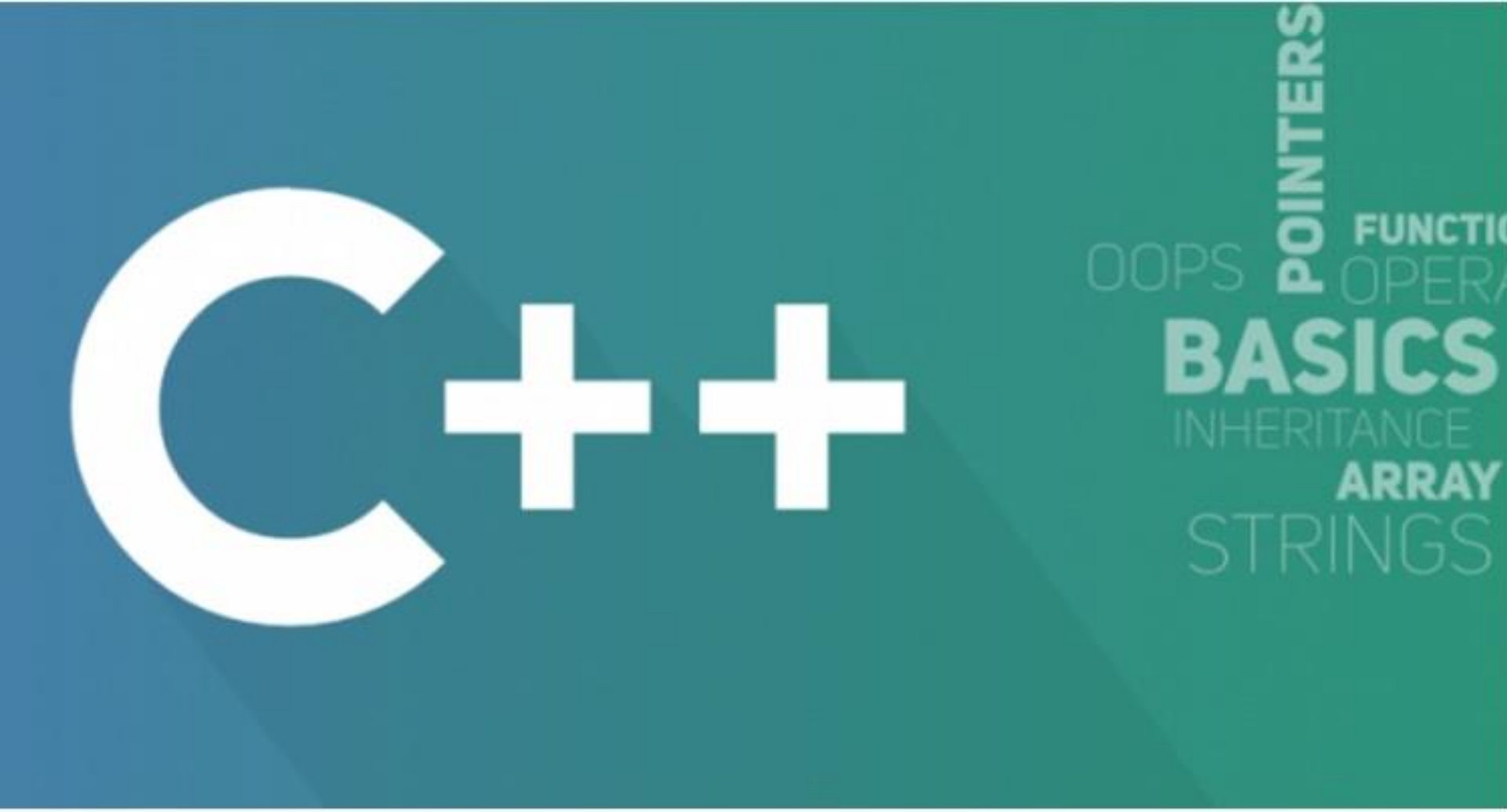

Bộ môn Kỹ thuật hệ thống và Mạng máy tính

Khoa Công nghệ thông tin

Chương 1. Cơ sở lập trình C++

1.1. Các thành phần cơ bản của C++

A. Tập ký tự

- Bộ ký tự bao gồm:
  - Các kí tự (viết thường và viết hoa) của bảng chữ cái tiếng Anh:
    a b c d …. x y z A B C D …. X Y Z
  - Các chữ số: 0 1 2 … 8 9
  - Các dấu và kí tự đặc biệt: + - * / % = ~ ! # % ^ & ( ) { } [ ] : . , ? \ ‘ “ _ ; |

- Phân biệt chữ hoa và chữ thường.
- Dấu ; dùng để kết thúc câu lệnh.
- { } bắt đầu và kết thúc khối chương trình.
- Dấu ‘ bắt đầu và kết thúc hằng ký tự.
- Dấu “ bắt đầu và kết thúc hằng xâu ký tự.

