KỸ THUẬT LẬP TRÌNH C/C++

Dạy bởi 1 SVBK

Đại học Bách Khoa Hà Nội
Kỹ thuật Điện tử - Viễn thông

2024-25

KIỂM TRA BÀI CŨ

Câu 1: Viết chương trình C nhập và in ra chữ cái đầu tiên của tên & tuổi của bản thân.

Câu 2:

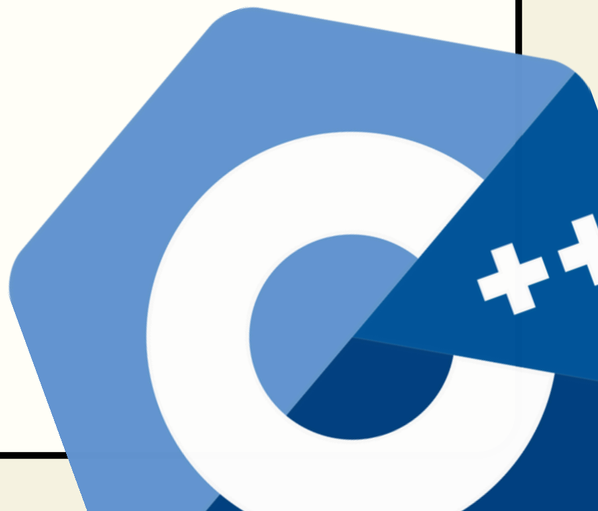
```
enum Colors {RED = 100, GREEN = 69, BLUE};
```

Giá trị của blue khi in ra là bao nhiêu?

Câu 3: Viết chương trình nhập vào 1 số thực bất kỳ và in số đó ra với độ chính xác 5 chữ số thập phân.

NỘI DUNG

- Hằng số
- Toán tử (operator)
- Các bài toán & hàm toán học phổ biến trong C
- Ép kiểu
- Boolean



HẰNG SỐ (CONSTANT)

Hằng số - constant được dùng để định nghĩa về... hằng số **(mang giá trị không thay đổi)**.

Cú pháp:

```
const datatype tênBiến = value;
```

Cú pháp:

```
const int C = 300000000;
const float PI = 3.14;
```

HẰNG SỐ (CONSTANT)

Chú ý: giá trị của một hằng số luôn luôn phải được **gán** khi khai báo hằng số đó.

```
const float PI = 3.14;
```

HỢP LỆ

```
const float PI;
PI = 3.14;
```

KHÔNG HỢP LỆ

HẰNG SỐ (CONSTANT)

Tips: nên viết hoa toàn bộ tên của hằng số như một thói quen.

```
const float PI = 3.14;
const int YEAR = 365;
const int BIRTHDAY = 24;
const int MSSV = 123456;
```

TOÁN TỬ (OPERATOR)

Toán tử (operator) được sử dụng để thực hiện các phép toán với biến & giá trị.

Phân loại:

- Assignment Operators – **Toán tử gán**
- Arithmetic Operators – **Toán tử số học**
- Relational Operators – **Toán tử quan hệ**
- Logical Operators – **Toán tử logic**
- Increment and Decrement Operators – **Toán tử tăng giảm**
- Ternary Operator – **Toán tử ba ngôi**
- Bitwise Operators – **Toán tử bit**

TOÁN TỬ GÁN (P1)

Toán tử gán (assignment operators) được sử dụng để gán giá trị cho biến. Toán hạng **vế trái là biến**, còn toán hạng **vế phải là giá trị được gán**. Giá trị được gán **cũng có thể là 1 biến**.
Giá trị ở vế phải phải có **kiểu dữ liệu khớp** với kiểu của biến ở vế trái, nếu không trình biên dịch sẽ báo lỗi.

Khi chưa học về toán tử số học, ta sẽ tìm hiểu về 1 toán tử gán đơn giản trước là dấu bằng “=”.

TOÁN TỬ GÁN (P1)

Cú pháp:

```
biến = giá trị;
```

Sau khi thực hiện phép gán, giá trị của biến sẽ chính là giá trị vừa được gán.

Ví dụ:

```c
#include <stdio.h>
int main() {
    // Gán giá trị 22 cho biến x
    int x = 22;
    // Gán cho biến a giá trị của biến x
    int a = x;
    printf("%d", a);
    return 0;
} // Output: 22
```

P2 của toán tử gán sẽ được tìm hiểu sau khi chúng ta học về toán tử số học.

