Ngôn ngữ lập trình C++

- Bài 2: Hàm, con trỏ, xâu trong C++
- Dịch bởi Đỗ Thị Bích Ngọc

dothibichngoc@gmail.com

Các thành phần của C++

- Modules: functions và classes
- Chương trình sử dụng cả các modules mới và đã có sẵn (“prepackaged”)
  - Mới: functions, classes do lập trình viên định nghĩa
  - Có sẵn: từ thư viện chuẩn
- Định nghĩa và sử dụng hàm: giống C

Hàm inline

- Dùng từ khoá inline
- Trình biên dịch sẽ copy code vào chương trình thay vì dùng lời gọi hàm
  - Giúp giảm số lần gọi hàm
- **Nên dùng cho hàm nhỏ, hay được gọi**
- Ví dụ: fig03_19.cpp

```
inline double cube( double s){
     return s * s * s; }
```

Chồng hàm

- Hàm có tên giống nhau nhưng tham số khác nhau
- Thực hiện nhiệm vụ tương tự nhau
  - Ví dụ: Hàm tính bình phương của int và hàm tính bình phương của float

```
int square( int x) {return x * x;}
float square(float x) { return x * x; }
```

- Phân biệt các hàm chồng nhau?
  – Dựa vào tên và kiểu tham số, có quan tâm tới thứ tự

Tham chiếu và tham trị

- Tham trị
  - copy dữ liệu vào hàm
  - Không thay đổi giá trị của dữ liệu (biến) ban đầu
- Tham chiếu
  - Hàm truy cập trực tiếp vào dữ liệu
  - Thay đổi giá trị của dữ liệu (biến) ban đầu
- Tham số dùng tham chiếu
  - Dùng thêm dấu & trong khai báo hàm:

```
void myFunction( int &data )
```

  - Lời gọi hàm không đổi
- Ví dụ: fig03_20.cpp

Truyền Arrays cho hàm

- Array mặc định truyền kiểu tham chiếu
  - hàm có thể thay đổi dữ liệu array
- Khuôn mẫu Function

```
void modifyArray( int b[], int arraySize );
void modifyArray( int [], int );
```

  - Cả 2 đều nhận 1 mảng int và 1 biến int:
    - Không cần kích thước mảng
    - Bỏ qua bởi trình biên dịch
- Nếu khai báo tham số mảng là const: Không được thay đổi mảng

```
void doNotModify( const int [] );
```

- Ví dụ: fig04_14.cpp & fig04_15.cpp

dothibichngoc@gmail.com

CON TRỎ

Con trỏ - Pointers

- Pointers
  – Hiệu quả, nhưng khó điều khiển
- Truyền theo tham chiếu
  – Quan hệ chặt với mảng và xâu
- Có thể khai báo con trỏ cho bất kì kiểu dữ liệu nào
- Khởi tạo Pointer
  – Khởi tạo bằng **0**, **NULL**(không trỏ vào gì cả), hoặc một ĐỊA CHỈ

Khai báo và khởi tạo biến con trỏ

- Biến con trỏ
  – lưu địa chỉ ô nhớ (memory addresses) như giá trị
- Gián tiếp

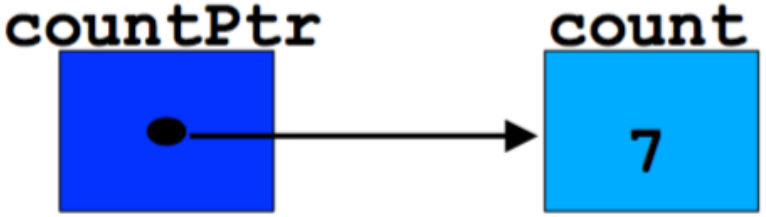
– Tham chiếu tới giá trị bằng con trỏ

- Khai báo

– **Thêm** * vào biến con trỏ

int *myPtr;

int *myPtr1, *myPtr2;



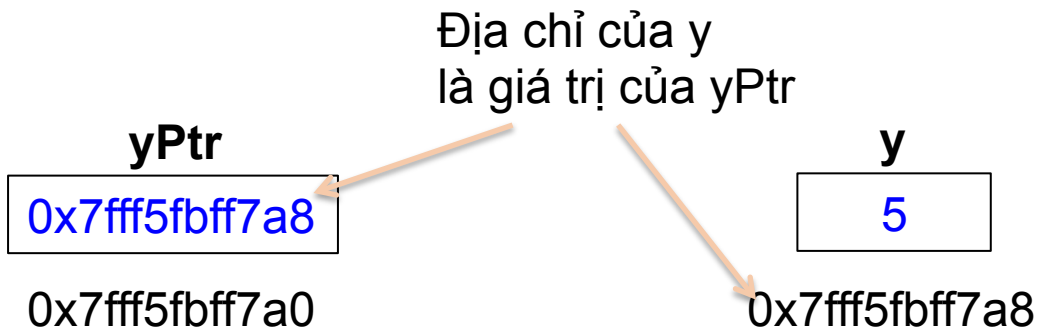
Phép toán với Pointer

- **&** (Phép toán địa chỉ - address operator)

– Trả về địa chỉ ô nhớ

– Ví dụ:

```
int y = 5;
int *yPtr;
yPtr = &y;// yPtr lấy địa chỉ của y
```



Phép toán với Pointer

- **Phép ***

– Lấy giá trị của biến do con trỏ trỏ đến

– ***yPtr** trả về **y** (vì **yPtr** trỏ vào **y**).