B. Từ khóa

- Là từ được quy định sẵn trong ngôn ngữ, người lập trình phải thực hiện đúng chính tả và ngữ pháp của từ khóa.
- Một số ví dụ từ khóa của C++
  - **char, int, float, double, struct, signed, unsigned, short, long**
  - **if, else, for, while, do, switch, case, break, continue, return**
  - **using, namespace, include, define**

C. Tên

- Tên là dãy ký tự dùng để định danh cho một phần tử trong một chương trình như kiểu dữ liệu, biến, hằng, hàm.
- Qui tắc đặt tên:
  - Tên dài không quá 255 kí tự
  - Chỉ bao gồm chữ cái, chữ số và dấu _
  - Phải bắt đầu bằng chữ cái hoặc _
  - Tên không được trùng với từ khóa
  - Trong cùng một phạm vi của chương trình, không được phép có đại lượng trùng tên.
- Ví dụ:

  i, x, y, a, b, _function, _MY_CONSTANT, PI, gia_tri_1

Tên (tiếp)

- Cách thức đặt định danh/tên:
  - Hằng số: chữ hoa
  - Các biến, hàm, kiểu dữ liệu : Bằng chữ thường.
  - Nếu tên gồm nhiều từ thì nên phân cách các từ bằng dấu gạch dưới.
- *Ví dụ*:

Định danh	Loại đối tượng
HANG_SO_1, _CONSTANT_2	hằng
a, b, i, j, count	biến
nhap_du_lieu, tim_kiem, xu_li	hàm
sinh_vien, mat_hang	cấu trúc

D. Lời giải thích (comment)

- Là những giải thích ngắn gọn về chương trình hoặc 1 phần của chương trình.
- Cách biểu diễn:
  - Giải thích trên một dòng: *// Dòng chú thích*
  - Giải thích bằng cả một đoạn

```
/*  Dòng giải thích 1
    Dòng giải thích 2
    Dòng giải thích 3  */
```

E. Các chỉ thị tiền xử lý

- Là bộ xử lý văn bản, làm việc trên văn bản chương trình nguồn như bước đầu của quá trình biên dịch.
- Thường được sử dụng để thay đổi chương trình hoặc dịch chương trình một cách dễ dàng tùy theo môi trường thực hiện khác nhau.
- Bắt đầu bằng dấu #, có thể xuất hiện ở bất cứ đâu trong chương trình.
- Cuối các chỉ thị tiền xử lý không có dấu ;

E. Các chỉ thị tiền xử lý

a) **Chỉ thị gộp, ghép, bao hàm:** Bổ sung các tệp khác vào chương trình nguồn tại vị trí của chỉ thị. Trong những tệp này có thể chứa các chỉ thị khác của bộ tiền xử lý, các chỉ thị đó sẽ được xử lý trước khi tiếp tục xử lý văn bản của tệp nguồn. Tác dụng là để ghép nối nhiều tệp và thư viện vào chương trình.

#include “[đường dẫn\]tên tệp”

#include <[đường dẫn\]tên tệp>

Khi không có [đường dẫn\] hai cách viết khác nhau:

- Cách 1 tìm tệp trong thư mục hiện hành trước rồi mới tìm trong thư mục INCLUDE của C++.
- Cách 2 chỉ tìm tệp trong thư mục INCLUDE của C++.

E. Các chỉ thị tiền xử lý

b) Chỉ thị định nghĩa hàm và macro

- Thay thế đơn giản:

#define Tên Giá_trị

Thay thế tên bằng giá trị đằng sau nó.

Giá trị có thể là hàm, hằng (không thực hiện với các hằng xâu ký tự), biểu thức, một đoạn chương trình nằm giữa { }

- Định nghĩa macro (có đối số):

#define Tên_macro(danh sách đối) Biểu thức thay thế

- Chấm dứt tác dụng của thay thế hoặc định nghĩa macro:

#undef Tên

#undef Tên_macro

E. Các chỉ thị tiền xử lý

c) **Phân vùng khi dịch chương trình:** Các chỉ thị biên dịch có điều kiện:

```
#if điều kiện
Đoạn chương trình
#endif
```

Hoặc

```
#if điều kiện
Đoạn chương trình 1
#else
Đoạn chương trình 2
#endif
```

d) Chỉ thị báo lỗi

#error thông_báo

Chỉ thị này sẽ dừng chương trình dịch khi có lỗi và in ra một thông báo lỗi.

