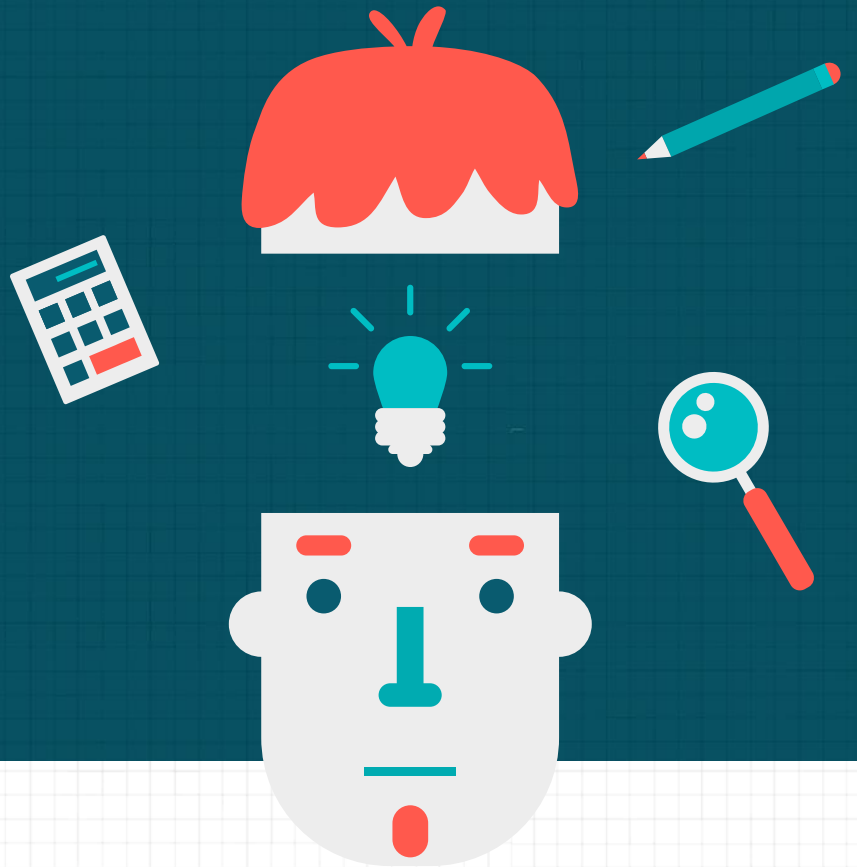
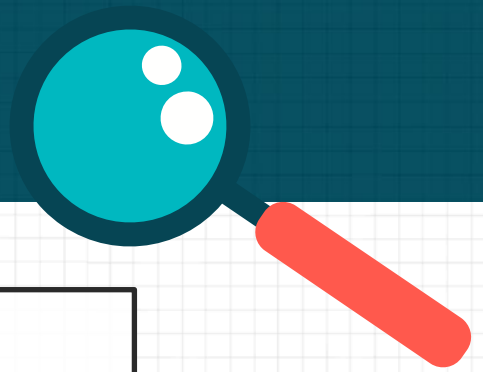
Bài 2
Quản lý bộ nhớ