– Lưu ý

***yptr = 9; // gán 9 cho y** • * và **&** ngược nhau

Gọi hàm dùng tham chiếu

- 3 cách truyền tham số cho hàm

– Pass-by-value (truyền theo giá trị)

– Pass-by-reference with reference arguments (truyền theo tham chiếu dùng đối số tham chiếu)

– Pass-by-reference with pointer arguments (truyền theo tham chiếu dùng đối số con trỏ)

• return chỉ có thể trả về 1 giá trị cho hàm

• Đối số tham chiếu

– Thay đổi giá trị đối số

– Trả về nhiều hơn 1 giá trị

Gọi hàm dùng tham chiếu

- Truyền tham chiếu bằng đối con trỏ – Tương tự như truyền theo tham chiếu

– Truyền địa chỉ của đối số bằng toán tử **&**

– Không truyền được Arrays bằng **&** vì tên mảng đã là pointer

– **toán tử *** dùng như alias/nickname cho tên biến bên trong hàm

- Ví dụ: fig05_07.cpp

Dùng const với con trỏ

- **const** pointers
  - Luôn trỏ tới 1 ô nhớ cố định
  - Mặc định với tên mảng
  - Phải được khởi tạo khi khai báo
- Ví dụ: fig05_13.cpp & fig05_14.cpp

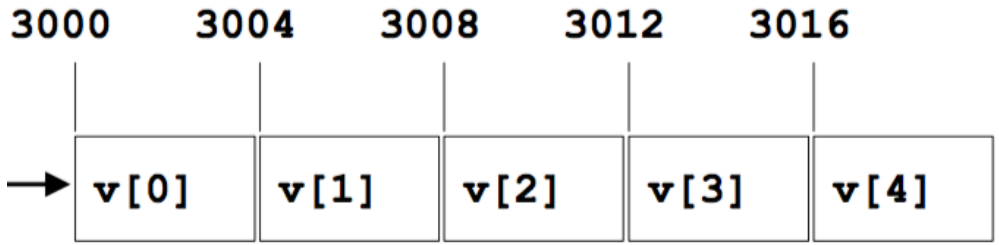
Biểu thức con trỏ và phép toán

- Phép toán con trỏ
  – Tăng/giảm pointer **(++** or **--)**
  – Cộng/trừ số nguyên với con trỏ(**+** or **+=** , **-** or **-=**) – Có thể trừ con trỏ cho nhau
  – Phép toán con trỏ chỉ có nghĩa khi thao tác trên mảng
- Mảng số nguyên 5 phần tử. Sử dụng 4 byte **int**s

– **vPtr** trỏ tới phần tử đầu**v[0]**, tại ô nhớ 3000 **vPtr = 3000**

– **vPtr += 2**; thiết lập **vPtr** = **3008**

vPtr trỏ tới **v[2]**



Biểu thức con trỏ và phép toán

- Phép trừ con trỏ
  – Trả về số phần tử giữa 2 địa chỉ

vPtr2 = v[2];

vPtr = v[0];

vPtr2 - vPtr == 2

- Phép gán con trỏ

–có thể được gán cho con trỏ khác nếu chúng có cùng kiểu

– Nếu không cùng kiểu, phải dùng phép toán ép kiểu cast

– Ngoại lệ: trỏ tới **void** (type **void** *)

• Biểu diễn bất kì kiểu nào

• Không cần ép kiểu cho con trỏ **void**

Quan hệ giữa con trỏ và mảng

- Mảng và con trỏ có quan hệ mật thiết
  - Tên mảng được coi như con trỏ hằng
  - Con trỏ có thể thao tác trên chỉ số mảng
- Truy cập các phần tử mảng bằng con trỏ
  - Phần tử **b[n]** có thể được truy cập bằng ***(bPtr+n)**
  - Địa chỉ, ví dụ: **&b[3]tương đương bPtr + 3**
  - Tên mảng có thể coi như con trỏ, ví dụ:

    b[3] tươngđương ***(b+3)**
  - Có thể đánh chỉ số cho con trỏ, ví dụ:

