Khoa CNTT – ĐH Khoa học Tự nhiên
TP HCM, tháng 02/2022

Kỹ thuật lập trình

Con trỏ - 02

Nguyễn Trọng Việt

Mục tiêu

Sau buổi học, sinh viên có thể:

- Nêu mối tương quan giữa con trỏ và mảng/chuỗi
- Cấp phát và giải phóng bộ nhớ động

Nội dung

- Con trỏ và mảng/chuỗi
- Cấp phát và giải phóng bộ nhớ động
- Xu hướng hiện đại

Nội dung

- Con trỏ và mảng/chuỗi
- Cấp phát và giải phóng bộ nhớ động
- Xu hướng hiện đại

Con trỏ và mảng

```
void main()
{
    int array[10];
    int* pointer;

    pointer = array;

    printf("%d\n", array);
    printf("%d\n", &array[0]);
    printf("%d\n", pointer);
}
```

Con trỏ và mảng

```
void main()
{
    int array[10];
    int* pointer;

    array = pointer;

    ...
}
```

Con trỏ và mảng (tiếp)

- Mảng là cấu trúc dữ liệu lưu trữ dạng 1 tập hợp có thứ tự, liên tiếp, có kích thước cố định gồm các phần tử thuộc cùng 1 kiểu dữ liệu
- Con trỏ là một biến dùng để lưu địa chỉ của một vùng nhớ

→ 2 khái niệm hoàn toàn khác nhau

http://www.c-faq.com/aryptr/aryptr2.html (05/2020)

Con trỏ và mảng (tiếp)

- Con trỏ và mảng tương đồng nhau:
  - Nội dung lưu trữ
  - Cách truy xuất nội dung phần tử
  - Cách thực hiện các phép toán số học
- Cách dùng tương tự nhau
- Không thể hoán đổi vai trò cho nhau
  - Có thể gán cho con trỏ địa chỉ của bất kì vùng nhớ có kiểu phù hợp
  - Không thể gán cho mảng địa chỉ của 1 vùng nhớ khác
- Compiler ngầm chuyển tự động 1 mảng thành con trỏ

http://www.c-faq.com/aryptr/aryptrequiv.html (05/2020)

Địa chỉ mảng

- Biến thuộc kiểu mảng chứa giá trị là địa chỉ mảng
- Địa chỉ mảng trùng với địa chỉ phần tử đầu tiên trong mảng
- Kích thước của 1 biến mảng bằng …

```
void main()
{
    int array[10];

    printf("%d\n", array);
    printf("%d\n", &array[0]);
}
```

Địa chỉ mảng (tiếp)

- Biến thuộc kiểu mảng chứa giá trị là địa chỉ mảng
- Có thể gán biến mảng cho con trỏ

```
void main()
{
    int array[10];
    int* pointer;

    pointer = array;

    printf("%d\n", array);
    printf("%d\n", &array[0]);
    printf("%d\n", pointer);
}
```

Địa chỉ mảng (tiếp)

- Có thể gán biến mảng cho con trỏ
- Không thể hoàn đổi vai trò

```
void main()
{
    int array[10];

    int a = 10;
    int* pointer = &a;

    array = pointer;    // wrong
}
```

Truy xuất giá trị phần tử

```
void main()
{
    int array[] = {1, 2, 3};
    int* pointer = array;

    int v1 = array[1];
    int v2 = pointer[1];
}
```

Chương trình thực hiện truy xuất giá trị phần tử thế nào ?

- array[1]:
- pointer[1]:

Truy xuất giá trị phần tử mảng - cách 1

- Toán tử: [ ]

```
<array_or_pointer>[<index>]
```

  - Toán tử 2 ngôi
  - Input:
    - Toán hạng 1: biến mảng/con trỏ
    - Toán hạng 2: chỉ số phần tử
  - Output:
    - Nội dung phần tử tại chỉ số của toán hạng 2
- Sử dụng
  - Toán hạng 1 đặt trước [
  - Toán hạng 2 kẹp giữa [và]

Truy xuất giá trị phần tử mảng - cách 1

```
void main()
{
    int array[10] = {1, 2, 3, 4];
    int* pointer;

    pointer = array;

    printf("%d\n", array[1]);
    printf("%d\n", pointer[1]);

    array[2] += 5;
    pointer[3] += 6;
}
```

Truy xuất giá trị phần tử mảng - cách 2

- Kết hợp:

```
*(<array_or_pointer> + <index>)
```

  - Di chuyển đến vị trí phần tử cần truy xuất
  - Sử dụng toán tử * để truy xuất giá trị
  - Input:
    - Toán hạng 1: biến mảng/con trỏ
    - Toán hạng 2: chỉ số phần tử
  - Output:
    - Nội dung phần tử tại chỉ số của toán hạng 2