F. Cấu trúc chương trình C/C++

#include <tên thư viện> //Khai báo tệp tiêu đề của thư viện
using namespace … //Khai báo không gian tên hàm
Khai báo kiểu dữ liệu người lập trình tự tạo (nếu có)
Khai báo hàm nguyên mẫu (nếu có)
Khai báo các hằng và biến dữ liệu toàn cục

int main( ) //Chương trình chính
{
các câu lệnh;
return 0;
}

Nội dung các hàm đã khai báo ở trên

1.2 Các kiểu dữ liệu chuẩn

Loại dữ liệu	Kiểu dữ liệu	Số ô nhớ	Miền giá trị
Boolean	bool	1 byte	0 hoặc 1. Trong đó 0 => false và 1 => true
Ký tự	char	1 byte	-128 tới 127 hoặc 0 tới 255
	unsigned char	1 byte	0 tới 255
	signed char	1 byte	-128 tới 127
Số nguyên	int	4 byte	-2147483648 tới 2147483647
	unsigned int	4 byte	0 tới 4294967295
	signed int	4 byte	-2147483648 tới 2147483647
	short int	2 byte	-32768 tới 32767
	long int	4 byte	-2,147,483,648 tới 2,147,483,647
Số thực	float	4 byte	3.4e -38 tới 3.4e+38 (~7 chữ số)
	double	8 byte	1.7e – 308 tới 1.7e+308 (~15 chữ số)
	long double	8 byte	1.7e – 308 tới 1.7e+308 (~15 chữ số)

Cách khai báo biến kiểu dữ liệu chuẩn

- Khai báo từng biến: ***<tên kiểu> <tên biến>;***

```
float x;      // biến kiểu thực float
double z;     // biến kiểu thực double
int i;        // biến kiểu nguyên int
```

- Khai báo một loạt biến: ***<tên kiểu> <danh sách tên biến>;***

```
float x,y,z;  // danh sách 3 biến kiểu float
int i,j,k;    // danh sách 3 biến kiểu int
```

- Khai báo và khởi tạo giá trị ban đầu:

<tên kiểu> <tên biến>=<giá trị>;

```
int a = 3;
float x = 5.0, y = 2.6;
```

Cách khai báo hằng dữ liệu

- Khai báo bằng chỉ thị:

#define <tên hằng> <giá trị>

```
#define MAX_SINH_VIEN  60          // hằng kiểu số nguyên
#define DIEM_CHUAN  23.5           // hằng kiểu số thực
```

- Khai báo với từ khóa:

const <tên kiểu> <tên hằng> = <giá trị>;

```
const int MAX_SINH_VIEN = 60;              // hằng kiểu số nguyên
const float DIEM_CHUAN = 23.5;             // hằng kiểu số thực
```

Biểu diễn giá trị hằng

Hằng nguyên

Hệ cơ số 10	Hệ cơ số 8	Hệ cơ số 16
2007	03727	0x7D7
396	0614	0x18C

Hằng kí tự

Kí tự cần biểu diễn	Cách 1 (kí tự)	Cách 2 (kiểu số nguyên –hệ 10)
Chữ cái A	'A'	(char) 65
Dấu nháy đơn '	'\''	(char)39
Dấu nháy kép "	'\"'	(char)34
Dấu \	'\\'	(char)92
Kí tự xuống dòng	'\n'	(char)10
Kí tự NULL	'\0'	(char)0
Kí tự Tab	'\t'	(char)9

Hằng thực

Số thực dấu phẩy tĩnh	Số thực dấu phẩy động
3.14159	31.4159 E-1
123.456	12.3456 E+1 hoặc 1.23456 E+2

00	NULL	(Null character)	32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
04	EOT	(End of Trans.)	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
08	BS	(Backspace)	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
09	HT	(Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	FF	(Form feed)	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	CR	(Carriage return)	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	SI	(Shift In)	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	FS	(File separator)	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	GS	(Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	US	(Unit separator)	63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp
127	DEL	(Delete)														

1.3. Biểu thức

- Biểu thức bao gồm các toán hạng kết hợp với nhau qua các toán tử
- Toán hạng có thể là các đại lượng biến, hằng, hàm
- Toán tử là các phép toán
- Các dạng biểu thức cơ bản
  - Biểu thức số học,
  - Biểu thức logic
  - Biểu thức quan hệ so sánh
- Một toán tử khác của C++
  - Toán tử với bit
  - Toán tử gán
  - Toán tử điều kiện

A. Biểu thức số học

- Toán tử: *các phép toán số học*

  `+, -, *, /, %`
- Toán hạng: *các kiểu dữ liệu số*
- Chú ý:
  - Phép chia / cho kết quả số thập phân nếu một trong các toán hạng có kiểu số thực
  - Phép chia / cho kết quả là phần nguyên nếu các toán hạng có kiểu số nguyên
- VD về phép chia

```
int a = 5;
int b;
float c;
b = a/2;       // kết quả b = 2
c = a/2.0;     //  kết quả c = 2.5
```

B. Biểu thức logic

- Giá trị của biểu thức logic là true (đúng) và false (sai)
- Toán hạng của biểu thức logic là các đại lượng có kiểu dữ liệu logic (bool), các mệnh đề quan hệ so sánh
- Toán tử là các phép toán logic :
  - Phép VÀ : &&
  - Phép HOẶC: ||
  - Phép PHỦ ĐỊNH: !