    bPtr[3] tương đương **b[3]**

Mảng các con trỏ

- Mảng có thể chứa con trỏ

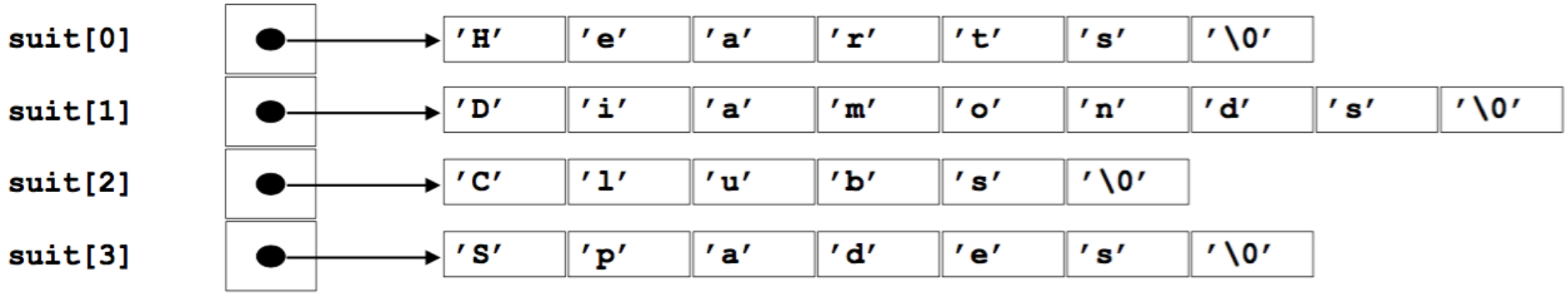
– Thường dùng để lưu mảng của các xâu kí tự

char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades" };

– Mỗi phần tử của **suit** trỏ tới **char *** (một xâu)

– Mảng không chứa xâu, mà chỉ trỏ vào các xâu

– Suit có kích thước cố định, nhưng xâu có kích thước bất kì



Cơ bản về Xâu và Kí tự

Cơ bản về Xâu và Kí tự

- Hằng kí tự - Character constant

– Số nguyên được biểu diễn như kí tự với dấu nháy đơn

– **'z' là giá trị nguyên của z**

• **122** trong ASCII

- Xâu - String

– Dãy các kí tự

– Có thể bao gồm kí tự, số, các kí tự đặc biệt **+**,**-**,*...

– Hằng xâu (string constants)

• Được viết trong dấu nháy kép: **"I like C++"**

– Mảng các kí tự, kết thúc bằng kí tự null **'\0'**

– Xâu là con trỏ hằng

• Trỏ tới phần tử đầu của xâu giống mảng

Cơ bản về Xâu và Kí tự

- Phép gán xâu

– Mảng kí tự: **char color[] = "blue";**

+ Tạo mảng 5 phần tử kiểu **char** có tên **color**

B phần tử cuối là **'\0'**

– Biến kiểu **char *: char *colorPtr = "blue";**

+ Tạo con trỏ **colorPtr** tới kí tự **b** trong xâu **“blue”**

– Dùng tập hợp: **char color[] = { ‘b’, ‘l’, ‘u’, ‘e’, ‘\0’ };**

Cơ bản về Xâu và Kí tự

- Đọc xâu

– Nhập xâu kí tự và gán cho mảng **word[20]**

cin >> word

- Đọc kí tự cho tới khi gặp dấu cách hoặc EOF
- Xâu có thể lớn hơn kích thước mảng

cin >> setw(20) >> word;

- Đọc 19 kí tự (để dành 1 vị trí cho **'\0'**)

Cơ bản về Xâu và Kí tự

- **cin.getline**

– Đọc dòng kí tự

– **cin.getline(array, size, delimiter);**

– Copy đầu vào vào 1 mảng **array** đã cho, tới khi

  - đạt tới kích thước **size-1**
  - **gặp kí tự delimiter**

– Ví dụ:

```
char sentence[ 80 ];
cin.getline( sentence, 80, '\n' );
```

Các hàm thao tác xâu và thư

- Thư viện quản lý xâu **<cstring>** cung cấp hàm để
  - Thao tác dữ liệu trên xâu
  - So sánh xâu
  - Tìm kiếm kí tự hoặc xâu kí tự trên xâu cho trước
  - Chia nhỏ xâu

Các hàm thao tác xâu và thư viện quản lý xâu

char *strcpy(char *s1, const char *s2);	copy xâu **s2** vào **s1**. Trả về xâu **s1**.
char *strncpy(char *s1, const char *s2, size_t n);	Copy tối đa **n** kí tự của xâu **s2** vào **s1**. Trả về **s1**.
char *strcat(char *s1, const char *s2);	**Nối xâu s2** vào **s1**. Kí tự đầu của **s2** sẽ ghi đè lên kí tự kết thúc null của **s1**. Trả về **s1**.
char *strncat(char *s1, const char *s2, size_t n);	Nối tối đa **n** kí tự của **s2** vào **s1**. Kí tự đầu của **s2** sẽ ghi đè lên kí tự kết thúc null của **s1**. Trả về **s1**.
int strcmp(const char *s1, const char *s2);	So sánh **s1** với **s2**. Trả về 0, <0 hoặc >0 nếu **s1** bằng, nhỏ hơn, lớn hơn **s2**.

Các hàm thao tác xâu và thư viện quản lý xâu

int strncmp(const char *s1, const char *s2, size_t n);	So sánh n kí tự đầu tiên của xâu **s1** với **s2**. Trả về 0, <0, >0 tương ứng với **s1** bằng, nhỏ hơn, hay lớn hơn **s2**.
size_t strlen(const char *s);	chiều dài xâu **s**