Truy xuất giá trị phần tử mảng - cách 2

```
void main()
{
    int array[10] = {1, 2, 3, 4];
    int* pointer;

    pointer = array;

    printf("%d\n", *(array + 1));
    printf("%d\n", *(pointer + 1));

    *(array + 2) += 5;
    *(pointer + 3) += 6;
}
```

Con trỏ và mảng

```
void main()
{
  int numbers[5];
  int* p;

  p = numbers;  *p = 10;
  p++;  *p = 20;

  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;

  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
}
```

http://www.cplusplus.com/doc/tutorial/pointers/ (05/2020)

Con trỏ và mảng (tiếp)

	Con trỏ	Mảng
Giống		
Khác		

Con trỏ và chuỗi

- Chuỗi giống mảng kí tự ???

```
void main()
{
  const char* str = "Hello";
  cout << str << endl;
}
```

Tham số mảng

- Biến kiểu mảng có giá trị là địa chỉ mảng
- Truyền tham số mảng = truyền địa chỉ mảng
- Quirk syntax

```
void printArray(int a[], int n)
{
    for (int i = 0; i < n; i++)
       printf("%d ", *(a++) );
}

void main()
{
    int  a[100];
    printArray(a, 100);

    for (int i = 0; i < n; i++)
        printf("%d ", *(a++) );
}
```

Tham số mảng (tiếp)

- Biến kiểu mảng có giá trị là địa chỉ mảng
- Truyền tham số mảng = truyền địa chỉ mảng

```
void printArray(int* p, int n)
{
    for (int i = 0; i < n; i++)
       printf("%d ", p[i] );
}

void main()
{
    int  a[100];
    printArray(a, 100);
}
```

Nội dung

- Con trỏ và mảng/chuỗi
- Cấp phát và giải phóng bộ nhớ động
- Xu hướng hiện đại

Cấp phát & giải phóng – C style

- Thư viện: `stdlib.h`
- Hàm cấp phát

```
void* malloc(<size_in_byte>)
```

- Hàm giải phóng

```
void free(void *)
```

- Trong đó:
  - Tham số cho `malloc`: `size_in_byte` là số lượng byte bộ nhớ muốn cấp phát
  - Tham số cho `free`: con trỏ trỏ đến vùng nhớ được cấp phát bởi `malloc`

Cấp phát & giải phóng – C style (tiếp)

- Các lưu ý khi sử dụng
  - Xác định số byte cần cấp phát khi gọi `malloc`
  - Phải kiểu tra xem vùng nhớ có được cấp phát thành công không trước khi sử dụng
  - Phải giải phóng vùng nhớ đã cấp phát

- Best practise
  - Nên ép về kiểu mong muốn sau khi cấp phát bằng `malloc`
  - Không cần check `NULL/nullptr` trước khi gọi `free`
  - Set con trỏ về `NULL/nullptr` sau khi gọi `free`

Cấp phát & giải phóng – C style (tiếp)

```
struct Student {
    char firstName[64];
    char lastName[64];
    char id[32];
    int age;
};

int main() {
    Student* p = (Student *)malloc(sizeof(Student));
    if ( !p ) {
        printf("memory drained\n");
        return 0;
    }

    ...
    free( p );
    p = nullptr;

    return 1;
}
```

Cấp phát & giải phóng – C style (tiếp)

```
struct Student {
    char firstName[64];
    char lastName[64];
    char id[32];
    int age;
};

int main() {
    Student* pa = malloc( sizeof(Student) * 5 );
    if ( !pa ) {
        printf("memory drained\n");
        return 0;
    }

    ...
    free( pa );
    pa = nullptr;

    return 1;
}
```

Cấp phát & giải phóng – C++ style

- Thư viện: `new`
- Toán tử cấp phát

```
T* new T;
T* new T[n_element];
```

- Toán tử giải phóng

```
delete p;
delete[] p;
```

- Trong đó:
  - `T`: kiểu dữ liệu
  - `n_element`: số lượng phần tử muốn cấp phát
  - `p`: con trỏ cần giải phóng

Cấp phát & giải phóng – C++ style (tiếp)