A	B	!A	A && B	A || B
False (0)	False (0)	True (1)	False (0)	False (0)
False (0)	True (1)	True (1)	False (0)	True (1)
True (1)	False (0)	False (0)	False (0)	True (1)
True (1)	True (1)	False (0)	True (1)	True (1)

C. Biểu thức quan hệ

- Biểu thức quan hệ gồm 2 toán hạng (gọi là mệnh đề so sánh) và phép toán quan hệ. Giá trị của mệnh đề so sánh là true (đúng) hoặc false (sai)
- Các phép toán so sánh : > , < , >= , <=, ==, !=
- Ví dụ
  - `Mệnh đề  5 > 7  có giá trị  là false`
  - `Mệnh đề  9 <= 10 có giá trị là true`
- Mệnh đề quan hệ so sánh và biểu thức logic

  `VD:  (b<=2) || (a>1)`

D. Làm việc với bit

Toán tử	Kí hiệu
Phép AND	&
Phép OR	|
Phép phủ định NOT	~
Phép XOR	^
Phép dịch trái - Shift left	<<
Phép dịch phải - Shift right	>>

D. Làm việc với bit

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Ví dụ:

$19_{(10)} = 10011_{(2)}$

$25_{(10)} = 11001_{(2)}$

$19 \,\&\, 25 = 10001_{(2)} = 17_{(10)}$

$19 \mid 25 = 11011_{(2)} = 27_{(10)}$

$19 \,\hat{}\, 25 = 01010_{(2)} = 10_{(10)}$

D. Làm việc với bit

- Toán tử dịch trái << và dịch phải >> :
  - A << n: dãy bit A dịch sang trái n bit (dãy bit A bỏ đi n bit tận cùng bên trái và thêm n bit 0 vào tận cùng bên phải).
    - Ví dụ: unsigned char a = 5; // $00000101_{(2)}$ unsigned char b = a << 4; // $01010000_{(2)} = 80_{(10)}$
  - A >> n: dãy bit A dịch sang phải n bit (dãy bit A bỏ đi n bit tận cùng bên phải và thêm n bit 0 vào tận cùng bên trái).
    - Ví dụ: unsigned char b = a >> 1; // $00000010_{(2)} = 2_{(10)}$

D. Làm việc với bit

- Ứng dụng: Tăng tốc khi viết các trình điều khiển thiết bị (drivers), đồ họa, giao thức truyền thông và giải mã.
- Ví dụ : Thanh ghi cờ 8 bit ứng với 8 cờ báo khác nhau. Làm thế nào để nhận ra 1 cờ báo (ví dụ cờ ở bit số 3) có giá trị =1?

E. Toán tử gán

- Phép gán `<tên_biến> = <biểu_thức>;`
- Tác dụng:
  - Đặt giá trị của biến bằng giá trị của biểu thức

```
int a,b,c,d;
    a = 1 ;
    b = 2 ;
    c = 3 ;
    d = (a+b+c)/2;
```

  - Tạo nên biểu thức gán: Giá trị của biểu thức gán bằng giá trị của biến sau khi phép gán được thực hiện
  - Toán tử gán có thể sử dụng trong các biểu thức và các câu lệnh như các toán tử khác.

    a = b = 5;

    z = (y = 2)*(x = 6);

Phép gán thu gọn

- Rút gọn phép gán `x = x + y` thành `x += y;`
- Áp dụng cho các phép toán số học : +, -, *, /, %, <<, >>, &, |, ^
- Ví dụ:

```
x -= 5;          // x = x - 5;
y *= b;          // y = y * b;
z /= 2;          // z = z / 2;
```

F. Toán tử điều kiện

- Cú pháp:

  `<biểu thức logic> ? <biểu thức 1>:<biểu thức 2>`

- Ý nghĩa:
  - Nếu `<biểu thức logic>` là Đúng thì giá trị của biểu thức điều kiện = giá trị của `<biểu thức 1>`
  - Ngược lại, thì = giá trị `<biểu thức 2>`
- Toán tử điều kiện "**?:**" là **toán tử 3 ngôi duy nhất** trong C++ (vì nó chứa 3 toán hạng).
- Ví dụ biểu thức điều kiện để tính trị tuyệt đối của a:

```
a>=0 ?  a : -a
```

G. Phép toán 1 ngôi

- Phép toán 1 ngôi là phép toán kết hợp với 1 toán hạng
  - VD: phép phủ định !a , phép đảo dấu -a
- Một số phép toán 1 ngôi khác:
  - Phép tăng 1 đơn vị: `i++` hoặc `++i` tương đương với i = i + 1

    `i++` (tăng sau/hậu tố); `++i` (tăng trước/tiền tố)
  - Phép giảm 1 đơn vị:

