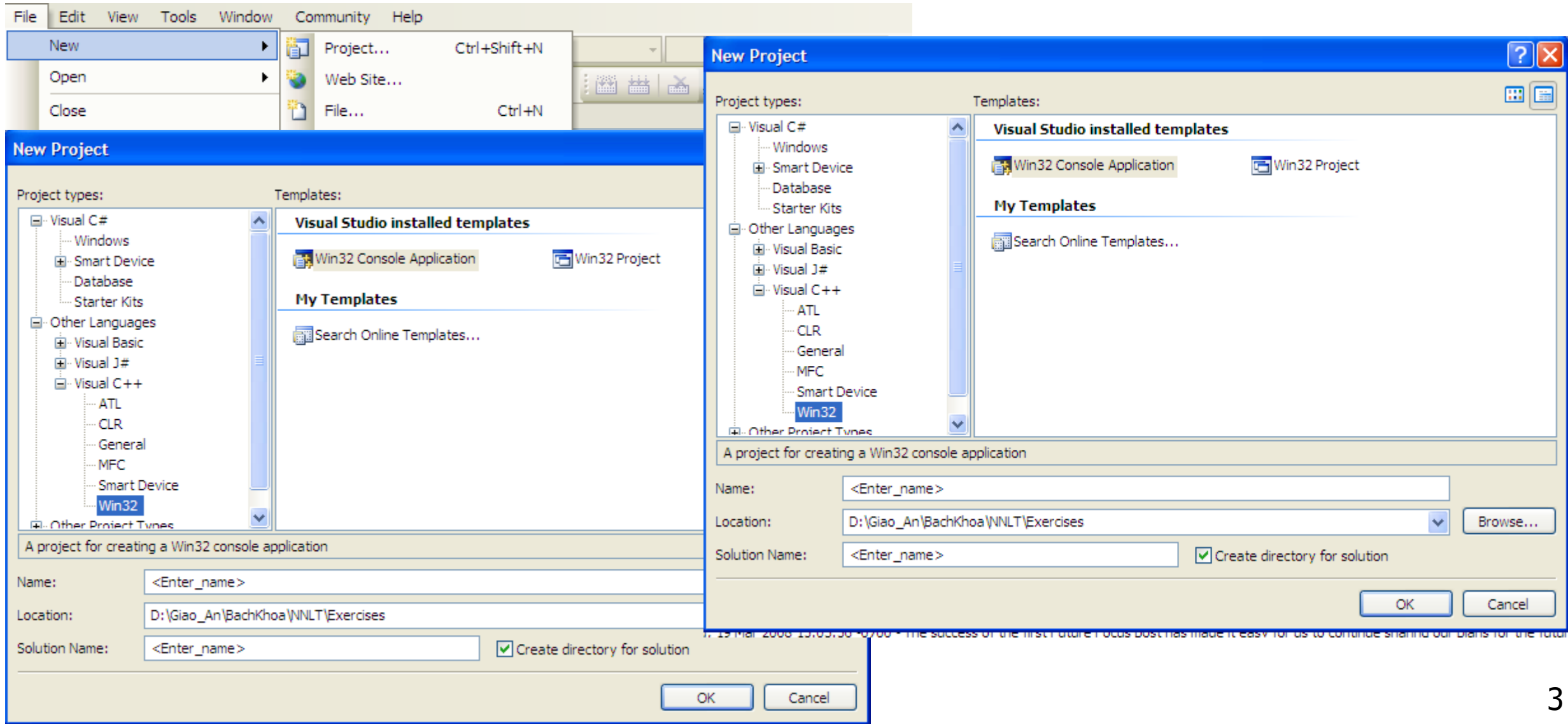
Nội dung





1. Tổ chức chương trình C/C++

Tạo dự án dùng visual studio .net 2019





Chương trình tính giai thừa: Phiên bản C

```
#include <stdio.h>
#include <conio.h>
```

Lệnh tiền xử lý: Khai báo sử dụng hàm thư viện

```
int gthua(int);
```

Khai báo hàm

```
void main() {
```

Chương trình chính

```
    char c = 'N';
```

```
    int N = 1;
```

```
    int kq;
```

```
    do {
```

```
        printf("\n Nhap mot nguyen duong");    /* Viet ra man hinh*/
```

```
        scanf("%d",&N);          /* Nhap tu ban phim cho N*/
```

```
        kq = gthua(N);
```

```
        printf("\n Giai thua cua %d la %d", N, kq);
```

```
        printf("\n Ban co muon tiep tục không? Y/N");
```

```
        c = getch();
```

```
    } while (c=='y' || c=='Y');
```

Lời chú thích

```
}
```

```
int gthua(int n) {
```

```
    int kq = 1;
```

```
    while (n > 1)
```

```
        kq *= n--;
```

```
    return kq;
```

Định nghĩa hàm (thân hàm)

```
}
```




Chương trình tính giai thừa: Phiên bản C++

```
#include <iostream.h>
```

```
#include <conio.h>
```

```
int gthua(int);
```

```
void main() {
```

```
    char c = 'N';
```

```
    int N = 1;
```

```
    do {
```

```
        cout << "\nNhap mot so nguyen duong: " << endl;
```

```
        cin >> N;
```

```
        int kq = gthua(N);
```

```
        cout << "\nGiai thua cua " << N << " la " << kq << endl;
```

```
        cout << "\nBan co muon tiep tục không? Y/N";
```

```
        c = getch();
```

```
    } while (c == 'y' || c == 'Y');
```

```
}
```

```
int gthua(int n) {
```

```
    int kq = 1;
```

```
    while (n > 1)
```

```
        kq *= n--;
```

```
    return kq;
```

```
}
```



Qui tắc soạn thảo mã nguồn

- Tên biến, tên hàm, tên kiểu mới:
 - ✓ Tránh sử dụng các từ khóa và tên kiểu cơ sở
 - ✓ Chỉ được dùng các ký tự sau: 'A'..'Z', 'a'..'z', '0'..'9', '_'
 - ✓ Chỉ được bắt đầu bằng chữ cái hoặc dấu _
 - ✓ Phân biệt giữa chữ hoa và chữ thường.
 - ✓ Nên đặt tên ngắn gọn và có ý nghĩa (có tính mô tả)
- Sau mỗi câu lệnh có chấm phẩy (;)
- Đoạn { ... } được coi là nhóm lệnh, không có dấu chấm phẩy sau đó, trừ khi khai báo kiểu
- Cấu trúc mã nguồn theo kiểu phân cấp => dễ đọc
- Bổ sung chú thích hợp lý (/*chú thích một đoạn*/ hoặc //chú thích một dòng)
- Chia một file lớn thành nhiều file nhỏ



@ Các từ khóa trong C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while



Các từ khóa trong C++

asm

case

const

delete

enum

float

friend

int

new

public

short

static_cast

this

typedef

unsigned

volatile

auto

catch

const_cast

else

false

dynamic_cast

goto

long

operator

register

signed

struct

throw

typeid

using

wchar_t

bool

char

continue

extern

double

export

if

mutable

private

reinterpret_cast

sizeof

switch

true

typename

virtual

while

break

class

default

do

explicit

for

inline

namespace

protected

return

static

template

try

union

void



@ Biên dịch (compile)

- Trong Visual studio .Net 2012: Gọi Compile (Ctrl + F7) để biên dịch riêng rẽ hoặc Build (F7) để kết hợp biên dịch và liên kết cho toàn bộ dự án
- Các kiểu lỗi biên dịch (compile error):
 - ✓ Lỗi cú pháp: Sử dụng tên sai qui định hoặc chưa khai báo, thiếu dấu chấm phẩy ;, dấu đóng }
 - ✓ Lỗi kiểu: Các số hạng trong biểu thức không tương thích kiểu, gọi hàm với tham số sai kiểu
 - ✓ ...
- Các kiểu cảnh báo biên dịch (warning):
 - ✓ Tự động chuyển đổi kiểu làm mất chính xác
 - ✓ Hàm khai báo có kiểu trả về nhưng không trả về
 - ✓ Sử dụng dấu = trong trường hợp nghi vấn là so sánh ==
 - ✓ ...