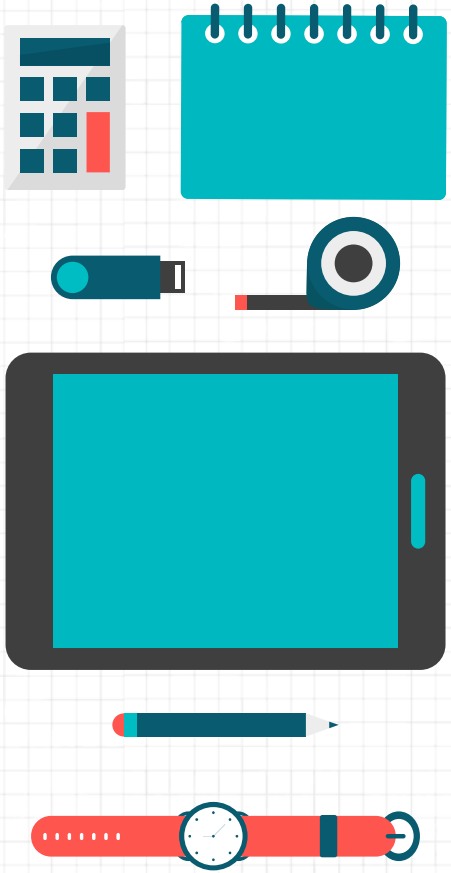


Ths. Phạm Minh Hoàng

Nội dung



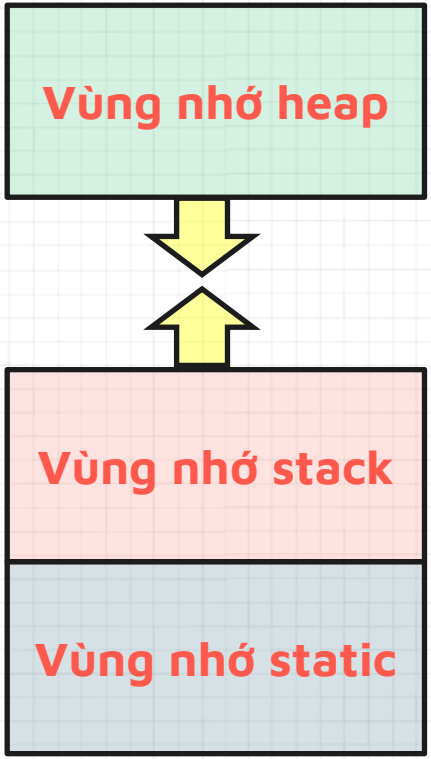
- Phân loại biến
- Quản lý bộ nhớ
- Ứng dụng
- Các lỗi thường gặp



01

Phân loại biến

Vùng nhớ stack – vùng nhớ heap



- Lưu các biến cấp phát động
- Cấp phát và giải phóng vùng nhớ bằng các hàm tường minh

- Lưu các biến cục bộ và tham số của hàm
- Cấp phát khi gọi hàm
- Giải phóng khi hàm kết thúc

- Lưu các câu lệnh thực thi
- Biến toàn cục
- Biến tĩnh

Phân loại Biến

- Thuộc tính của biến
  - Tên (name)
  - Kiểu dữ liệu (data type)
  - Giá trị (value)
  - Địa chỉ (address)
  - Tầm vực (scope): đoạn chương trình có thể truy xuất hợp lệ đến biến
  - Thời gian sống (lifetime): thời gian biến tồn tại từ lúc được cấp phát vùng nhớ đến khi vùng nhớ được giải phóng

Phân loại Biến

- Phân loại biến
  - Biến khai báo trong vùng nhớ static
  - Biến khai báo trong vùng nhớ stack
  - Biến khai báo trong vùng nhớ heap

Biến khai báo trong vùng nhớ static

- Biến toàn cục
  - Cú pháp

```
<kiểu dữ liệu> <tên biến>;
```

  - Khai báo tại vị trí nằm ngoài tất cả khối lệnh
  - Tầm vực: truy xuất biến toàn cục tại mọi điểm trong chương trình
  - Thời gian sống: từ khi thực thi câu khai báo tới khi chương trình kết thúc

Biến khai báo trong vùng nhớ static

- Biến tĩnh
  - Cú pháp

```
static <kiểu dữ liệu> <tên biến>;
```

  - Giá trị mặc định của biến tĩnh luôn là 0
  - Tầm vực: tùy theo vị trí khai báo
  - Thời gian sống: từ khi thực thi câu khai báo tới khi chương trình kết thúc

Biến khai báo trong vùng nhớ static

- Ví dụ

```
int a = 10;
void func(){
    static int b;
    ++b;
    a = a + 2;
    cout << b;
}
void main(){
    a = a*2;
    func();
    cout << a;
    func();
    cout << a;
}
```

Biến khai báo trong vùng nhớ stack

- Biến cục bộ
  - Khai báo cục bộ trong hàm hoặc trong các khối lệnh
  - Tầm vực: trong khối lệnh mà biến được khai báo
  - Kích thước vùng nhớ phải được xác định cụ thể
  - Biến được cấp phát vùng nhớ khi câu lệnh khai báo được gọi
  - Vùng nhớ của biến được tự động giải phóng khi hàm hoặc khối lệnh thực thi của biến kết thúc
  - Thời gian sống: từ câu khai báo thực thi đến khi khối lệnh kết thúc

```
void func(int b){
    int a = b + 1;
}
```