TOÁN TỬ SỐ HỌC

Toán tử số học là loại toán tử được sử dụng để **thực hiện các phép toán cơ bản** như cộng, trừ, nhân, chia,...

Toán tử	Tên gọi	Chức năng	Ví dụ
+	Phép cộng	Cộng 2 giá trị với nhau	x + y
-	Phép trừ	Trừ 2 giá trị cho nhau	x - y
*	Phép nhân	Nhân 2 giá trị với nhau	x * y
/	Phép chia	Chia 2 giá trị cho nhau	x / y
%	Phép chia lấy dư (Module)	Lấy dư của phép chia	x % y

TOÁN TỬ SỐ HỌC

```c
#include <stdio.h>

int main()
{
    int a = 100, b = 50;
    printf("%d\n", a + b); // Output: 150
    printf("%d\n", a - b); // Output: 50
    printf("%d\n", a * b); // Output: 5000
    printf("%d\n", a / b); // Output: 2
    printf("%d\n", a % b); // Output: 0
    return 0;
}
```

TOÁN TỬ GÁN (P2)

Ngoài toán tử gán cơ bản là dấu bằng (=), còn có các toán tử gán mở rộng khác. Chúng là sự kết hợp giữa các toán tử số học (như cộng, trừ, nhân, chia) với dấu bằng (=), giúp biểu diễn các phép toán một cách ngắn gọn và dễ đọc hơn.

Các toán tử gán mở rộng này thực hiện cả phép gán và phép tính **trên cùng một dòng lệnh.**

TOÁN TỬ GÁN (P2)

Toán tử	Ví dụ	Chức năng
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5

TOÁN TỬ SO SÁNH

Các **toán tử so sánh (comparison operators)** được sử dụng để **so sánh hai giá trị (hoặc biến)**. Điều này rất quan trọng trong lập trình vì nó giúp chúng ta tìm ra câu trả lời và **đưa ra các quyết định**.
Giá trị trả về của một phép so sánh là **1 hoặc 0**, tương ứng với **đúng (1) hoặc sai (0)**. Các giá trị này được gọi là **giá trị Boolean**, và ta sẽ tìm hiểu thêm về chúng trong chương "Booleans và If..Else".

TOÁN TỬ SO SÁNH

Toán tử	Tên gọi	Chức năng	Ví dụ
>	Lớn hơn	Trả về 1 nếu giá trị thứ nhất **lớn hơn** giá trị thứ hai.	x > y
<	Bé hơn	Trả về 1 nếu giá trị thứ nhất **nhỏ hơn** giá trị thứ hai	x < y
>=	Lớn hơn hoặc bằng	Trả về 1 nếu giá trị thứ nhất **lớn hơn hoặc bằng** giá trị thứ hai	x >= y
<=	Bé hơn hoặc bằng	Trả về 1 nếu giá trị thứ nhất **nhỏ hơn hoặc bằng** giá trị thứ hai	x <= y
==	Bằng	Trả về 1 nếu hai giá trị **bằng nhau**	x == y
!=	Không bằng	Trả về 1 nếu hai giá trị **không bằng nhau**	x != y

TOÁN TỬ LOGIC

Các **toán tử logic** trong C được sử dụng để **kết hợp nhiều điều kiện/ràng buộc.** Toán tử logic **trả về giá trị 0 hoặc 1**, tùy thuộc vào việc kết quả của biểu thức là đúng (true) hay sai (false). Trong lập trình C, chúng ta sử dụng các toán tử logic để **đưa ra quyết định.**

Ví dụ, nếu A và B đúng thì..., nếu A hoặc B xảy ra thì..., nếu x bằng y thì...

TOÁN TỬ LOGIC

Các **toán tử logic** trong C được sử dụng để **kết hợp nhiều điều kiện/ràng buộc.** Toán tử logic **trả về giá trị 0 hoặc 1**, tùy thuộc vào việc kết quả của biểu thức là đúng (true) hay sai (false). Trong lập trình C, chúng ta sử dụng các toán tử logic để **đưa ra quyết định.**

Ví dụ, nếu A và B đúng thì..., nếu A hoặc B xảy ra thì..., nếu x bằng y thì...