@ Liên kết (link)

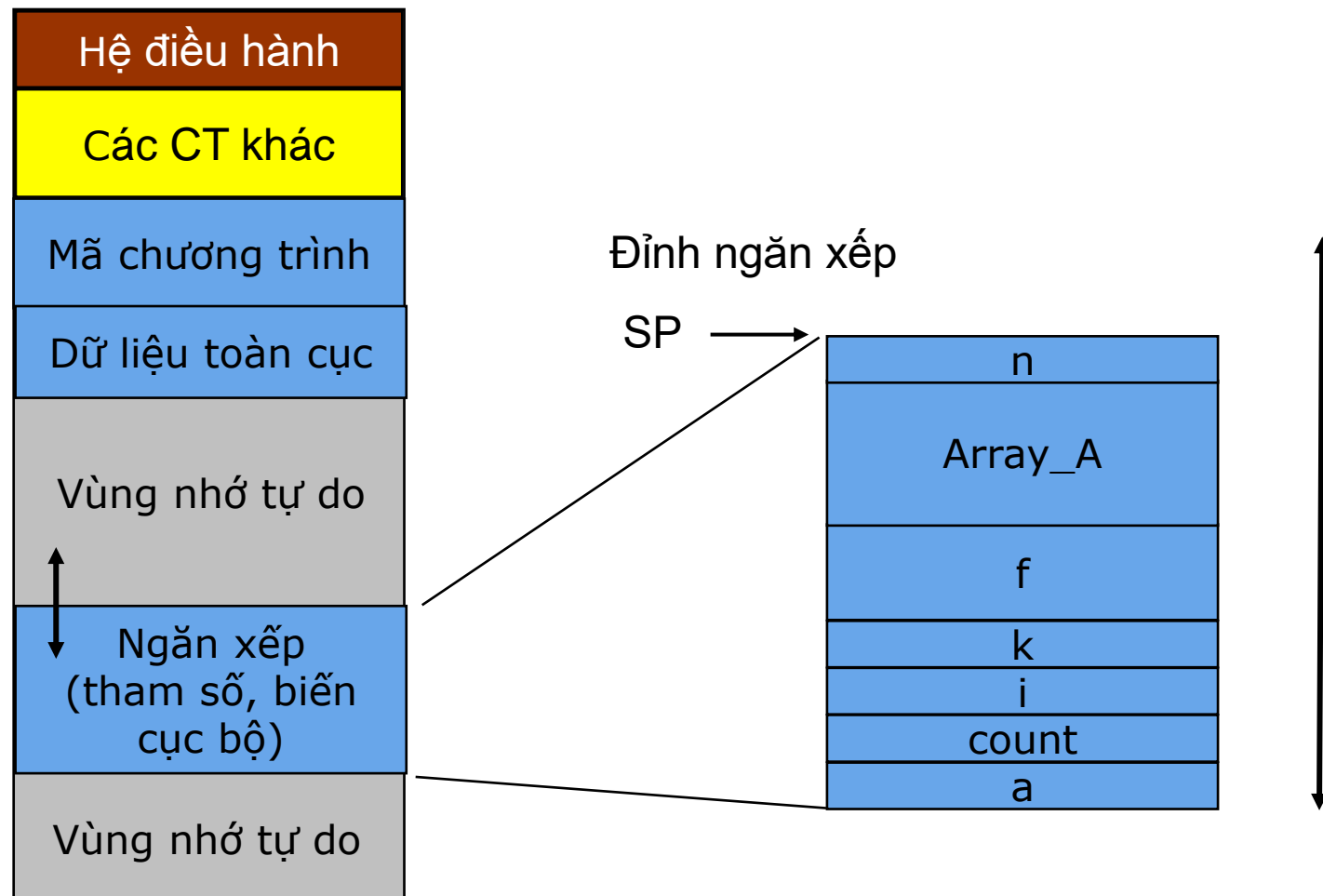
- Liên kết là quá trình ghép nhiều file đích (*.obj, *.lib) để tạo ra chương trình chạy cuối cùng *.exe
- Trong Visual studio .Net 2012: Gọi Build (F7)
- Lỗi liên kết có thể là do:
 - ✓ Sử dụng hàm nhưng không có định nghĩa hàm
 - ✓ Biến hoặc hàm được định nghĩa nhiều lần
 - ✓ ...



@ Chạy thử và gỡ rối (debug)

- Chạy thử trong Visual studio .Net 2012: Start without debugging hoặc Ctrl+F5
- Tìm lỗi:
 - ✓ Lỗi khi chạy là lỗi thuộc về phương pháp, tư duy, thuật toán, không phải về cú pháp
 - ✓ Lỗi khi chạy bình thường không được báo
 - ✓ Lỗi khi chạy rất khó phát hiện, vì thế trong đa số trường hợp cần tiến hành debug.
- Chạy Debug trong Visual studio .Net 2012:
 - ✓ Chạy tới chỗ đặt cursor: F9
 - ✓ Chạy từng dòng lệnh: F10
 - ✓ Chạy vào trong hàm: F11
 - ✓ Chạy tiếp bình thường: F5
 - ✓ Xem kết quả dưới cửa sổ Output hoặc gọi QuickWatch

@ Tổ chức bộ nhớ





2. Biến và các kiểu dữ liệu cơ bản

- Biểu thức = dữ liệu + phép toán
- Biểu diễn dữ liệu: Thông qua **biến** hoặc **hằng số**, kèm theo **kiểu**
- Nội dung trong phần này:
 - ✓ Các kiểu dữ liệu cơ bản
 - ✓ Các phép toán áp dụng
 - ✓ Tương thích và chuyển đổi kiểu
 - ✓ Khai báo biến, phân loại biến



Các kiểu dữ liệu cơ bản của C/C++

Kiểu	Kích cỡ thông dụng (tính bằng bit)	Phạm vi tối thiểu
char	8	-127 to 127
signed char	8	-127 .. 127
unsigned char	8	0 .. 255
int	32	-32767 .. 32767
signed int	32	-nt-
unsigned int	32	0 .. 65535
short	16	-32767 .. 32767
signed short	16	nt
unsigned short	16	0 .. 65535
long	32	-2147483647 .. 2147483647
signed long	32	nt
unsigned long	32	0 .. 4294967295
float	32	Độ chính xác 6 chữ số
double	64	Độ chính xác 10 chữ số
long double	80	Độ chính xác 10 chữ số
bool (C++)	-	-
wchar_t (C++)	16	-32767 .. 32767



Các phép toán cơ bản

<i>Phép toán</i>	<i>Ký hiệu</i>	<i>Kiểu nguyên</i>	<i>Kiểu số thực</i>	<i>Kiểu bool</i>
<i>Gán</i>	=	X	X	X
<i>Số học</i>	+, -, *, / , +=, -=, *=, /=	X	X	x
	%, %=	X		x
	++, --	X		x
<i>So sánh</i>	>, <, >=, <=, ==, !=	X	X	X
<i>Logic</i>	&&, , !	X	X	X
<i>Logic bit</i>	&, , ^, ~ &=, =, ^=	X		x
<i>Dịch bit</i>	<<, >>, <<=, >>=	X		x
<i>Lựa chọn</i>	? :	X	X	X
<i>Lũy thừa?</i>	Không có!			