    `i--` hoặc `--i tương đương với i= i-1`

    `i--` (giảm sau/hậu tố); `--i` (giảm trước/tiền tố)
  - sizeof: cung cấp thông tin về số byte bộ nhớ mà đối số của nó chiếm (hoặc có thể chiếm):

sizeof(kiểu dữ liệu) Ví dụ: sizeof(char);

sizeof biểu thức Ví dụ: int a[10]; sizeof a;

G. Phép toán 1 ngôi

- Một số phép toán 1 ngôi khác:
  - Ép kiểu: làm một kiểu dữ liệu này biến đổi thành kiểu dữ liệu khác:
    (kieu_du_lieu) bieu_thuc
  - Ví dụ: double a = 15.65653;

```
int c ;
c = (int) a; // c=15
```

Ngôn ngữ **C++ cho phép** thực hiện ép kiểu tường minh C-style với **cú pháp như một lời gọi hàm**:

```
c= int (a);
```

  - Phân loại:
    1. **Nới rộng (widening):** Là quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin.
    2. **Thu hẹp (narrowwing):** Là quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin

G. Phép toán 1 ngôi

- Một số phép toán 1 ngôi khác:
  - Ép kiểu (tiếp): Ép kiểu tường minh không được trình biên dịch (compiler) kiểm tra tại thời điểm biên dịch, nên sẽ không đưa ra những cảnh báo trong những trường hợp chuyển đổi không đúng $\Rightarrow$ một số toán tử ép kiểu được hỗ trợ bởi C++:

    static_cast
    const_cast
    reinterpret_cast
    dynamic_cast
  - static_cast: yêu cầu compiler kiểm tra kiểu dữ liệu tại thời điểm biên dịch chương trình, hạn chế được những lỗi ngoài ý muốn.

    static_cast<kiểu dữ liệu>(biểu thức)
  - Ví dụ: c = static_cast<int>(a);

H. Thứ tự ưu tiên thực hiện phép toán

Mức ưu tiên	Nhóm toán tử	Toán tử	Thứ tự thực hiện
1	Trong (…)		
2	Phạm vi	::	Trái sang phải
3	Hậu tố (một ngôi)	++, --, [], (), ., ->	Trái sang phải
4	Tiền tố (một ngôi)	++,--, !,~,sizeof, -, +, *, &	Phải sang trái
5	Số học	*, /, %	Trái sang phải
6	Số học	+, -	Trái sang phải
7	Dịch bit	<<, >>	Trái sang phải
8	So sánh	>, <, >=, <=	Trái sang phải
9	So sánh	==, !=	Trái sang phải
10	AND bit	&	Trái sang phải
11	XOR bit	^	Trái sang phải
12	OR bit	|	Trái sang phải

H. Thứ tự ưu tiên thực hiện phép toán

Mức ưu tiên	Nhóm toán tử	Toán tử	Thứ tự thực hiện
13	Logic	&&	Trái sang phải
14	Logic	||	Trái sang phải
15	Điều kiện	?:	Phải sang trái
16	Gán	=, *=,/=,%=,+=,-=,<<=, >>=, &=,^=,|=	Phải sang trái
17	Dấu phẩy	,	Trái sang phải

Ví dụ: Thực hiện từ phải sang trái:

```
int a, b, c = 2, i = 0;
a = b = c *= i += 3;
```

3

6

6

6

H. Thứ tự ưu tiên thực hiện phép toán

1) Các phép toán đặt trong cặp () thực hiện trước
2) Phép toán 1 ngôi: Nếu các phép toán cùng mức hậu tố thực hiện từ trái qua phải, tiền tố phải qua trái
3) Phép toán 2 ngôi: nếu 2 phép toán cùng mức thì ưu tiên thực hiện từ trái qua phải
   - Phép số học: *, /, %
   - Phép số học: +, -
   - Phép dịch bit
   - Phép so sánh lớn hơn nhỏ hơn: >, <, >=, <=,
   - Phép so sánh bằng: ==, !=
   - Phép AND bit
   - Phép XOR bit
   - Phép OR bit
   - Phép logic AND: &&
   - Phép logic OR: ||
4) Phép toán 3 ngôi: Nếu các phép toán cùng mức thì thực hiện từ phải qua trái.
5) Phép gán: Nếu các phép toán cùng mức thì thực hiện từ phải qua trái.
6) Dấu phẩy

I. Các hàm toán học của C++

- Để sử dụng cần có chỉ thị #include <math.h>
- Các hàm:
  - log(x) : tính loga cơ số tự nhiên của x
  - log10(x) : tính loga cơ số 10 của x
  - exp(x): tính e^x
  - tan(x): tính tan(x)
  - sin(x)
  - cos(x)
  - pow(a,x): tính a^x
  - sqrt(x) : căn bậc 2 của x
  - fabs(x): trị tuyệt đối của x