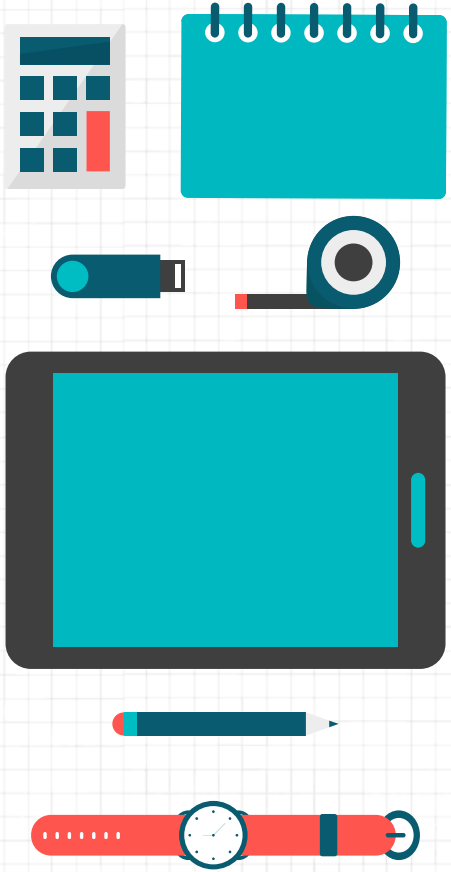
Biến khai báo trong vùng nhớ heap

- Biến cấp phát động
  - Kích thước vùng nhớ tùy biến theo khai báo của người dùng
  - Vùng nhớ được cấp phát và giải phóng bằng các câu lệnh tường minh
  - Tầm vực: theo vị trí của câu lệnh khai báo
  - Thời gian sống: từ khi gọi câu lệnh cấp phát đến khi gọi câu lệnh giải phóng vùng nhớ

Bài tập

```
const int LIMIT = 60;
void add(int x, int y){
    static int total;
    total = x + y
    if(total > LIMIT)
        total = 0;
    return total
}
void main(){
    int a = 32, b = 45;
    int s = add(a, b);
}
```

- Xác định
  - Thời gian sống, tầm vực của LIMIT
  - Thời gian sống, tầm vực biến total
  - Thời gian sống, tầm vực biến x, y
  - Thời gian sống, tầm vực biến a, b



02

Quản lý bộ nhớ

Quản lý bộ nhớ

- Quản lý bộ nhớ
  - Cấp phát vùng nhớ
    - Trong C: malloc/calloc/realloc
    - Trong C++: new/new []
  - Giải phóng vùng nhớ
    - Trong C: free
    - Trong C++: delete/delete []
  - Các thao tác trên vùng nhớ
    - Copy: memcpy
    - Move: memmove

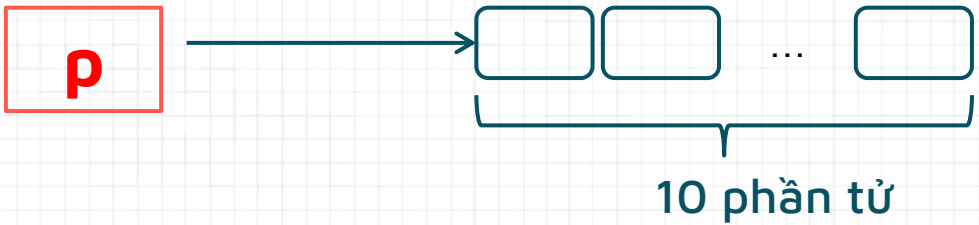
Cấp phát vùng nhớ động trong C

- Trong C, dùng câu lệnh cấp phát malloc
  - Include <stdlib.h>
  - Cú pháp `void* malloc(size_t size);`
  - Ý nghĩa: cấp phát vùng nhớ có kích thước size byte
  - Trả về:
    - Nếu thành công: trả về con trỏ đến vùng nhớ đang cấp phát
    - Nếu thất bại: trả về NULL