Tương thích và chuyển đổi kiểu

- Những kiểu dữ liệu tương thích nhau có thể tự động chuyển đổi kiểu:
 - ✓ Giữa các kiểu số nguyên với nhau (lưu ý phạm vi giá trị)
 - ✓ Giữa các kiểu số thực với nhau (lưu ý độ chính xác)
 - ✓ Giữa các kiểu số nguyên và số thực (lưu ý phạm vi giá trị và độ chính xác)
 - ✓ Kiểu bool sang số nguyên, số thực: true => 1, false => 0
 - ✓ Số nguyên, số thực sang kiểu bool: $\neq 0$ => true, 0 => false
- Nếu có lỗi hoặc cảnh báo => khắc phục bằng cách ép chuyển đổi kiểu
 - ✓ VD:
 $i = \text{int}(5.1) \% 2; //C$
 $j = (\text{int})5.1 + 2; // C++$

@ Khai báo biến

char	c = 'N';	← Khai báo và khởi tạo giá trị
bool	b = true ;	
int	kq;	← Chỉ khai báo, giá trị bất định
double	d;	
long	count, i=0;	← Khai báo kết hợp, chỉ i=0
unsigned	vhexa= 0x 00fa;	← Đặt giá trị đầu hexa
unsigned	vocal= 0 72;	← Đặt giá trị đầu octal -> 58 chứ không phải 72

- C: Toàn bộ biến phải khai báo ngay đầu thân hàm
- C++: Có thể khai báo tại chỗ nào cần
- Phân loại biến:
 - ✓ Biến toàn cục: Khai báo ngoài hàm, lưu giữ trong vùng nhớ dữ liệu chương trình
 - ✓ Biến cục bộ: Khai báo trong thân hàm, lưu giữ trong ngăn xếp (vùng nhớ tự do)
 - ✓ Tham số: Khai báo trên danh sách tham số của hàm, lưu giữ trong ngăn xếp

@ Ví dụ khai báo các loại biến

Biến toàn cục

Biến cục bộ

Hai biến cục bộ cùng tên ở hai phạm vi khác nhau, không liên quan gì đến nhau!

```
int N = 1;
void main() {
    char c = 'N';
    do {
        cout<<"\nNhap so> 0:";
        cin>>N;
        int kq = gthua(N);
        ...
    } while (c == 'y' || c == 'Y')
}

int gthua(int n) {
    int kq = 1;
    while (n > 1)
        kq *= n--;
    return kq;
}
```

Tham số



3. Các kiểu dữ liệu dẫn xuất trực tiếp

- Kiểu liệt kê
- Kiểu hằng
- Kiểu con trỏ
- Kiểu mảng
- Kiểu tham chiếu (C++)

@ Kiểu liệt kê (enum)

- Tương tự như một kiểu số nguyên.
- Được dùng khi:
 - ✓ Cần hạn chế phạm vi sử dụng
 - ✓ Hoặc khi muốn sử dụng tên cụ thể thay cho hằng số nguyên

- Ví dụ

```
void main()
{
    enum Color {Red, Green, Blue};
    enum WeekDay {Mon = 2, Tue, Wed, Thu, Fri, Sat, Sun = 1 };
    //Tue=3, Wed=4...
    int n=Red;
    cout<<n; //in ra số 0
    Color c=Green;
    n=c; //n=1;
}
```

- Lưu ý: không thể gán trực tiếp một giá trị nguyên cho một biến enum mà phải ép kiểu

✓ VD:

c=2;//Error

c=Color(2);//OK

@ Kiểu hằng (const)

- Khi muốn một biến không bị thay đổi trong suốt chương trình => khai báo hằng số

```
void main() {  
    const double pi = 3.1412;  
    const int x = 1;  
    pi = 3.14;           // error  
    x = 2;               // error  
}
```



Kiểu con trỏ

- Trong chương trình, mỗi biến chiếm một số byte nhất định và lưu ở một vị trí cụ thể trong bộ nhớ máy tính
- Con trỏ là một biến mang địa chỉ của một ô nhớ (của một biến khác hoặc một hàm)
- Cách khai báo: `Data_type* pointer_var_name;`

```
int    x = 2;  
int*    p = &x; // p chứa địa chỉ của x  
*p = 3;      // x=3  
int    y;  
p = &y;      // p chứa địa chỉ của y  
*p = 5;      // y = 5, x = 3
```



Con trỏ hằng và hằng con trỏ

- Con trỏ hằng:

- ✓ Là con trỏ trỏ đến vùng dữ liệu hằng.
- ✓ Không thể thay đổi giá trị mà nó đang trỏ đến.
- ✓ Có thể thực hiện tăng giảm địa chỉ con trỏ hay cho nó trỏ đi nơi khác.
- ✓ Khai báo:

```
const int* p;
```

- Hằng con trỏ:

- ✓ Là 1 con trỏ nếu trỏ vào 1 ô nhớ nào đó thì nó sẽ nằm “dính chặt” vào ô nhớ đó.
- ✓ Không thể cho nó trỏ đi nơi khác và tăng giảm địa chỉ của con trỏ.
- ✓ Có thể thay đổi được giá trị mà nó trỏ đến.
- ✓ Khai báo:

```
int* const p;
```

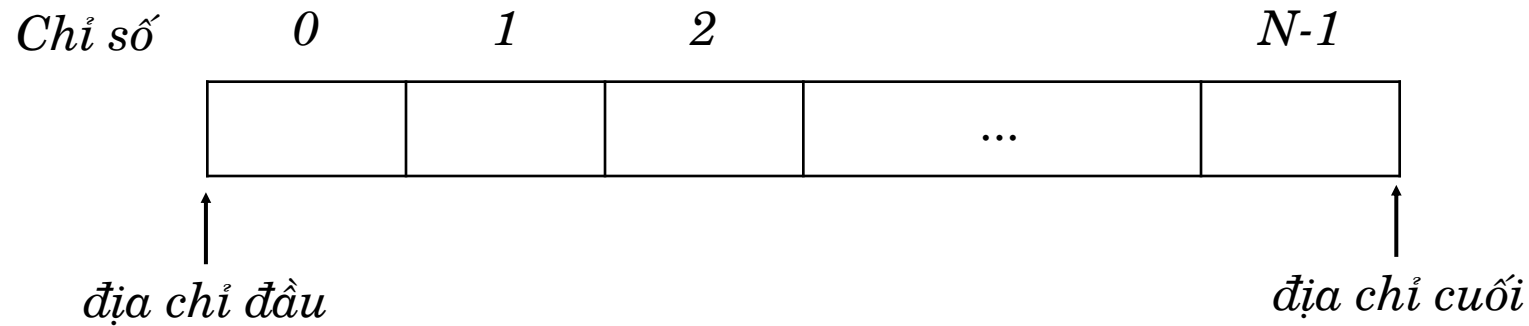


@ Kết luận về con trỏ

- Con trỏ là một biến chứa địa chỉ byte đầu của một biến dữ liệu, được sử dụng để truy cập gián tiếp dữ liệu đó
- Nếu khai báo con trỏ mà không khởi tạo, mặc định con trỏ mang một địa chỉ bất định
- Để khởi tạo giá trị cho con trỏ, ta có thể cho nó trỏ vào một địa chỉ nhớ cụ thể hoặc địa chỉ của một biến khác thông qua toán tử lấy địa chỉ &
- Con trỏ có thể mỗi lúc đại diện cho một biến dữ liệu khác, do đó địa chỉ mà con trỏ mang có thể thay đổi được.
- Dùng toán tử * để truy nhập nội dung của biến mà con trỏ trỏ tới, có thể đọc hoặc thay đổi nội dung của biến đó.
- Không bao giờ sử dụng toán tử truy nhập nội dung, nếu con trỏ chưa mang một địa chỉ ô nhớ mà chương trình có quyền kiểm soát