- Các lưu ý khi sử dụng:
  - Cú pháp cấp phát và giải phóng con trỏ trỏ đến vùng nhớ đơn và mảng phần tử là khác nhau
  - Cấp phát vùng nhớ đơn bằng `new T` phải giải phóng bằng `delete`
  - Cấp phát mảng bang `new T[]` phải giải phóng bằng `delete[]`
  - Tuyệt đối không dùng toán tử giải phóng vùng nhớ đơn để giải phóng mảng và ngược lại
  - Tuyệt đối không dùng sai cặp `new` – `free`, `malloc` – `delete`
  - Phải giải phóng vùng nhớ đã cấp phát

Cấp phát & giải phóng – C++ style (tiếp)

- Best practice:
  - Không cần kiểm tra con trỏ có được cấp phát thành công sau `new/new T[]`
  - Xử lý ngoại lệ (exception) khi `new/new T[]` thất bại
  - Không cần check `nullptr` trước khi gọi `delete/delete[]`
  - Set con trỏ về `nullptr` sau khi gọi `delete/delete[]`

Cấp phát & giải phóng – C++ style (tiếp)

```
struct Student {
    char firstName[64];
    ...
};

int main() {
    Student* p = nullptr;
    try {
        p = new Student;
    }
    catch(std::bad_alloc& e) {
        printf("memory drained\n");
        return 1;
    }

    ...
    delete p;
    p = nullptr;

    return 0;
}
```

Cấp phát & giải phóng – C++ style (tiếp)

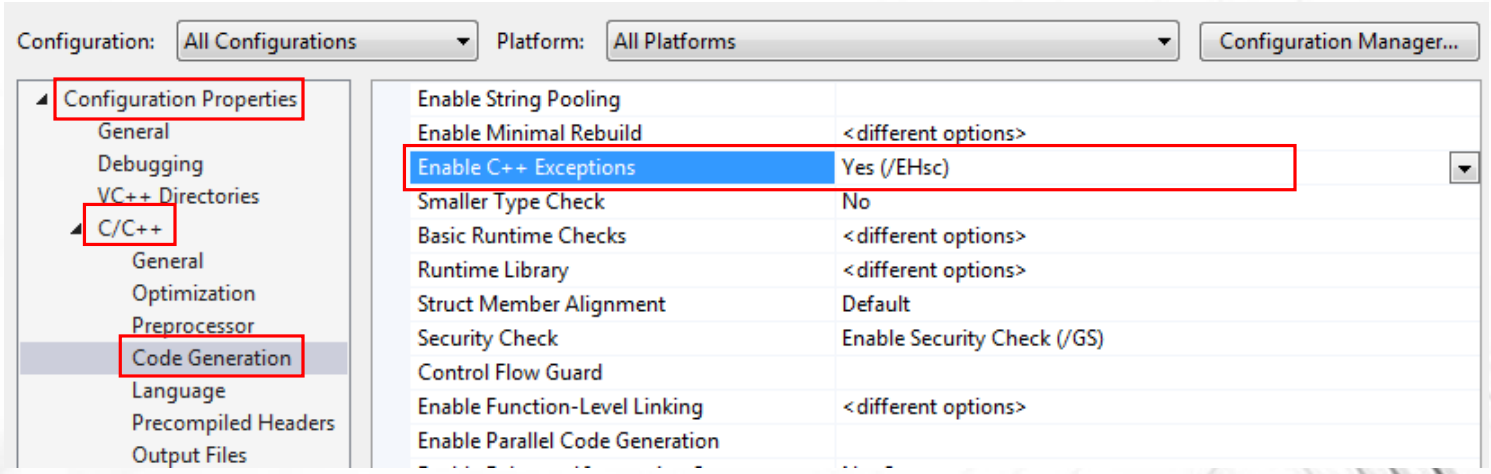
```
struct Student;

int main() {
    int n = 0;
    Student* p = nullptr;
    ...
    try {
        p = new Student[n];
    }
    catch(std::bad_alloc& e) {
        printf("failed to new, %s\n", e.what());  return 1;
    }
    catch(std::bad_array_new_length& e) {
        printf("length issue, %s\n", e.what());   return 2;
    }
    catch(std::exception& e) {
        printf("other issue, %s\n", e.what());   return 3;
    }
    ...
    delete[] p;
    p = nullptr;

    return 0;
}
```

Cấp phát & giải phóng – C++ style (tiếp)

- Sợ exception, dùng phiên bản khác của `new`

```
T* new (std::nothrow) T;
T* new (std::nothrow) T[n_element];
```

- Cần đảm bảo chương trình được compile với chỉ thị: /Ehsc
- Vào Project -> Properties -> Configuration Properties -> C/C++ -> Code Generation



Cấp phát & giải phóng – C++ style (tiếp)

