NGÔN NGỮ LẬP TRÌNH C++

SILDE 2.5: CON TRỎ HÀM VÀ LAMBDA EXPRESSION

Giảng viên: Th.S Bùi Văn Kiên

Nội dung

- Con trỏ hàm
- Hàm inline
- Lambda expression

1. Con trỏ hàm

- Con trỏ hàm (pointer to a function) là một biến lưu trữ địa chỉ của một hàm, thông qua biến đó, ta có thể gọi hàm mà nó trỏ tới.
- Cú pháp

  <kiểu trả về> (*<tên con trỏ>)(<danh sách tham số>);
- Ví dụ:

  int(*funcPtr)(int);

  void(*funcPtr)(int, int);

1. Con trỏ hàm

- Ví dụ:

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

void sayHello(){
    cout << "Hello World\n";
}

void sayGoodbye(){
    cout << "Goodbye, World!\n";
}

int main(){
    void (*funcPtr)(); /// khai báo một con trỏ hàm void

    funcPtr = sayHello; /// trỏ tới hàm sayHello
    /// gọi hàm thông qua con trỏ
    funcPtr();          /// -> in ra: Hello World

    /// trỏ tới hàm sayGoodbye
    funcPtr = sayGoodbye;
    funcPtr();          /// -> in ra: Goodbye, World!

    return 0;
}
```

1. Con trỏ hàm

Chức năng:

- Dynamic Function Call: Chúng cho phép gọi hàm động, nghĩa là bạn có thể chọn hàm nào sẽ thực thi khi chạy
- Dynamic Memory Management: phân bố và quản lý bộ nhớ động tốt hơn, đặc biệt là khi bộ nhớ có kích thước giới hạn.
- Callbacks: được sử dụng trong lệnh gọi lại cho lập trình hướng sự kiện (xử lý ngắt trong lập trình nhúng).
- Flexibility (Tính linh hoạt): Cho phép truyền các hàm làm đối số cho các hàm khác, tạo điều kiện thuận lợi cho việc tạo mã linh hoạt hơn và có thể tái sử dụng
- Efficiency (Hiệu quả): Có thể làm cho việc triển khai state machine hiệu quả hơn bằng cách giảm nhu cầu về nhiều cấu trúc if-else.

1. Con trỏ hàm

- Lỗi thường gặp
  - int(*fnPtr)() = funcA(); // lỗi, không dùng dấu ngoặc đơn () sau tên hàm
  - int(*fnPtrA)() = funcA; // ok, con trỏ fcnPtrA trỏ đến hàm funcA
  - fnPtrA = funcB;

  // ok, fcnPtrA có thể trỏ đến một hàm khác có cùng cấu trúc
  - fnPtrA = &funcB; // ok

  - int(*fnPtr1)() = funcA; // ok
  - void(*fnPtr2)() = funcA;

  // lỗi, kiểu trả về của con trỏ hàm và hàm khác nhau
  - void(*fnPtr3)() = funcC; // ok
  - double(*fnPtr4)(int) = funcD; // ok

1. Con trỏ hàm

- Truyền tham số con trỏ hàm
  - void sort(int *arr, int n, bool(*comparisonFunc)(int, int))
  - void sort(int *arr, int n, bool(*comparisonFunc)(int, int) = asc)

1. Con trỏ hàm

- Ví dụ:

```
bool ascending(int left, int right){
    return left > right;
}

bool descending(int left, int right){
    return left < right;
}

void selectionSort(int *arr, int length, bool (*comparisonFunc)(int, int)){
    for (int i_start = 0; i_start < length; i_start++)  {
        int minIndex = i_start;

        for (int i_current = i_start + 1; i_current < length; i_current++){
            // use function pointer as ascending or descending function
            if (comparisonFunc(arr[minIndex], arr[i_current])) {
                minIndex = i_current;
            }
        }