Kiểu mảng



- Số phần tử của mảng: N
- Số lượng các phần tử của mảng (tĩnh) là cố định
- Các phần tử có cùng kiểu
- Các phần tử được sắp xếp kế tiếp trong bộ nhớ
- Có thể truy nhập từng phần tử một cách tự do theo chỉ số hoặc theo địa chỉ



Khai báo mảng

- Số phần tử của mảng phải là hằng số nguyên.
- Khai báo không khởi tạo:

```
int      a[5];  
enum     {index = 5};  
double   b[index];  
const int N = 2;  
char     c[N];
```

- Khai báo với số phần tử và khởi tạo giá trị các phần tử

```
int      d[3]= {1, 2, 3};  
double   e[5]= {1, 2, 3};  
char     f[4]= {0};
```



Khai báo mảng [...]

- Khai báo và khởi tạo giá trị các phần tử, số phần tử được tự động xác định

```
int    a[] = {1, 2, 3, 4, 5};
```

```
double b[] = {1, 2, 3};
```

```
double c[] = {0};
```

```
char    s[] = {'a'};
```

- Khai báo mảng nhiều chiều

```
double M[2][3];
```

```
int X[3][] = {{1,2},{3,4},{5,6}};
```

```
short T[3][2] = {1,2,3,4,5,6};
```




Mảng đặc biệt: Chuỗi ký tự

- Trong C/C++, chuỗi ký tự không phải là kiểu cơ bản, mà thực chất là một mảng
- Phân biệt chuỗi ký tự thường và chuỗi ký tự kết 0

```
char s1[] = {'H', 'E', 'L', 'L', 'O'};  
char    s2[] = "HELLO";  
wchar_t s3[] = "HELLO";
```

- Đa số các hàm trong thư viện C làm việc với chuỗi ký tự kết 0
- Với C++, chuỗi ký tự được định nghĩa bằng lớp string trong thư viện chuẩn, không sử dụng byte kết 0



@ Con trỏ và mảng

- Tên mảng là địa chỉ của phần tử đầu tiên của mảng.

```
void main() {  
    int a[3]; // a có 3 phần tử, giá trị chưa xác định  
    int* p;  
    p = a;           // p trỏ đến a[0]  
    p = &a[0];       // tương tự như trên  
    *p = 1;          // a[0]=1  
    ++p;             // p trỏ đến a[1]  
    *p = 2;          // a[1]=2  
    p++;             // p trỏ đến a[2]  
    *p = 3;          // a[2]=3  
}
```



Kết luận về mảng

- Mảng là một tập hợp các dữ liệu cùng kiểu, sắp xếp liên kề trong bộ nhớ
- Số phần tử của mảng (tĩnh) là cố định (khi khai báo phải là hằng số), không bao giờ thay đổi được
- Có thể truy cập các phần tử mảng thông qua tên mảng kèm theo chỉ số hoặc thông qua biến con trỏ (theo địa chỉ của từng phần tử)
- Biến mảng (tĩnh) thực chất là một con trỏ hằng, mang địa chỉ của phần tử đầu tiên
- Có thể đặt giá trị đầu cho các phần tử của mảng qua danh sách khởi tạo.
- Không gán được hai mảng cho nhau. Nếu cần sao chép hai mảng thì phải sử dụng hàm
- Không bao giờ được phép truy nhập với chỉ số nằm ngoài phạm vi, nếu N là số phần tử thì phạm vi cho phép là từ $0..N-1$
- Con trỏ không phải là một mảng, nó chỉ có thể mang địa chỉ của một mảng và sử dụng để quản lý mảng (dù là động hay tĩnh)



Kiểu tham chiếu (C++)

- Biến tham chiếu không được cấp phát bộ nhớ, không có địa chỉ riêng.
- Một biến tham chiếu là một biến đại diện trực tiếp cho một biến khác (dùng để làm bí danh cho biến đó và nó sử dụng vùng nhớ của biến này)
- Ý nghĩa: sử dụng chủ yếu trong truyền tham số cho hàm

```
void main() {  
    int a = 1;  
    int& r = a;    // r tham chiếu đến a  
    r = 2;        // a = 2  
    int& r2;    // error, tham chiếu phải được khởi tạo khi khai báo  
    int& r3 = 0; // error, tham chiếu phải đại diện cho một biến khác  
    int b = 0;  
    r = b;        // r = 0, a=0  
    r = 3;        // r = a= 3, b=0  
}
```



Tham chiếu hằng

- Có thể tham chiếu đến một biến hoặc một hằng
- Ý nghĩa: thường được sử dụng làm đối của hàm để cho phép hàm sử dụng giá trị của các tham số trong lời gọi hàm nhưng tránh không làm thay đổi giá trị của các tham số
- Không cho phép dùng tham chiếu hằng để thay đổi giá trị vùng nhớ mà nó tham chiếu.

• VD:

```
int n=10, m;  
const int &r=n;  
n++; //OK  
m=2*n; //m=22  
r=r+1; //Error
```

- Có thể tham chiếu đến một hằng.
VD:

```
const int y = 7;  
int& ref=y; //error  
const int &ref = y; //ok
```




Kết luận về biển tham chiếu

- Khi khai báo phải chỉ rõ nó tham chiếu đến biển nào.
- Có thể tham chiếu đến một phần tử của mảng
- Không cho phép khai báo mảng tham chiếu
- Có thể tham chiếu đến một hằng.
- Không cho phép dùng tham chiếu hằng để thay đổi giá trị vùng nhớ mà nó tham chiếu.

@ Typedef

- Từ khóa typedef tạo ra một tên mới cho một kiểu dữ liệu có sẵn, không định nghĩa một kiểu mới
- Ý nghĩa: đưa tên mới dễ nhớ, phù hợp với ứng dụng cụ thể.
- Cú pháp: typedef Kieu_dang_co Ten_moi

```
void main()
{
    typedef int tuoi; /* Tạo tên mới cho kiểu int là "tuoi" */
    typedef char ten[30]; /* Tạo tên mới cho kiểu char là "ten" */
    tuoi sv_tuoi, gv_tuoi;
    ten sv_ten, gv_ten;
    ...
}
```



5. Định nghĩa kiểu dữ liệu mới

- Cấu trúc (struct):
 - ✓ Tập hợp những dữ liệu dùng để mô tả một đối tượng, truy nhập theo tên (biến thành viên).
 - ✓ Thông dụng nhất trong C, ý nghĩa được mở rộng trong C++
- Hợp nhất (union):
 - ✓ Một tên kiểu chung cho nhiều dữ liệu khác nhau (chiếm cùng chỗ trong bộ nhớ).
 - ✓ Ít thông dụng trong cả C và C++
- Lớp (class):
 - ✓ mở rộng struct cũ
 - ✓ thêm những hàm thành viên.
 - ✓ Chỉ có trong C++

@ Kiểu struct

- Cú pháp khai báo:

```
struct struct_name
{
    //khai báo các thành phần của struct
    type1 field1;
    type2 field2;
    ...
} [bien_co_kieu_struct_name];
```

- Cách truy nhập vào các thuộc tính:

Ten_bien_kieu_struct.Ten_thuoc_tinh.

- Khởi tạo giá trị cho biến **struct** giống như với mảng, đặt các giá trị trong dấu { }, phân cách nhau bằng dấu “,”.