1.4. Nhập dữ liệu từ bàn phím bằng **cin >>**

- cin : là đối tượng chuẩn của C++ chuyên dùng để nhập dữ liệu cho biến từ bàn phím
- Khi dùng cần khai báo thư viện `#include <iostream>`
  và không gian tên std: using namespace std;
- Cú pháp cơ bản: `cin >> tên_biến;`

```
VD:   int a;
      cin >> a;  // nhap du lieu bien a
```

- Cách nhập dữ liệu cho nhiều biến liên tiếp

```
VD: float a, b;
    int  c;
    cin >> a >> b >> c; // nhap a, b, c
```

1.5. Xuất dữ liệu ra màn hình bằng cout <<

- cout : là đối tượng chuẩn của C++ chuyên dùng để xuất dữ liệu cho biến ra màn hình.
- Khi dùng cần khai báo thư viện `#include <iostream>`

  và không gian tên std: using namespace std;
- Xuất giá trị của biến: `cout << tên_biến ;`

```
VD:   int a = 2;
      cout << a;  // xuat du lieu bien a
```

- Có thể xuất dữ liệu cho nhiều biến liên tiếp

```
VD: float a, b;
    int  c;
    cout << a << b << c; // xuất a, b, c
```

Xuất giá trị của hằng, hàm

- Xuất giá trị của hằng: `cout << hằng`

  VD:
  ```
  const float PI = 3.14;
  cout << "PI=" << PI << endl;
  ```

- Xuất giá trị của hàm: `cout << hàm`

  VD: Xuất giá trị hàm toán học

```
#include <math.h>
#include <iostream>
…
const float PI= 3.14;
cout << sin(PI);
```

Cách căn chỉnh định dạng khi in ra màn hình

- **#include <iomanip>**
- **endl** hoặc **'\n'** xuống dòng
- **left** căn lề trái
- **right** căn lề phải
- **setw(n)** đặt độ rộng n vị trí khi in dữ liệu ra màn hình
- **setprecision(n):** in ra n chữ số khi in một số thực
- Nếu **setprecision(n)** kèm theo **fixed:** in số thực dấu chấm tĩnh với đủ n chữ số của phần thập phân
- Nếu **setprecision(n)** kèm theo **scientific:** in số thực dấu chấm động với đủ n chữ số của phần thập phân
- **showpoint** : hiển thị phần thập phân của số thực
- **boolalpha** : hiển thị true, false thay vì 1 (true), 0 (false)

NGÔN NGỮ LẬP TRÌNH C++

1.6. CÁC LỆNH CƠ BẢN

A. Lệnh khối/ghép

- Gồm một dãy câu lệnh tuần tự nối tiếp nhau trong một cặp ngoặc { }
- Cú pháp:

```
{
    câu lệnh 1;
   câu lệnh 2;
   …
   câu lệnh n;
 }
```

- Tác dụng: Tạo luồng điều khiển tuần tự
- Các lệnh khối có thể lồng nhau
- C và C++ không giới hạn số lệnh bên trong khối và số lệnh khối lồng nhau

B. Lệnh rẽ nhánh **if…else**

- Điều khiển rẽ nhánh theo giá trị true hay false của biểu thức điều kiện
- Cú pháp 1: `if(biểu_thức) việc;`
- Cú pháp 2:
```
if(biểu_thức) việc_1;
        else việc_2;
```
- Chú ý:
  - `biểu_thức` phải có kiểu số nguyên hoặc logic và đặt trong cặp ngoặc ( ).
  - Các lệnh if … else có thể lồng nhau.
  - Trong lệnh *if…else* lồng nhau, mệnh đề *else* đi cùng mệnh đề *if* ngay trước nó ➡Nên dùng lệnh khối/ghép để tránh nhầm lẫn.
  - Nếu việc gồm nhiều lệnh phải dùng lệnh khối/ghép.

C. Lệnh lựa chọn switch…case

- Lựa chọn rẽ nhánh theo giá trị số nguyên của biểu thức điều kiện
- Cú pháp:

```
switch(biểu_thức)
{
    case giá_trị_1:  việc_1; break;
    case giá_trị_2:  việc_2; break;
    …
    case giá_trị_n:  việc_n; break;

    default: lệnh_n+1; break; // có thể không có
}
```

Trình tự thực hiện lệnh switch … case

1) Tính giá trị của `biểu_thức` (kiểu số nguyên)
2) Thực hiện rẽ nhánh
   - Nếu `biểu_thức == giá_trị_1`: chuyển đến lệnh 1
   - …
   - Nếu `biểu_thức == giá_trị_n`: chuyển đến lệnh n
   - Nếu `biểu_thức` có giá trị khác: chuyển đến `default`

Chú ý với lệnh break;

- Lệnh **break** dùng để ngắt luồng điều khiển và thoát ra ngoài cấu trúc lệnh **switch**.
- Nếu thiếu **break**, luồng điều khiển sẽ thực hiện các lệnh tiếp bên trong cấu trúc.