- Sợ exception, dùng phiên bản `std::nothrow` của `new`

```
#include <new>
struct Student;

int main() {
    int n = 0;
    ...
    Student* p = new (std::nothrow) Student[n];
    if ( !p ) {
        printf("memory drained\n");
        return 0;
    }
    ...
    delete[] p;
    p = nullptr;

    return 1;
}
```

Cấp phát & giải phóng – C++ style (tiếp)

```
int* p1 = new int[10];
delete p1;

int* p2 = (int*)malloc( sizeof(int) * 10 );
delete[] p2;

int *p3 = new int[10];
free(p3);

Student* p4 = new Student;
delete p4;
delete p4;

Student* p5 = new Student;
Student* p6 = p5;
delete p5;
delete p6;
```

Cấp phát ma trận động 2 chiều

- Nhập số dòng và cột từ bàn phím
- Cấp phát ma trận động 2 chiều
- Gán giá trị ngẫu nhiên cho ma trận
- Giải phóng ma trận đã cấp phát

Tản mạn giải phóng con trỏ đến mảng

- Làm cách nào C++ có thể thu hồi đủ vùng bộ nhớ được cấp phát động cho mảng 1 chiều thông qua con trỏ ?

```
int* p = new int[100];
delete[] p;
```

Sơ lược lớp đối tượng

Lớp đối tượng trong C++

- Kiểu dữ liệu do người dùng tự định nghĩa
- Được định nghĩa với từ khóa `class`
- Kết hợp cả dữ liệu và hàm thao tác trên dữ liệu vào cùng 1 gói (package).
  - Dữ liệu gọi là thuộc tính
  - Hàm gọi là phương thức
- Hỗ trợ nhiều hình thức truy xuất thuộc tính và gọi phương thức với các từ khóa `private`, `protected` và `public`

Sơ lược lớp đối tượng

Lớp đối tượng định nghĩa 2 phương thức đặc biệt

- **Constructor** (hàm khởi tạo) được gọi tự động khi đối tượng của lớp được tạo ra, sau khi vùng nhớ của đối tượng được cấp phát
- **Destructor** (hàm hủy) được gọi tự động khi đối tượng bị hủy, trước khi vùng nhớ của đối tượng bị thu hồi

Sơ lược lớp đối tượng

```
// person.h file
class Person
{
public:
    Person();  // constructor
    ~Person(); // destructor

    void eat();
    void walk();

private:
    char m_name[128];
    int m_age;
};
```

```
// person.cpp file
Person::Person(){
   printf("constructor\n");
}

Person::~Person(){
   printf("destructor\n");
}

void Person::eat() {
}

void Person::walk() {
}
```

```
// main.cpp
void main() {
    Person me;
    me.walk();
    me.eat();
}
```

Cấp phát & giải phóng con trỏ đối tượng

```
class Person;

void main()
{
    Person* p = new Person(); // call constructor
    p->walk();
    p->eat();

    delete p;                 // call destructor
    p = nullptr;
}
```

- Cú pháp cấp phát & giải phóng tương tự structure
- Phải dùng `new` vì sau khi cấp phát vùng nhớ cho đối tượng, `new` sẽ gọi constructor của đối tượng
- Phải dùng `delete` vì trước khi giải phóng vùng nhớ của đối tượng, `delete` sẽ gọi destructor của đối tượng

Nội dung

- Con trỏ và mảng/chuỗi
- Cấp phát và giải phóng bộ nhớ động
- Xu hướng hiện đại

Xu hướng hiện đại

- Sử dụng vector thay cho cấp phát mảng động với `malloc/new`

```
#include <vector>

void main()
{
    std::vector<int> a = {1, 2, 3, 4};
    a.push_back(5);
    cout << a[4];
    cout << a.size();

    for( int item: a ) {
        cout<< item;
    }
}
```

Xu hướng hiện đại

- Sử dụng con trỏ thông minh thay cho con trỏ thô
  - `std::unique_ptr`
  - `std::shared_ptr`
  - `std::weak_ptr`

Xu hướng hiện đại

- Sử dụng con trỏ thông minh thay cho con trỏ thô

```
#include <memory>

class Person;

void main()
{
    std::shared_ptr<Person> me = std::make_shared<Person>();
    std::weak_ptr<Person> shadow = me;

    me->walk();
    me->eat();

    me = nullptr;

    std::shared_ptr<Person> me_again = shadow.lock();
    if ( !me_again ) {
        cout << "object has been destroyed";
    }
}
```

Đánh giá đạt mục tiêu

Sau buổi học, liệu sinh viên có thể:

- Nêu mối tương quan giữa con trỏ và mảng/chuỗi ?
- Cấp phát và giải phóng bộ nhớ động ?