Ví dụ minh họa struct

```
#include <iostream.h>
struct emp
{
    int MaSo;
    char ten[35];
};
void main()
{
    struct emp e1={100,"Nguyen X"};
    cout<<"Eno: " <<e1.Maso<<endl;
    cout<<"Ename: "<<e1.ten;
}
```

C++ ←



Kết luận về struct

- Cấu trúc (struct) được sử dụng để nhóm các dữ liệu liên quan mô tả một đối tượng, các dữ liệu có thể cùng hoặc khác kiểu
- Định nghĩa kiểu cấu trúc bằng cách khai báo tên các biến thành viên. Định nghĩa kiểu cấu trúc chưa phải là định nghĩa các biến cụ thể, vì thế không được đặt giá trị đầu cho các biến
- Kích cỡ của cấu trúc \geq tổng kích cỡ các thành viên
- Truy cập một biến cấu trúc: `tên_biến_cấu_trúc.tên_biến_thành_viên`
- Các kiểu cấu trúc có thể lồng vào nhau, trong cấu trúc có thể sử dụng mảng, một mảng có thể có các phần tử là cấu trúc, v.v...
- Các biến có cùng kiểu cấu trúc có thể gán cho nhau, có thể sử dụng để khởi tạo cho nhau (khác hẳn với mảng)
- Có thể sử dụng con trỏ để truy nhập dữ liệu cấu trúc thông qua toán tử toán tử (*.) hoặc (->)



Hợp nhất (union)

- Cú pháp khai báo:

```
union union_name
{
    //Khai báo các thành phần của union
    Type1 field1;
    Type2 field2;
    ...
};
```

- Ví dụ:

```
union VD
{
    unsigned long u;
    unsigned char a[4];
};
void main()
{ VD vd1;
vd1.u=0xDDCCBBAA;
cout<<vd1.a[0];
cout<<vd1.a [1];
cout<<vd1.a[2];
cout<<vd1.a [3];
}
```



Kết luận về hợp nhất

- Hợp nhất (union) là một tập hợp (không có cấu trúc chặt chẽ) chứa các biến sử dụng chung ô nhớ, ở mỗi ngữ cảnh chỉ sử dụng một biến riêng biệt
- Union thường được sử dụng khi dữ liệu đầu vào có thể có kiểu khác nhau
- Các thành viên của một union không liên quan đến nhau, không cùng nhau tạo thành một thực thể thống nhất
- Kích cỡ của union bằng kích cỡ của biến lớn nhất
- Khai báo kiểu union tương tự như khai báo struct, nhưng ý nghĩa khác hẳn
- Truy nhập biến thành viên cũng tương tự như struct, có thể qua biến trực tiếp hoặc qua biến con trỏ.
- Union có thể chứa struct, struct có thể chứa union, union có thể chứa mảng, các phần tử của mảng có thể là union.



Các lệnh điều khiển chương trình

- Lệnh switch ... case
- Lệnh while(dieu_kien)...
- Lệnh do ... while(dieu_kien)
- Lệnh for ...
- Lệnh break thoát khỏi một vòng lặp
- Lệnh continue
- Lệnh if ... else
- Lệnh goto

@ Cấu trúc if-else

- Cú pháp:

```
if ( expression )  
    stmt1;
```

Hoặc

```
if ( expression )  
    stmt1;  
else  
    stmt2;
```

- Khi số câu lệnh trong mỗi nhánh rẽ lớn hơn 1 thì đặt chúng trong dấu { }



Ví dụ về if-else

```
if (npoints >= 60)
    cout << "Passed";
if (npoints >= 80 && npoints <= 90) {
    grade = 'A';
    cout << grade;
}
```

```
if (npoints >= 90)
    cout << 'A';
else if (npoints >= 80)
    cout << 'B';
else if (npoints >= 70)
    cout << 'C';
else if (npoints >= 60)
    cout << 'D';
else
    cout << 'F';
```



Cấu trúc switch-case

- Cú pháp:

```
switch( expression )
{
    case constant-expr1:
        stmt;
        break;
    case constant-expr2:
        stmt;
        break;
    . . .
    default:
        stmt
}
```

- Ví dụ:

```
switch( c )
{
    case 'Y':
        cout<< "Yes" ;
        break;
    case 'N':
        cout<<"No" ;
        break;
    default:
        cout<< "What?" ;
}
```


@ Cấu trúc switch-case (...)

- Có thể dùng nhiều nhãn case trên cùng một nhánh

```
switch( c )
{
    case 'Y':
    case 'y':
        cout<< "Yes" ;
        break;
    case 'N': case 'n':
        /* 2 điều khiển trên cùng 1
dòng */
        cout<< "No";
        break;
    default:
        cout<<"What?";
}
```

@ Cấu trúc while

- Cú pháp:

```
while ( expression )  
    stmt ;
```

- Lệnh (khối lệnh) *stmt* chỉ được thực hiện khi *expression* là đúng.

- Ví dụ:

```
x = 1;  
while ( (x * 5) >= (x * x) )  
{  
    cout<<x;  
    x++;  
}
```



Cấu trúc do...while

- Cú pháp:

```
do  
    stmt ;  
while ( expression );
```

- Lệnh (khối lệnh) stmt được thực hiện cho đến khi expression còn đúng.
- Ví dụ:

```
lng = 0;  
do  
{  
    c = getchar();  
    ++lng;  
}  
while ( c != '\n' );  
cout<< "line length is: "<<lng  
;
```



Cấu trúc for

- Cú pháp:

```
for ( init-expr; cont-expr; loop-expr )  
    stmt;
```

- Thực hiện:

- ✓ Tính toán init-expr nếu có
- ✓ Tính toán cont-expr nếu có
- ✓ Nếu cont-expr là true hoặc được bỏ qua thì thực hiện stmt, sau đó loop-expr được thực hiện
- ✓ Lặp lại bước 2 đến khi cont-expr là false

@ Cấu trúc for (...)

- Ví dụ:

```
/* Bảng cửu chương*/  
const int max=9  
for ( i = 0; i <= max; i++ )  
{  
    for ( j = 0; j <= max; j++ )  
    {  
        cout<<i<< " times"<<j<<" is"<<i*j<<"\n";  
    }  
}
```

```
9 times 0 is 0  
9 times 1 is 9  
9 times 2 is 18  
9 times 3 is 27  
9 times 4 is 36  
9 times 5 is 45  
9 times 6 is 54  
9 times 7 is 63  
9 times 8 is 72  
9 times 9 is 81
```

Hàm là gì?