        swap(arr[i_start], arr[minIndex]);
    }
}
```

1. Con trỏ hàm

- Ví dụ:

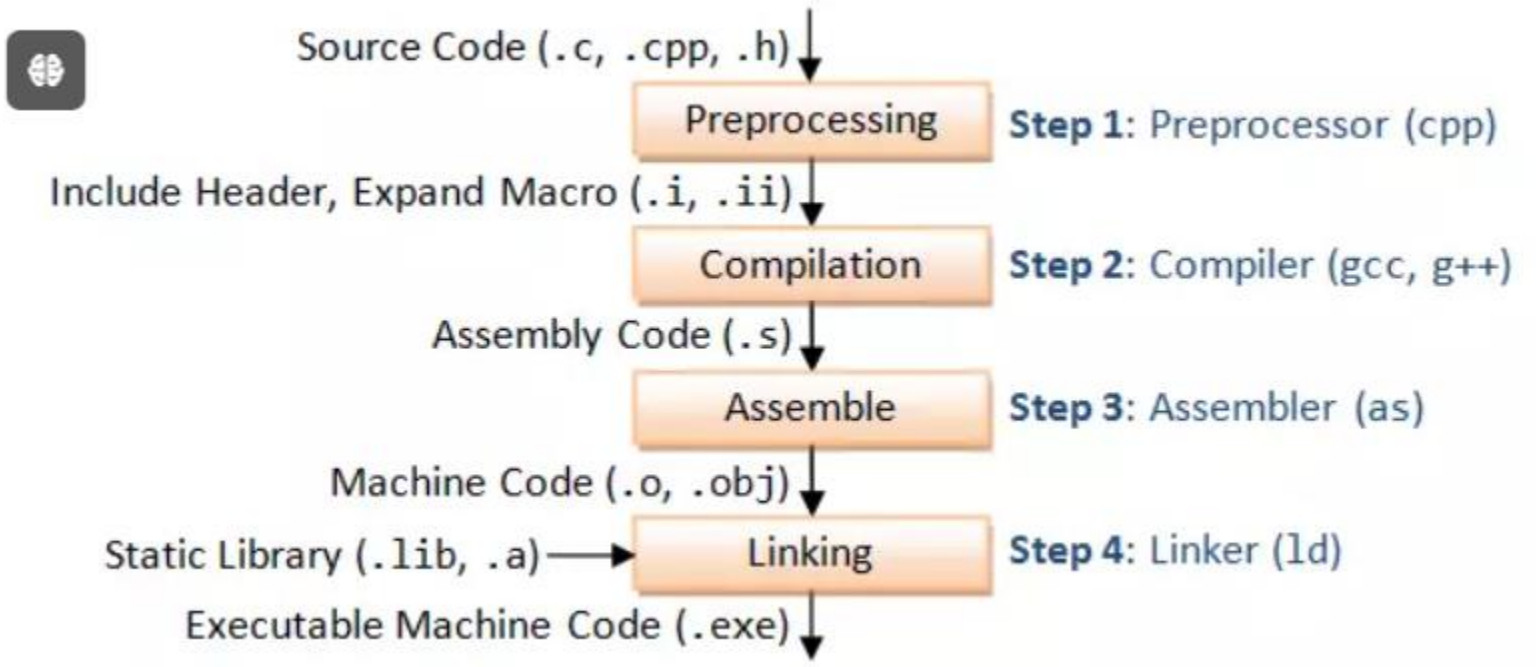
```
int main(){
    int arr[] = { 1, 4, 2, 3, 6, 5, 8, 9, 7 };
    int length = sizeof(arr) / sizeof(int);

    cout << "Before sorted: ";
    for(int i = 0; i < length; i++)
        cout << arr[i] << " ";
    cout << endl;

    selectionSort(arr, length, descending);

    cout << "After sorted:  ";
    for(int i = 0; i < length; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

2. Hàm inline

- Inline function là một chức năng trong ngôn ngữ lập trình C++.
- Hàm inline là hàm được định nghĩa bằng từ khóa inline phía trước tên hàm.
- Hàm inline được sử dụng để yêu cầu trình biên dịch (compiler) thay thế lời gọi hàm bằng toàn bộ mã code của hàm nhằm mục đích giảm thời gian chạy chương trình.

2. Hàm inline

- Hàm inline sẽ được trình biên dịch thực hiện thay thế vị trí gọi hàm bằng nội dung hàm inline (step 4).



Quá trình biên dịch một chương trình

2. Hàm inline

- Thông thường, mỗi hàm sẽ có địa chỉ nhất định. Khi gọi thì CPU sẽ nhảy tới địa chỉ đó.

→ Việc này sẽ gây lãng phí tài nguyên đối khi chạy chương trình với các hàm ngắn chỉ gồm 1 vài câu lệnh (vì thời gian gọi hàm lớn hơn nhiều so với thời gian thực hiện mã hàm).

- Khi sử dụng hàm inline, complier sẽ chèn luôn code của hàm đó vào.

→ dùng hàm inline sẽ tiết kiệm nhiều tài nguyên trong quá trình chạy chương trình, tuy nhiên, nó sẽ làm tăng kích thước file thực thi.