TOÁN TỬ LOGIC

Toán tử	Tên gọi	Chức năng	Ví dụ
&&	**AND** (và)	Trả về 1 nếu cả 2 mệnh đề đúng	x && y
||	**OR** (hoặc)	Trả về 1 nếu 1 trong 2 mệnh đề đúng	x || y
!	**NOT** (không)	Đảo ngược giá trị chân lý mệnh đề	!x

TOÁN TỬ TĂNG GIẢM

Trong C, **toán tử tăng – increment operator (++)** và **toán tử giảm – decrement operator (--)** là 2 toán tử dùng để **tăng hoặc giảm toán tử 1 đơn vị**. Tuy nhiên, toán tử này có 2 loại, phụ thuộc vào vị trí đứng của nó so với biến:

- Tiền tố – prefix: ++a --a
- Hậu tố – postfix: a++ a--

TOÁN TỬ TĂNG GIẢM

Khi là prefix: giá trị sẽ được **tăng/giảm trước** theo thứ tự ưu tiên của toán tử, sau đó các phép toán có độ ưu tiên thấp hơn mới được thực hiện.

```
int x;
int a = x++;
```

có thể được viết như:

```
int x;
x = x + 1;
int a = x;
```

TOÁN TỬ TĂNG GIẢM

Khi là postfix: phép tăng/giảm sẽ là phép được thực hiện sau cùng.

```
int x;
int a = x++;
```

có thể được viết như:

```
int x;
x = x + 1;
int a = x;
```

TOÁN TỬ BA NGÔI (TERNARY OPERATOR)

Toán tử ba ngôi (ternary operator) trong C có phần tương tự với câu lệnh if-else, vì nó tuân theo cùng một thuật toán **như if-else**. Tuy nhiên, toán tử ba ngôi chiếm ít không gian hơn và giúp viết các câu lệnh if-else theo cách ngắn gọn nhất có thể.

Cú pháp:

```
(biểu thức logic ? biểu thức nếu đúng : biểu thức nếu sai);
```

TOÁN TỬ BA NGÔI (TERNARY OPERATOR)

VÍ DỤ

```c
int a = 10, b = 20;
int max = a > b ? a : b;
printf("%d", max);
```

Biến max sẽ nhận giá trị là a
vì a > b trả về giá trị logic “sai”.

TOÁN TỬ BIT (BIT OPERATOR)

Trong ngôn ngữ C, có 6 toán tử thao tác trên bit **(bitwise operators)**, còn được gọi là toán tử bit vì chúng hoạt động trên từng bit của dữ liệu. Chúng được sử dụng để thực hiện các phép toán bit trong C:

- Toán tử AND (**&**)
- Toán tử OR (**|**)
- Toán tử XOR (**^**)
- Toán tử dịch trái – left shift (**<<**) và dịch phải – right shift (**>>**)
- Toán tử NOT (**~**)

TOÁN TỬ BIT (BIT OPERATOR)

Đối với ba toán tử **AND, OR và XOR:**

- Biến đổi các số về hệ nhị phân (số bit sẽ tương ứng với bội số của 4 (4, 8, 12, 16, v.v.), sao cho đủ để biểu diễn số đó.
- Thực hiện phép toán với các bit rồi đưa kết quả từ **hệ nhị phân về hệ đếm mong muốn.**

VÍ DỤ: Thực hiện phép tính 5 & 9.

Đổi: 5 → 0101, 9 → 1001

Tính:

$$\begin{array}{r} \& \quad \mathbf{0101} \\ \mathbf{1001} \\ \hline \mathbf{0001} \end{array}$$

Như vậy kết quả là: 0001 → 1 (hệ 10)

TOÁN TỬ BIT (BIT OPERATOR)

BẢNG GIÁ TRỊ CHÂN LÝ

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

TOÁN TỬ BIT (BIT OPERATOR)

Đối với toán tử **NOT:**

- Biến đổi các số về hệ nhị phân (Số lượng bit sẽ được mở rộng thành bội số của 8 (8, 16, 32, v.v.), sao cho đủ để biểu diễn số đó).
- Đảo ngược bit 0 → 1 và 1 → 0.
- Kết quả sẽ đang ở **hệ bù 2.**
- Thực hiện phép toán với các bit rồi đưa kết quả từ **hệ bù 2 về hệ đếm mong muốn.**

VÍ DỤ: Thực hiện phép tính ~5.