- Một đơn vị tổ chức chương trình, một đoạn mã chương trình có cấu trúc để thực hiện một chức năng nhất định, có giá trị sử dụng lại
- Các hàm có quan hệ với nhau thông qua lời gọi, các biến tham số (đầu vào, đầu ra) và giá trị trả về
- Cách thực hiện cụ thể một hàm phụ thuộc nhiều vào dữ kiện (tham số, đối số của hàm):
- Thông thường, kết quả thực hiện hàm mỗi lần đều giống nhau nếu các tham số đầu vào như nhau
- Một hàm không có tham số thì giá trị sử dụng lại rất thấp
- Trong C/C++: Không phân biệt giữa thủ tục và hàm, cả đoạn mã chương trình chính cũng là hàm

- Yêu cầu bài toán: Tính tổng một dãy số nguyên (liên tục) trong phạm vi do người sử dụng nhập. In kết quả ra màn hình.
- Các nhiệm vụ:
 - ✓ Nhập số nguyên thứ nhất:
 - Yêu cầu người sử dụng nhập
 - Nhập số vào một biến
 - ✓ Nhập số nguyên thứ hai
 - Yêu cầu người sử dụng nhập
 - Nhập số vào một biến
 - ✓ Tính tổng với vòng lặp
 - ✓ Hiển thị kết quả ra màn hình



Ví dụ: Cách 1

```
#include <iostream.h>
void main() {
    int a, b;
    char c;
    do {
        cout << "Nhap so nguyen thu nhat: ";
        cin >> a;
        cout << "Nhap so nguyen thu hai: ";
        cin >> b;
        int tong = 0;
        for (int i = a; i <= b; ++i)
            tong += i;
        cout << "Tong tu " << a << " den " << b
            << " la " << tong << endl;
        cout << "ban co muon tinh tiep? (Y/N):";
        cin >> c;
    } while (c == 'y' || c == 'Y');
}
```




Cách 2 (phân hoạch hàm)

```
#include <iostream.h>

int  Nhap();
int  Tinh_tong(int,int);
void Hien_thi(int a, int b, int kq);

void main() {
    char c;
    do {
        int a = Nhap();
        int b = Nhap();
        int T = Tinh_tong(a,b);
        Hien_thi(a,b,T);
        cout << "Ban co muon tinh tiep? (Y/N):";
        cin  >> c;
    } while (c == 'y' || c == 'Y');
}
```



Cách 2 (...)

```
int Nhap() {  
    cout << "Nhap mot so nguyen: ";  
    int N;  
    cin >> N;  
    return N;  
}  
  
int Tinh_tong(int a, int b) {  
    int T = 0;  
    for (int i = a; i <= b; ++i)  
        T += i;  
    return T;  
}  
  
void Hien_thi (int a, int b, int kq) {  
    cout << "Tong tu " << a << " den " << b  
        << " la " << kq << endl;  
}
```



Ưu nhược điểm của phân hoạch hàm

- Chương trình dễ đọc hơn => dễ phát hiện lỗi
- Chương trình dễ mở rộng hơn
- Có giá trị sử dụng lại
- Mã nguồn dài hơn
- Mã chạy lớn hơn
- Chạy chậm hơn

=> Không phải cứ phân hoạch thành nhiều hàm là tốt, mà vấn đề nằm ở cách phân hoạch và thiết kế hàm làm sao cho tối ưu!



Khai báo và định nghĩa hàm

- Định nghĩa hàm: tạo mã thực thi hàm

Kiểu trả về Tên hàm Tham số (hình thức)

```
int Tinh_tong(int a, int b) {  
    int T = 0;  
    for (int i = a; i <= b; ++i)  
        T += i;  
    return T;  
}
```

- Khai báo hàm thuần túy: không tạo mã hàm

```
int Tinh_tong (int a, int b);
```

Kiểu trả về Tên hàm Kiểu tham số

- Tại sao và khi nào cần khai báo hàm?



Khai báo hàm và lời gọi hàm

- Ý nghĩa của khai báo hàm:
 - ✓ Khi cần sử dụng hàm (gọi hàm)
 - ✓ Trình biên dịch cần lời khai báo hàm để kiểm tra lời gọi hàm đúng hay sai về cú pháp, về số lượng các tham số, kiểu các tham số và cách sử dụng giá trị trả về.
- Có thể định nghĩa hàm sau lời gọi hàm
- Gọi hàm: yêu cầu thực thi mã hàm với tham số thực tế (tham trị)

```
int x = 5;  
int k = Tinh_tong(x, 10);  
          ↑      ↑  
          Tên hàm Đối số (thực tế)
```



Khai báo hàm C/C++ ở đâu?

- Ở phạm vi toàn cục (ngoài bất cứ hàm nào)
- Một hàm phải được khai báo trước lời gọi đầu tiên trong một tệp tin mã nguồn
- Nếu sử dụng nhiều hàm thì sẽ cần rất nhiều dòng mã khai báo (mất công viết, dễ sai và mã chương trình lớn lên?):
 - ✓ Nếu người xây dựng hàm (định nghĩa hàm) đưa sẵn tất cả phần khai báo vào trong một tệp tin => Header file (*.h, *.hx,...) thì người sử dụng chỉ cần bổ sung dòng lệnh
`#include <filename>`
 - ✓ Mã chương trình không lớn lên, bởi khai báo không sinh mã!
- Một hàm có thể khai báo nhiều lần tùy ý!

@ Định nghĩa hàm ở đâu?

- Ở phạm vi toàn cục (ngoài bất cứ hàm nào)
- Có thể định nghĩa trong cùng tệp tin với mã chương trình chính, hoặc tách ra một tệp tin riêng. Trong Visual C++:
 - *.c => C compiler,
 - *.cpp => C++ compiler
- Một hàm đã có lời gọi thì phải được định nghĩa chính xác 1 lần trong toàn bộ (dự án) chương trình, trước khi gọi trình liên kết (lệnh Build trong Visual C++)
- Đưa tệp tin mã nguồn vào dự án, không nên:
#include "xxx.cpp"
- Một hàm được định nghĩa sẵn bằng C, C++, hợp ngữ hoặc bằng một ngôn ngữ khác và dùng trong C/C++ => Sử dụng hàm không cần mã nguồn!
- Một thư viện cho C/C++ bao gồm:
 - ✓ Header file (thường đuôi *.h, *.hxx, ..., nhưng không bắt buộc)
 - ✓ Tệp tin mã nguồn (*.c, *.cpp, *.cxx,...) hoặc mã đích (*.obj, *.o, *.lib, *.dll, ...)

@ Tham số và đối số

- Cách gọi khác của tham số và đối số là tham biến và tham trị

```
int Tinh_tong (int a, int b) {  
    ...  
}
```

Tham số
(hình thức)

```
int x = 5;  
int k = Tinh_tong (x, 10);  
...
```

Đối số
(thực tế)

```
int a = 2;  
k = Tinh_tong (a,x);
```


@ Truyền giá trị

```
int Tinh_tong (int, int);
```

```
void main() {  
    int x = 5;  
    int k = Tinh_tong(x, 10);
```

```
    ...
```

```
}
```

```
// Định nghĩa hàm
```

```
int Tinh_tong(int a, int b) {
```

```
    ...
```

```
}
```

SP →

b = 10

SP →

a = 5

k = 45

x = 5

Ngăn xếp

@ Ví dụ về truyền giá trị

```
#include <iostream.h>
void func(int N) {
    cin>>N; //nhap vao so 15
}

void main() {
    int x = 5;
    cout<< "Hay nhap so nguyen: ";
    func(x);
    cout << "Bay gio x la " << x;
    ...
}
```

❖ Kết quả???



@ Truyền giá trị (...)