Chú ý với mệnh đề **case**

- Sau mệnh đề **case … :** có thể đặt nhiều câu lệnh mà không cần lệnh khối/ghép.

```
case giá_trị:
    lệnh 1;
    lệnh 2;
    …
    lệnh n;
    break;
```

- Dùng lệnh khối/ghép khi muốn khai báo + khởi tạo giá trị đầu cho một biến trong **case… :**

```
case  giá_trị:
{ <kiểu dữ liệu>  tên biến = <giá trị>;
   lệnh 1;
    lệnh 2;
     …
    lệnh n;
    break;
}
```

D. Lệnh chu trình có số vòng lặp xác định **for()**

- Cú pháp:

for (biểu thức 1; biểu thức 2; biểu thức 3)

{

việc;

}

- Biểu thức 1: Khởi tạo giá trị ban đầu của biến điều khiển chu trình.
- Biểu thức 2: biểu thức điều kiện để thực hiện chu trình. Nếu == true thì tiếp tục chu trình, ==false thì ra khỏi chu trình.
- Biểu thức 3: Chuẩn bị cho bước sau: Tính giá trị tiếp theo của biến điều khiển chu trình.
- Ví dụ: (giaithua.cpp)

```
gt=1;
for(i=1; i<=N; i++)
    gt*=i;
```

E. Lệnh chu trình có số vòng lặp không xác định **while** và **do…while**

- Lệnh **while** và **do…while** tạo chu trình lặp theo giá trị true hay false của biểu thức điều kiện
- Cú pháp
  - Lệnh **while :**

```
while (biểu thức)
{
 việc
}
```

  - Lệnh **do…while**

```
do
{
việc
}
while (biểu thức);
```

Trình tự thực thi lệnh

- **while**
  - Kiểm tra giá trị biểu thức trước. Nếu true thì thực hiện lệnh. Nếu false thì thoát vòng lặp.
  - Việc đặt sau **while** có thể không thực hiện lần nào.
- **do…while**
  - Thực hiện lệnh trước rồi kiểm tra giá trị biểu thức điều kiện. Nếu true thì tiếp tục, false thì thoát.
  - Việc đặt sau **do** được thực hiện ít nhất một lần.

Chú ý

- Các chu trình có thể lồng nhau.
- Thoát khỏi chu trình bằng lệnh **break**
  - Cú pháp: `break;`
  - Đặt trong phần **việc** của chu trình
  - Tác dụng: Thoát khỏi chu trình ngay cả khi biểu thức điều kiện là true.

Chú ý (tiếp)

- Chuyển tới lần lặp tiếp theo bằng lệnh **continue**
  - Cú pháp: `continue;`
  - Đặt trong phần **việc** của chu trình.
- Từ khóa **continue** thường được sử dụng trong vòng lặp **for.**
  - 4 bước thực thi cơ bản của của 1 vòng lặp **for**:

  (1) Khởi tạo giá trị đầu tiên của biến điều khiển chu trình

  (2) Kiểm tra điều kiện để thoát khỏi chu trình

  (3) Thực hiện việc (công thức lặp)

  (4) Chuẩn bị cho bước sau
  - Nếu có **continue** trong bước (3), chương trình sẽ bỏ qua phần còn lại của bước (3) để chuyển đến thực hiện bước (4), và bắt đầu 1 lần lặp mới từ bước (2).
- Trong vòng lặp **while** và **do…while**: bỏ qua thực thi các câu lệnh nằm sau lệnh **continue** trong chu trình và chuyển về đầu chu trình để thực thi lần lặp kế tiếp

NGÔN NGỮ LẬP TRÌNH C++

1.7. CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC

A. KIỂU XÂU KÝ TỰ

- Khai báo: **string** <danh sách biến>;
- Khai báo và khởi tạo:

  string <tên biến>(“nội dung xâu”);

  string <tên biến>{ “nội dung xâu” };

  string <tên biến>=“nội dung xâu”;
- Hằng xâu ký tự: nằm giữa hai dấu “

  “”: xâu rỗng.
- Cần có: **using namespace std;**

Nhập/Xuất xâu ký tự

- **Nhập:** Sử dụng **cin >>** biến xâu;
- Nếu xâu ký tự chứa dấu cách:

  getline(**cin**, biến xâu);
- Chú ý: Khi nhập dữ liệu chuyển từ số sang chữ cần xóa bộ đệm bằng **cin.ignore();**
- **Xuất:** Sử dụng **cout << …** ;

B. KIỂU MẢNG

- Mảng là tập hợp các phần tử có cùng kiểu sắp xếp theo một trật tự nhất định.
- Khai báo mảng một chiều:

  <kiểu dữ liệu> <tên mảng> [<số phần tử tối đa>];

  – Số phần tử tối đa là hằng số.
  – Nên sử dụng chỉ thị tiền xử lý

    #define để định nghĩa <số phần tử tối đa>.