Cấp phát vùng nhớ động trong C

- Ví dụ

```
#include <stdlib.h>
void main(){
    int* p = (int*)malloc(10*sizeof(int));
    if(p == NULL)
        cout << "Cap phat khong thanh cong!!";
}
```



Cấp phát vùng nhớ động trong C

- Trong C, dùng câu lệnh cấp phát calloc
  - Include <stdlib.h>
  - Cú pháp
```
void* calloc(size_t num, size_t size);
```
  - Ý nghĩa: cấp phát vùng nhớ gồm num phần tử nằm liên tiếp nhau, mỗi phần tử có kích thước size byte
  - Trả về: Nếu thành công: trả về con trỏ đến vùng nhớ đang cấp phát. Nếu thất bại: trả về NULL
- Ví dụ

```
int* p = (int*)calloc(10, sizeof(int));
if(p == NULL)
    cout << "Cap phat khong thanh cong!!";
```

Cấp phát vùng nhớ động trong C

- Trong C, dùng câu lệnh cấp phát lại realloc
  - Include <stdlib.h>
  - Cú pháp `void* realloc(void* ptr, size_t size)`
  - Ý nghĩa: cấp phát lại vùng nhớ có kích thước size byte do ptr trỏ đến
    - Nếu ptr == NULL, cấp phát vùng mới
    - Nếu size == 0, giải phóng vùng nhớ do ptr trỏ đến
  - Trả về: Nếu thành công: trả về con trỏ đến vùng nhớ đang cấp phát. Nếu thất bại: trả về NULL

Cấp phát vùng nhớ động trong C

- Ví dụ

```
#include <stdlib.h>
void main(){
    int* p = (int*)malloc(10*sizeof(int));
    p = (int*)realloc(p, 20*sizeof(int));
    if(p == NULL)
        cout << "Cap phat khong thanh cong!!";
}
```

Cấp phát vùng nhớ động trong C++

- Trong C++, dùng câu lệnh cấp phát new
  - Cú pháp

```
<type>* <pointer> = new <type>[size];
<type>* <pointer> = new type;
```

  - Ý nghĩa: cấp phát lại vùng nhớ gồm size phần tử có cùng kiểu dữ liệu type
  - Trả về:
    - Nếu thành công: trả về con trỏ đến vùng nhớ đang cấp phát
    - Nếu thất bại: trả về NULL
- Ví dụ

```
int* p = new int[10];
int* q = new int;
```

Giải phóng vùng nhớ trong C

- Trong C, dùng câu lệnh giải phóng vùng nhớ free
  - Include <stdlib.h>
  - Cú pháp `void free(void* ptr);`
  - Ý nghĩa: giải phóng vùng nhớ mà ptr đang trỏ tới
  - Trả về: không có
- Ví dụ

```
int* p = malloc(10*sizeof(int));
free(p);
p = NULL;
```

Giải phóng vùng nhớ trong C++

- Trong C++, dùng câu lệnh giải phóng vùng nhớ delete
  - Cú pháp

```
void delete[] ptr;
void delete ptr;
```

  - Ý nghĩa: giải phóng vùng nhớ mà ptr đang trỏ tới
  - Trả về: không có
- Ví dụ

```
int* p = new int[20];
delete[] p;
p = NULL;
int* q = new int;
delete q;
q = NULL;
```