3. Lambda expression

- Lambda function (hàm lambda) là một tính năng được giới thiệu từ C++11, cho phép định nghĩa các hàm inline vô danh (anonymous functions) một cách ngắn gọn. Lambda function trong C++ rất hữu ích khi chúng ta cần một hàm nhỏ, chỉ dùng một vài lần và không cần định nghĩa riêng biệt.

3. Lambda expression

Cấu trúc

- [capture](parameters) -> return_type { body }
  - capture: Phần này chỉ định các biến bên ngoài mà lambda có thể truy cập.
  - parameters: Danh sách các tham số của lambda (giống như tham số hàm thông thường).
  - return_type: Kiểu trả về của lambda (có thể bỏ qua nếu trình biên dịch có thể suy luận).
  - body: Thân của lambda, nơi thực hiện các tính toán.

3. Lambda expression

- Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
	// Lambda đơn giản không có tham số và không bắt biến ngoài
	auto greet = []() {
		cout << "Hello, World!" << endl;
	};
	greet();  // Output: Hello, World!

	return 0;
}
```

3. Lambda expression

- Phần capture: cho phép lambda truy cập các biến bên ngoài phạm vi của nó. Có các tùy chọn để bắt biến:
  - [ ] : Không bắt bất kỳ biến nào.
  - [x] : Bắt biến x theo kiểu giá trị.
  - [&x] : Bắt biến x theo kiểu tham chiếu.
  - [=] : Bắt tất cả các biến bên ngoài theo kiểu giá trị.
  - [&] : Bắt tất cả các biến bên ngoài theo kiểu tham chiếu.
  - [=, &y] : Bắt tất cả các biến theo kiểu giá trị trừ y là tham chiếu.
  - [&, x] : Bắt tất cả các biến theo kiểu tham chiếu trừ x là giá trị.

3. Lambda expression

- Phần capture:

```
int main() {
    int a = 5, b = 10;
    // Bắt cả hai biến a và b theo giá trị
    auto sum = [=]() {
        return a + b;
    };

    // Bắt biến a theo giá trị, b theo tham chiếu
    auto update_b = [a, &b]() {
        b = a + 20;
    };

    cout << "Sum: " << sum() << endl;  // Output: 15
    update_b();
    cout << "Updated b: " << b << endl;  // Output: 25
    return 0;
}
```

3. Lambda expression

- Kiểu trả về: Lambda có thể suy luận kiểu trả về dựa trên biểu thức bên trong nó. Nếu bạn cần chỉ định kiểu trả về, bạn có thể sử dụng cú pháp -> return_type.

```
int main() {
    // Lambda với kiểu trả về
    auto divide = [](int a, int b) -> double {
        if (b == 0) {
            return 0;  // Tránh chia cho 0
        }
        return (double)a / b;
    };

    cout << "Result: " << divide(10, 3) << endl;  // Output: 3.33333
    return 0;
}
```

3. Lambda expression

- parameters: Đây là các tham số giống như trong một hàm thông thường.

```
int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    auto sum = [](int a, int b) -> int {
        return a + b;
    };
    int total = 0;
    for (int number : numbers) {
        total = sum(total, number);
    }

    std::cout << "Tổng là: " << total << std::endl;
    return 0;
}
```

3. Lambda expression

- Sử dụng lambda expression với sort của thư viện STL

```
int main() {
    std::vector<int> numbers = {5, 2, 9, 1, 5, 6};

    std::sort(numbers.begin(), numbers.end(), [](int a, int b) {
        return a > b;
    });

    for (int num : numbers) {
        std::cout << num << " ";  // Output: 9 6 5 5 2 1
    }

    return 0;
}
```

3. Lambda expression

- Sử dụng sort của STL với con trỏ hàm

```
int cmp(int a, int b) {
    return a > b;
}

int main() {
    std::vector<int> numbers = {5, 2, 9, 1, 5, 6};

    std::sort(numbers.begin(), numbers.end(), cmp);
    for (int num : numbers) {
        std::cout << num << " ";  // Output: 9 6 5 5 2 1
    }
    return 0;
}
```

4. Bài tập

- Viết chương trình nhập số N ($N < 10^6$), in ra các số từ 1 → N thỏa mãn có tổng các chữ số là số nguyên tố.
- Nhập vào một số (không quá 500 kí tự).

In ra YES nếu như đây là số thuận nghịch và chỉ chứa hoàn toàn các chữ số lẻ.

In ra NO trong trường hợp ngược lại.

QUESTIONS & ANSWERS