- Truyền giá trị là cách thông thường trong C
- Tham số chỉ nhận được bản sao của đối số (biến đầu vào)
- Thay đổi tham số chỉ làm thay đổi vùng nhớ cục bộ, không làm thay đổi đối số
- Tham số chỉ có thể mang đầu vào của hàm, không chứa được kết quả của hàm
- Truyền giá trị trong nhiều trường hợp kém hiệu quả do mất công sao chép dữ liệu



Truyền địa chỉ

```
int Tong_mang(int* p, int N);
```

```
// Goi ham
```

```
void main() {
```

```
    int a[] = {1, 2, 3, 4};
```

```
    int k = Tong_mang(a,4);
```

```
    ...
```

```
}
```

```
// Dinh nghia ham
```

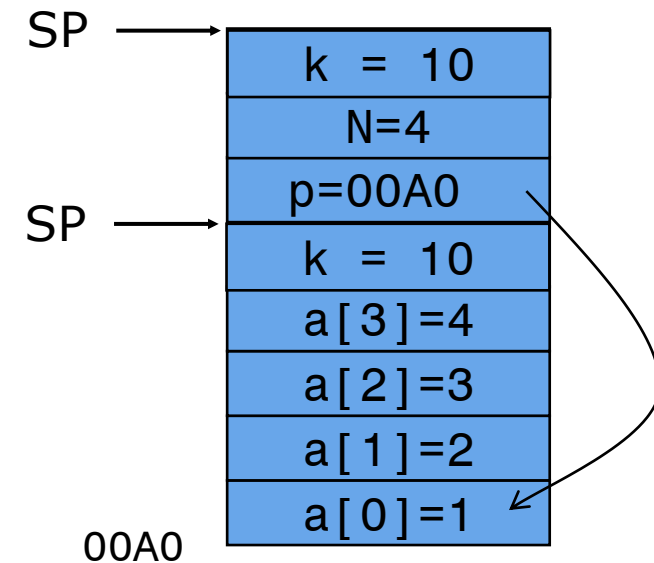
```
int Tong_mang(int* p, int N) {
```

```
    int k = 0;
```

```
    for (int i=0; i<N; i++)
```

```
        k += *p++;
```

```
    return k; }
```



@ Truyền mảng tham số?

```
int Tong_mang(int p[4], int N);  
    // Goi ham  
void main() {  
    int a[] = {1, 2, 3, 4};  
    int k = Tong_mang(a,4);  
    ...  
}
```

Bản chất giống
truyền địa chỉ

```
// Dinh nghĩa ham  
int Tong_mang(int p[4], int N) {  
    int k = 0;  
    for (int i=0; i<N; i++)  
        k += *p++;  
    return k;  
}
```

@ Thử lại ví dụ trước

```
#include <iostream.h>
void func(int* pN) {
    *pN=15;
}

void main() {
    int x = 5;
    cout<<"Nhap so nguyen:";
    func(&x);
    cout << "Bay gio x la " << x;
    ...
}
```

❖ Kết quả ???



Khi nào sử dụng truyền địa chỉ(con trỏ)?

- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Khi kích cỡ kiểu dữ liệu lớn => tránh sao chép dữ liệu vào ngăn xếp
- Truyền tham số là một mảng => bắt buộc truyền địa chỉ
- Lưu ý: Sử dụng con trỏ để truyền địa chỉ của vùng nhớ dữ liệu đầu vào. Bản thân con trỏ có thể thay đổi được trong hàm nhưng địa chỉ vùng nhớ không thay đổi (nội dung của vùng nhớ đó thay đổi được)



Truyền tham chiếu (C++)

```
#include <iostream.h>
void func(int& N) {
    N=15;
}

void main() {
    int x = 5;
    cout<<"Nhap so nguyen:";
    func(x);
    cout << "Bay gio x la " << x;
    ...
}
❖ Kết quả: ???
```


@ Ví dụ hàm hoán vị

- Viết một hàm hoán đổi giá trị của hai phần tử nguyên, sau đó viết chương trình ứng dụng hàm đó.

```
#include <iostream.h>
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
void main() {
    int x, y;
    cout << "nhap vao 2 so nguyen";
    cin >>x >>y;
    swap(x,y);
    cout << "Now x is " << x << ", y is " << y;
}
```

- Còn cách nào giải quyết bài toán trên?

@ Ví dụ hàm swap

- Hàm truyền địa chỉ

```
#include <iostream>
using namespace std;
void swap1(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void main() {
    int x, y;
    cout << "nhap vao 2 so nguyen";
    cin >> x >> y;
    swap1(&x, &y);
    cout << "Now x is " << x << ", y
is " << y;
}
```

Watch 1		
Name	Value	Type
&x	0x0012ff60	int *
	1	int
&y	0x0012ff54	int *
	2	int
&a	0x0012fe7c	int **
	0x0012ff60	int *
	1	int
&b	0x0012fe80	int **
	0x0012ff54	int *
	2	int



Khi nào sử dụng truyền tham chiếu?

- Truyền tham chiếu chỉ dùng trong C++
- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Một tham số tham chiếu có thể đóng vai trò là đầu ra (chứa kết quả), hoặc có thể vừa là đầu vào và đầu ra
- Khi kích cỡ kiểu dữ liệu lớn => tránh sao chép dữ liệu vào ngăn xếp.



Kiểu trả về

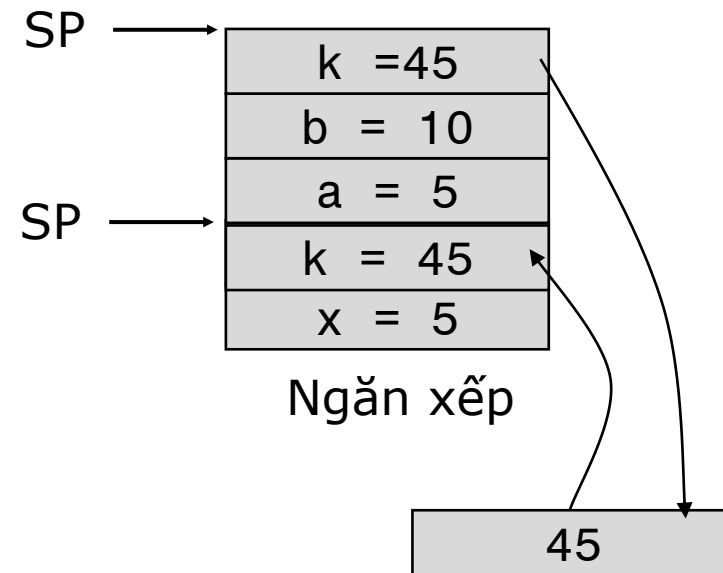
- Kiểu trả về gần như tùy ý.
- Không thể trả về trực tiếp một mảng
- Có thể trả về kiểu:
 - ✓ Giá trị
 - ✓ Con trỏ (địa chỉ)
 - ✓ Tham chiếu
- Tuy nhiên, cần rất thận trọng với trả về địa chỉ hoặc tham chiếu:
 - ✓ Không bao giờ trả về con trỏ hoặc tham chiếu vào biến cục bộ
 - ✓ Không bao giờ trả về con trỏ hoặc tham chiếu vào tham số truyền qua giá trị



Cơ chế trả về

```
int Tinh_tong(int a, int b) {  
    int k = 0;  
    for (int i=a; i <= b; ++i)  
        k +=i;  
    return k;  
}
```

```
void main() {  
    int x = 5, k = 0;  
    k = Tinh_tong(x,10);  
    ...  
}
```





Trả về con trỏ

- Viết hàm tìm giá trị lớn nhất của một mảng?
- Viết hàm trả về địa chỉ của phần tử lớn nhất trong một mảng và giá trị của phần tử đó?
- Viết chương trình ứng dụng các hàm trên.