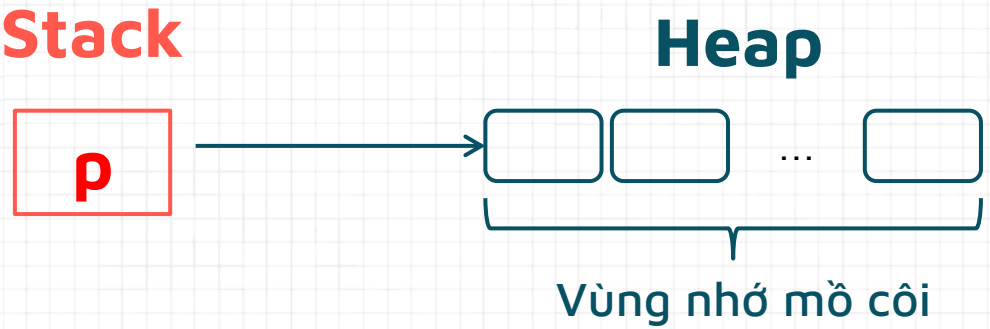
Giải phóng vùng nhớ

- Cấp phát vùng nhớ bằng malloc/calloc/realloc thì giải phóng vùng nhớ bằng free
- Cấp phát vùng nhớ cho một phần tử bằng new thì giải phóng vùng nhớ bằng delete
- Cấp phát vùng nhớ gồm nhiều phần tử bằng new[] thì giải phóng vùng nhớ bằng delete[]
- Giải phóng vùng nhớ xong phải gán lại con trỏ bằng NULL

Giải phóng vùng nhớ

- Ví dụ: chuyện gì xảy ra nếu không giải phóng vùng nhớ

```
#include <stdlib.h>
void main(){
    int* p = (int*)malloc(10*sizeof(int));
    if(p == NULL)
        cout << "Cap phat khong thanh cong!!";
}
```



Các thao tác trên vùng nhớ

- Trong thư viện <memory.h> hoặc <string.h>:
  - Hàm memset: gán giá trị mặc định cho vùng nhớ

```
void* memset(void* ptr, int ch, int n);
```

  - Hàm memcpy: sao chép nội dung giữa 2 vùng nhớ

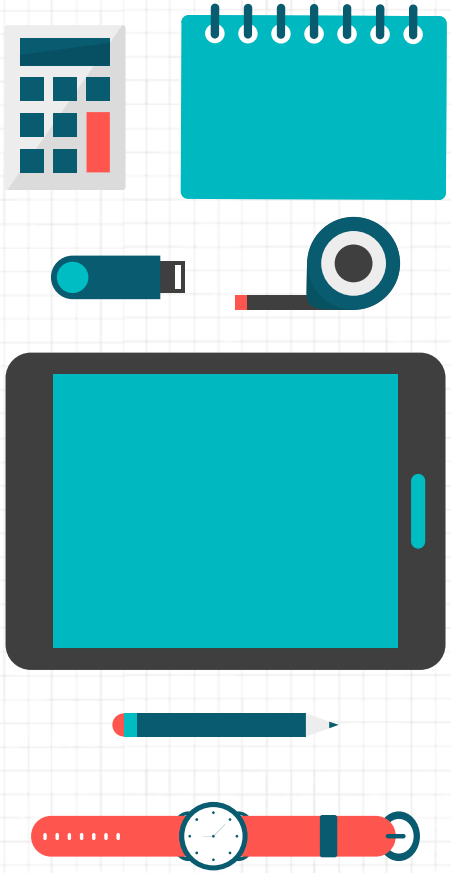
```
void* memcpy(void* dest, const int* src, int n);
```

  - Hàm memmove: di chuyển nội dung giữa 2 vùng nhớ

```
void* memmove(void* dest, const int* src, int n);
```

Bài tập

- Cho người dùng nhập số nguyên dương n
  - Viết hàm cho người dùng nhập mảng động gồm n phần tử
  - Sắp xếp mảng theo thứ tự tăng dần
  - Tính số lần xuất hiện của mỗi giá trị trong mảng
  - Sắp xếp các giá trị của mảng giảm dần theo số lần xuất hiện