Đổi: 5 → 0000 0101

Tính:

$$\sim \begin{array}{r} 0000\ 0101 \\ \hline 1111\ 1010 \end{array}$$

Vậy kết quả là: 1111 1010 (hệ bù 2 – two complementary) → -6 (hệ 10)

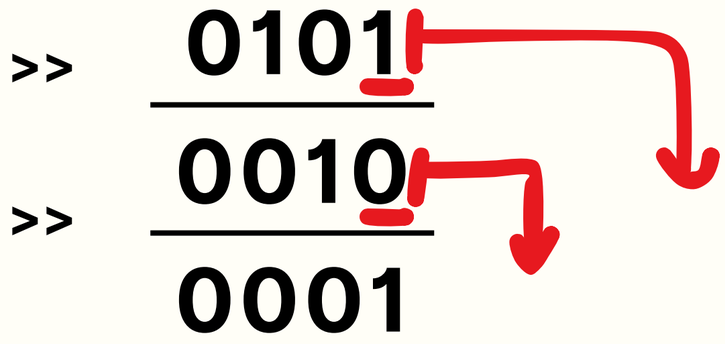
TOÁN TỬ BIT (BIT OPERATOR)

Đối với toán tử “**>>>**” và “**<<<**”:

- Biến đổi các số về hệ nhị phân (số bit sẽ tương ứng với bội số của 4 (4, 8, 12, 16, v.v.), sao cho đủ để biểu diễn số đó.
- Đẩy số gần nhất ra (dịch trái thì đẩy ngoài cùng bên trái, dịch phải tương tự.
- Thay thế bit vừa đẩy bằng bit 0 ở hướng ngược lại.

VÍ DỤ: Thực hiện phép tính 5 >> 2.

Đổi: 5 = 0101. **Tính:**



Vậy kết quả là: 0001 (hệ nhị phân)
→ 1 (hệ thập phân)

CÁC HÀM TOÁN HỌC TRONG C

~x = -(x + 1)

MẸO NHANH

x << y = x * 2 ^ y

x >> y = x / (2 ^ y) (lấy phần nguyên)

CÁC HÀM TOÁN HỌC TRONG C

Trong quá trình lập trình việc sử dụng các hàm toán học là rất phổ biến, ngôn ngữ C cung cấp cho ta một thư viện là **math.h** chứa các hàm toán học.

```
#include <math.h>
```

CÁC HÀM TOÁN HỌC TRONG C

Hàm	Chức năng
pow(x,y)	x mũ y
sqrt(x)	Căn bậc 2
cbrt(x)	Căn bậc 3
ceil(x)	Làm tròn lên **(output là int)**
floor(x)	Làm tròn xuống **(output là int)**
round(x)	Làm tròn **(output là int)**
fabs(x)	Giá trị tuyệt đối