//Hàm trả về giá trị lớn nhất của mảng

```
void Max(int* p, int n, int& max) {  
    int *p2 = p + n; max=*p;  
    while (p < p2) {  
        if (*p > max)  
            max = *p;  
        ++p;  
    }  
}
```

//Hàm trả về giá trị lớn nhất của mảng và địa chỉ của phần tử lớn nhất đó

```
int* Max(int* p, int n, int& max)  
{  
    int *pMax = p; int *p2 = p + n;  
    while (p < p2) {  
        if (*p > *pMax)  
            pMax = p;  
        ++p;  
    }  
    max=*pMax;  
    return pMax;  
}  
  
void main() {  
    int s[5] = { 1, 2, 3, 4, 5};  
    int *p = Max(s,5);  
}
```



Lý do trả về con trỏ hoặc tham chiếu

- Tương tự như lý do truyền địa chỉ hoặc truyền tham chiếu:
- Tránh sao chép dữ liệu lớn không cần thiết
- Để có thể truy cập trực tiếp và thay đổi giá trị đầu ra
- Có thể trả về con trỏ hoặc tham chiếu vào đâu?
- Vào biến toàn cục
- Vào tham số truyền cho hàm qua địa chỉ hoặc qua tham chiếu
- Nói chung: vào vùng nhớ mà còn tiếp tục tồn tại sau khi kết thúc hàm



Tham số mặc định của hàm

- VD:

```
void func(int a, int b=0, int c=0)
{ ..... }
void main()
{
    int x=4, y=5, z=6;
    func(x,y,z);//OK
    func(x,y);//OK
    func(x);//OK
}
```


@ Tham số mặc định của hàm (...)

- Các tham số mặc phải nằm ở cuối cùng của dãy tham số (tính từ trái sang phải)
- Tham số thiếu vắng trong lời gọi hàm sẽ tương ứng với các tham số mặc định cuối cùng (tính từ trái sang phải)
- Đối với một hàm có tiền khai báo và định nghĩa hàm, đối số mặc định có thể được khai báo ở một trong hai, nhưng không phải cả hai.

```
int sum(int a, int b, int c = 0);  
int sum(int a, int b, int c)  
{  
    return a + b + c;  
}
```

Hoặc

```
int sum(int a, int b, int c);  
int sum(int a, int b, int c = 0)  
{  
    return a + b + c;  
}
```



Ví dụ tìm min, max của mảng

- Viết một hàm đồng thời tìm giá trị lớn nhất và giá trị nhỏ nhất của một mảng. Viết một chương trình ứng dụng hàm đó?



Ví dụ tìm min, max của mảng

```
#include <iostream>
using namespace std;
void min_max(int *p, int N, int& max, int& min)
{
    int* p2=p+N;
    max=p[0];
    min = p[0];
    while (p<p2){
        if (max<*p)
            max=*p;
        if(min>*p)
            min = *p;
        p++;
    }
}
```

```
void main()
{
    //const int N=7;
    int a[]={1,8,9,5,7,99,-9};
    int min,max;
    min_max(a, 7, max, min);
    cout<<"min = "<<min<<" max = "
    <<max<<endl;
}
```



Nạp chồng tên hàm trong C++

- Trong C++ có thể xây dựng nhiều hàm có cùng tên, ví dụ:

```
int    max(int a, int b);  
double max(double a, double b);  
double max(double a, double b, double c);  
double max(double *seq, int n);
```

- Mục đích của nạp chồng tên hàm:
 - ✓ Đơn giản hóa cho người xây dựng hàm trong việc chọn tên (thay vì maxInt, maxDouble, maxDouble3, maxDoubleSequence,...)
 - ✓ Đơn giản hóa cho người sử dụng hàm, chỉ cần nhớ 1 tên quen thuộc thay cho nhiều tên phức tạp



Ví dụ: định nghĩa các hàm max()

```
int max(int a, int b) {                // (1)
    return (a > b)? a : b;
}
```

```
double max(double a, double b) {      // (2)
    return (a > b)? a : b;
}
```

```
double max(double a, double b, double c); { // (3)
    if (a < b) a = b;
    if (a < c) a = c;
    return a;
}
```

```
double max(double *seq, int n) {      // (4)
    int i = 0, kq = seq[0];
    while (i < n) {
        if (kq < seq[i]) kq = seq[i];
        ++i;
    }
    return kq;
}
```



Ví dụ: sử dụng các hàm max()

```
int max(int a, int b);           // (1)
double max(double a, double b); // (2)
double max(double a, double b, double c); // (3)
double max(double *seq, int n); // (4)
```

```
void main() {
    int k = max(5,7);           // call?
    double d = max(5.0,7.0);    // call?
    double a[] = {1,2,3,4,5,6};
    d = max(d, a[1], a[2]);    // call?
    d = max(a, 5);             // call?
    d = max(5,7);              // call?
    d = max(d, 5);             // call?
}
```

- Đây trách nhiệm kiểm tra và tìm hàm phù hợp cho compiler!



Một số qui tắc về nạp chồng tên hàm

- Các hàm cùng tên được định nghĩa cùng trong một file/ trong một thư viện hoặc sử dụng trong cùng một chương trình phải khác nhau ít nhất về:
 - ✓ Số lượng các tham số, hoặc
 - ✓ Kiểu của ít nhất một tham số (int khác short, const int khác int, int khác int&, ...)
- Không thể chỉ khác nhau ở kiểu trả về
- Tại sao vậy?
 - ✓ Compiler cần có cơ sở để quyết định gọi hàm nào
 - ✓ Dựa vào cú pháp trong lời gọi (số lượng và kiểu các tham số thực tế) compiler sẽ chọn hàm có cú pháp phù hợp nhất
 - ✓ Khi cần compiler có thể tự động chuyển đổi kiểu theo chiều hướng hợp lý nhất (vd short=>int, int => double)

@ Một số qui tắc về nạp chồng tên hàm

- Hàm có đối số mặc định có thể được nạp chồng.

Ví dụ:

```
void print(int a);  
void print(double a = 0);
```

- Các tham số có đối số mặc định không được sử dụng để xác định tính duy nhất trong nạp chồng hàm.

Ví dụ:

```
void print(int a);  
void print(int a, int b = 0); // lỗi
```




Hàm nội tuyến (inline) trong C++

- Vấn đề: Hàm tiện dụng, nhưng nhiều khi hiệu suất không cao, đặc biệt khi mã thực thi hàm ngắn vì
 - ✓ Các thủ tục như nhớ lại trạng thái chương trình, cấp phát bộ nhớ ngăn xếp, sao chép tham số, sao chép giá trị trả về, khôi phục trạng thái chương trình mất nhiều thời gian
 - ✓ Nếu mã thực thi hàm ngắn thì sự tiện dụng không bù so với sự lãng phí thời gian



Dùng hàm inline trong C++

- Điều duy nhất cần làm là thêm từ khóa inline vào đầu dòng khai báo và định nghĩa hàm

```
inline int max(int a, int b) {  
    return (a > b)? a : b;  
}
```

- Hàm inline khác gì hàm bình thường?
 - ✓ "Hàm inline" thực chất không phải là một hàm!
 - ✓ Khi gọi hàm thì lời gọi hàm được thay thế một cách thông minh bởi mã nguồn định nghĩa hàm, không thực hiện các thủ tục gọi hàm



Khi nào nên dùng hàm inline

- Ưu điểm của hàm inline:
 - ✓ Tiện dụng như hàm bình thường
 - ✓ Hiệu suất như viết thẳng mã, không gọi hàm
 - ✓ Tin cậy, an toàn.
- Nhược điểm của hàm inline:
 - ✓ Nếu gọi hàm nhiều lần trong chương trình, mã chương trình có thể lớn lên nhiều (mã thực hiện hàm xuất hiện nhiều lần trong chương trình)
 - ✓ Mã định nghĩa hàm phải để mở => đưa trong header file
- Lựa chọn xây dựng và sử dụng hàm inline khi:
 - ✓ Mã định nghĩa hàm nhỏ (một vài dòng lệnh, không chứa vòng lặp)
 - ✓ Yêu cầu về tốc độ đặt ra trước dung lượng bộ nhớ



@ Ví dụ

- Yêu cầu bài toán: Tính tổng một dãy số nguyên (liên tục) trong phạm vi do người sử dụng nhập. In kết quả ra màn hình. (sử dụng hàm inline)