  Ví dụ: #define Max 5

  int dayso[Max];

  – Chỉ số của mảng tính từ 0 đến <tổng số phần tử> - 1.

KIỂU MẢNG MỘT CHIỀU (tiếp)

- Khởi tạo mảng một chiều:
  - Khởi tạo giá trị cho mọi phần tử của mảng:

Ví dụ: int dayso[5] = { 4 , 3 , 1 , 5 , 2};

  - Khởi tạo giá trị cho một số phần tử đầu mảng:

Ví dụ: int dayso[5] = { 4 , 3}; // 4, 3, 0, 0, 0

  - Khởi tạo giá trị 0 cho mọi phần tử của mảng:

Ví dụ: int dayso[5] = { }; // 0, 0, 0, 0, 0

  - Tự động xác định số lượng phần tử của mảng:

Ví dụ: int dayso[ ] = { 4 , 3 , 1 , 5 , 2};

KIỂU MẢNG MỘT CHIỀU (tiếp)

- Truy nhập vào mảng một chiều:

  <tên biến mảng>[chỉ số]

- Nhập mảng một chiều:

  cin >> <tên biến mảng>[chỉ số];

trong một vòng lặp **for**.

- Xuất mảng một chiều:

  cout << <tên biến mảng>[chỉ số];

trong một vòng lặp **for**.

KIỂU MẢNG 2 CHIỀU

- Khai báo mảng hai chiều:

<kiểu dữ liệu> <tên mảng>[số hàng][số cột];

- Khởi tạo mảng hai chiều:
  - Khởi tạo giá trị cho mọi phần tử của mảng:

Ví dụ: int matran[2] [3] = { {1, 2 , 3},{4 , 5 , 6} };

  - Khởi tạo giá trị cho một số phần tử đầu mảng:

Ví dụ: int matran[2] [3] = { {1}, {4 , 5, 6}};

  - Khởi tạo giá trị 0 cho mọi phần tử của mảng:

Ví dụ: int matran[2] [3] = { };

  - Tự động xác định số lượng phần tử của hàng

Ví dụ: int matran[ ][3]= { {1, 2 , 3},{4 , 5 , 6} };

KIỂU MẢNG 2 CHIỀU (tiếp)

- Truy nhập vào mảng 2 chiều:

  <tên biến mảng>[chỉ số hàng][chỉ số cột]

- Nhập mảng 2 chiều:

  cin >> <tên biến mảng>[chỉ số hàng][chỉ số cột];

trong **hai** vòng lặp **for** lồng nhau.

- Xuất mảng một chiều:

  cout << <tên biến mảng>[chỉ số hàng][chỉ số cột];

trong **hai** vòng lặp **for** lồng nhau.

C. KIỂU CẤU TRÚC

- Khai báo kiểu: **struct** <tên kiểu cấu trúc>
  {
  <kiểu dữ liệu> <thành phần 1>;
  <kiểu dữ liệu> <thành phần 2>;
  …
  <kiểu dữ liệu> <thành phần n>;
  };
- Ví dụ:
```
struct  sinhvien
{  string hoten;
   float diem[10];
   float dtb;
}  ;
```
- Chú ý: Có thể khai báo ngoài hoặc trong hàm main()

KIỂU CẤU TRÚC (tiếp)

- Khai báo biến: kiểu cấu trúc <danh sách biến>;
- Ví dụ:
```
struct  sinhvien
{       string hoten;
        float diem[10];
        float dtb;
 } sv1, sv2;
```
- hoặc: sinhvien sv1, sv2, sv3;
- Khai báo và khởi tạo:
```
sinhvien sv=
{ “Hoang Lan Anh”,
  {10, 9, 8},
  0
};
```
- Truy nhập vào thành phần cấu trúc:

  Biến cấu trúc.tên thành phần
- Ví dụ: sv1.hoten, sv2.diem[3], sv3.dtb

KIỂU CẤU TRÚC (tiếp)

STT	Ho va ten	Diem				DTB
		Mon 1	Mon2	…	Mon M	
1	Nguyen Van An	10	9		9	
2	Do Viet Binh	8	7		8	
…						
N	Hoang Hai Yen	10	9		10	

Tổ chức dữ liệu:

1) Theo mảng:

```
string hoten[60];
float diem[60][10], dtb[60];
```

2) Theo cấu trúc $\Rightarrow$ Mảng các cấu trúc

KIỂU CẤU TRÚC (tiếp)

- Mảng cấu trúc:

  <kiểu cấu trúc> tên mảng[số phần tử];
- Ví dụ: sinhvien sv[50];
- Truy nhập vào tên của sv[i]: sv[i].hoten
- Truy nhập vào điểm môn thứ j của sv[i]:

  sv[i].diem[j]
- Ví dụ: VDstruct.cpp