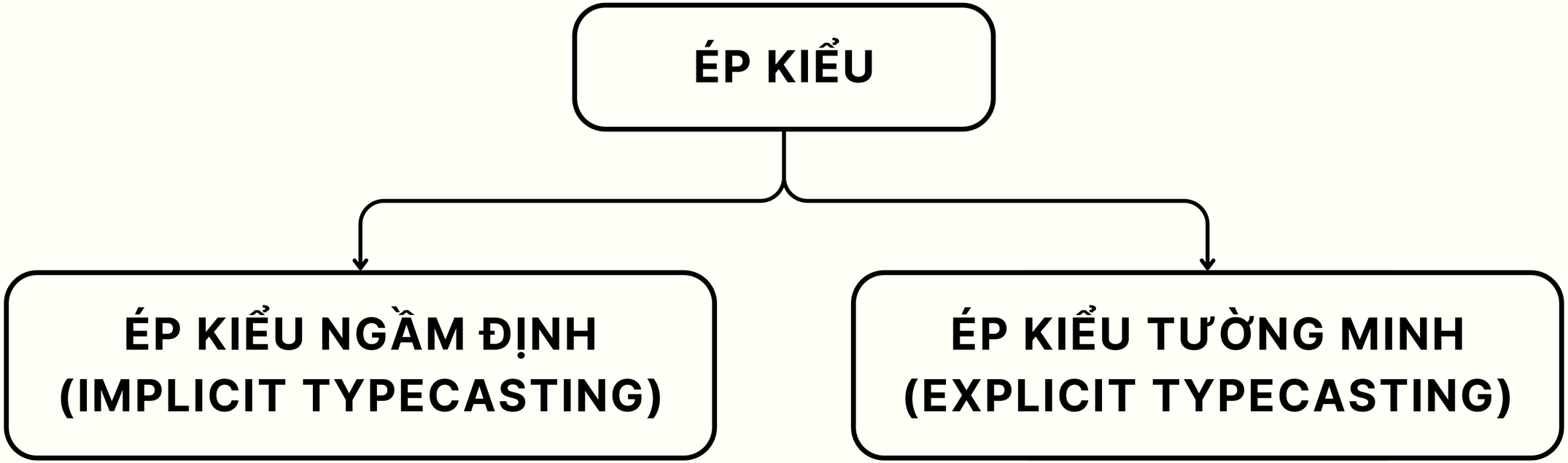
Hàm	Chức năng
exp(x)	e mũ x
fmod(x, y)	x % y
log(x)	logarit của x
log10(x)	logarit thập phân của x
cos(x), sin(x), tan(x)	hàm lượng giác của x **(radian)**
acos(x), asin(x), atan(x)	hàm arc lượng giác của x **(radian)**

ÉP KIỂU (TYPECASTING)

Ép kiểu (Typecasting) trong C là quá trình **chuyển đổi một kiểu dữ liệu này sang một kiểu dữ liệu khác** do lập trình viên thực hiện bằng toán tử ép kiểu trong quá trình thiết kế chương trình.
Trong ép kiểu, kiểu dữ liệu đích có thể nhỏ hơn kiểu dữ liệu nguồn khi chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác, vì vậy nó cũng được gọi là chuyển đổi thu hẹp (narrowing conversion).

ÉP KIỂU (TYPECASTING)

PHÂN LOẠI



ÉP KIỂU (TYPECASTING)

Ép kiểu trong C được chia thành hai loại chính:

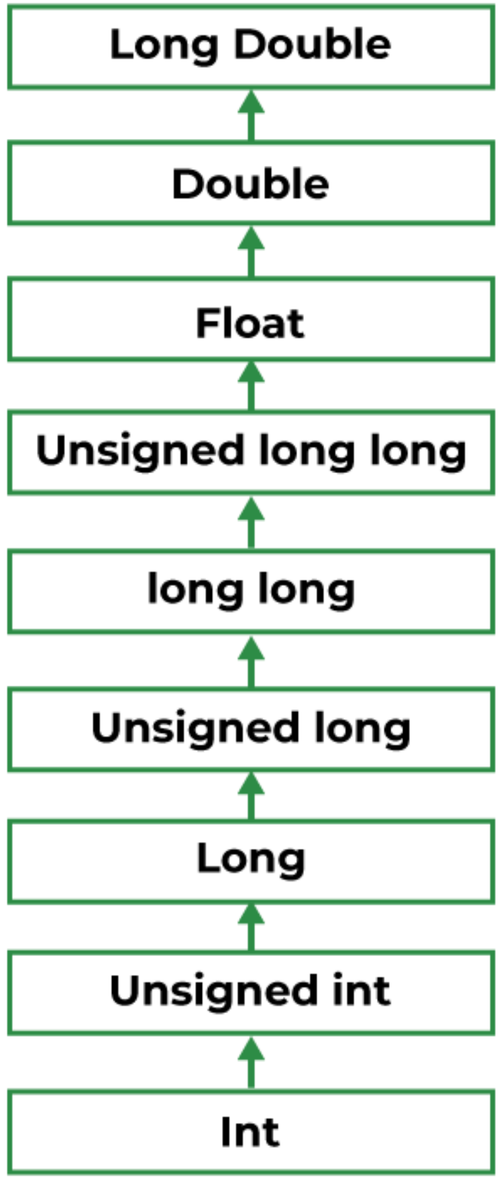
1. Ép kiểu **ngầm định** (Implicit Type Conversion - Type Promotion)
   - Do trình biên dịch tự động thực hiện.
   - Xảy ra khi kiểu dữ liệu có kích thước nhỏ hơn được chuyển đổi sang kiểu dữ liệu có kích thước lớn hơn.
   - Tránh mất dữ liệu nhưng có thể làm tăng bộ nhớ sử dụng.
2. Ép kiểu **tường minh** (Explicit Type Conversion - Type Casting)
   - Do lập trình viên thực hiện bằng cách sử dụng toán tử ép kiểu (type).
   - Có thể làm mất dữ liệu nếu kiểu dữ liệu đích nhỏ hơn kiểu nguồn.

ÉP KIỂU (TYPECASTING)

ÉP KIỂU NGẦM ĐỊNH (IMPLICIT TYPECASTING)

Ép kiểu ngầm định - Implicit Type Casting là khi chuyển đổi kiểu dữ liệu mà **không làm mất đi giá trị ban đầu của biến.** Ép kiểu ngầm định được thực hiện một cách tự động khi bạn gán giá trị của một biến này cho biến khác mang kiểu dữ liệu tương thích, thường là kiểu dữ liệu bao hàm.

Sử dụng hệ thống thứ bậc dưới đây, quá trình chuyển đổi sẽ diễn ra như sau nếu cả hai có kiểu dữ liệu khác nhau:



ÉP KIỂU (TYPECASTING)

ÉP KIỂU NGẦM ĐỊNH (IMPLICIT TYPECASTING)

VÍ DỤ

```
#include <stdio.h>
int main() {
    int a = 10;
    float b = a; // int tự động chuyển thành float
    printf("a = %d, b = %.2f\n", a, b);
    return 0;
} // Output: a = 10, b = 10.00
```

ÉP KIỂU (TYPECASTING)

ÉP KIỂU TƯỜNG MINH (EXPLICIT TYPECASTING)

Có một số trường hợp mà nếu kiểu dữ liệu không thay đổi, chương trình có thể cho kết quả không chính xác. Trong những trường hợp như vậy, ép kiểu có thể giúp nhận được kết quả đúng và giảm thời gian biên dịch.

Trong **ép kiểu tường minh (explicit type casting)**, chúng ta phải buộc thực hiện chuyển đổi giữa các kiểu dữ liệu. Loại ép kiểu này được định nghĩa rõ ràng trong chương trình.

ÉP KIỂU (TYPECASTING)

ÉP KIỂU TƯỜNG MINH (EXPLICIT TYPECASTING)

VÍ DỤ

```
#include <stdio.h>
int main() {
    int a = 10, b = 3;
    // Ép kiểu a thành float trước khi chia
    float c = (float) a / b;
    printf("c = %.2f\n", c);
    return 0; } // Output: c = 3.33
```

BOOLEAN

Thông thường, trong lập trình, bạn sẽ cần một kiểu dữ liệu chỉ có thể có một trong hai giá trị, chẳng hạn như:

- Yes/No - Có/Không
- On/Off - Bật/Tắt
- True/False - Đúng/Sai

Đối với lĩnh vực này, C có một kiểu dữ liệu bool hay còn được gọi là **booleans**, để đại diện cho hai giá trị thật **(truth value)** là **true** và **false** trong đại số Boolean.

Boolean sẽ trả về giá trị có kiểu dữ liệu interger:

- **1 (hoặc tất cả các giá trị khác 0)** tương ứng với **đúng**.
- **0** tương ứng với **sai**.

BOOLEAN

Để sử dụng boolean ta phải thêm thư viện **<stdbool.h>.**

Cú pháp:

```
bool tên_Biến = giá trị (*);
```

(*): true **hoặc** false

Ví dụ:

```
// Tạo các biến kiểu dữ liệu boolean
bool bachKhoaOachKhong = true;
bool hocVuiKhong = false;
// Return boolean values
printf("%d", bachKhoaOachKhong);   // Returns 1 (true)
printf("%d", hocVuiKhong);         // Returns 0 (false)
```

BOOLEAN

Ứng dụng:

```c
#include <stdio.h>
int main()
{
    int tuoiTao = 20;
    int tuoiMay = 18;

    printf("%d", tuoiTao >= tuoiMay); // Output là 1 vì mệnh đề đúng
    return 0;
}
```