03

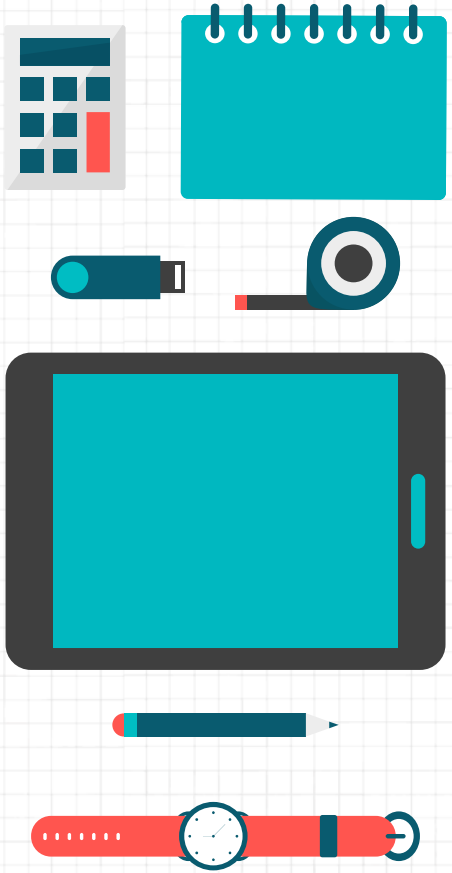
Ứng dụng

Mảng động 1 chiều

- Cài đặt các hàm xử lý trên mảng động 1 chiều các số nguyên tương tự cấu trúc vector trong thư viện std
  - Tạo mảng động a gồm n số nguyên
  - Giải phóng vùng nhớ cho mảng động a
  - Nhập/xuất mảng động a gồm n số nguyên
  - Lấy phần tử tại vị trí index của mảng động a
  - Thêm 1 phần tử x vào trước/sau mảng a
  - Xóa phần tử tại vị trí index của mảng a

Mảng động 2 chiều

- Cài đặt các hàm xử lý trên mảng động 2 chiều các số nguyên
  - Tạo mảng động a gồm m dòng, n cột
  - Nhập/xuất mảng động a gồm m dòng, n cột
  - Giải phóng vùng nhớ cho mảng động a
  - Lấy phần tử tại vị trí (i, j) của mảng động a
  - Thêm 1 dòng/cột vào trước/sau mảng a
  - Xóa 1 dòng/cột tại vị trí index của mảng a



04

Các lỗi thường gặp

Rò rỉ vùng nhớ (memory leak)

- Rò rỉ vùng nhớ là lỗi không giải phóng vùng nhớ đã được cấp phát động trước đó
- Ví dụ

```
void func(){
    int* p = new int[10];
    p = new int[3];
    delete[] p;
}

void main(){
    int n = 100;
    for(int i = 0; i < n; i++){
        func();
    }
}
```

Rò rỉ vùng nhớ (memory leak)

- Cho biết đoạn code sau lỗi ở đâu

```
struct A{
    int* ptr;
    int n;
}
void main(){
    A* x = new A;
    x->n = 10;
    x->ptr = new int[x.n];
    delete x;
}
```

Con trỏ treo (dangling pointer)

- Con trỏ treo là con trỏ trỏ đến vùng nhớ đã bị thu hồi
- Ví dụ

```
void main(){
    int* p = new int[10];
    int* q = p;
    delete[] p;
}
```

Con trỏ treo (dangling pointer)

- Ví dụ

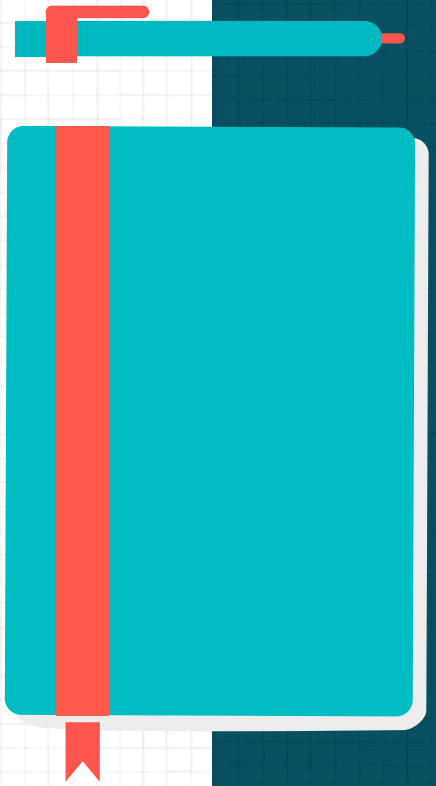
```
int* func(){
    int p = 10;
    int* q = &p;
    return q;
}

void main(){
    int* ptr = func();
}
```

Bài tập 1

- Viết hàm cho mảng động 1 chiều
  - Tìm các phần tử phân biệt trong mảng
  - Liệt kê số lần xuất hiện các phần tử phân biệt trong mảng
  - Tìm các mảng con tăng dần liên tục trong mảng
  - Tìm tất cả mảng con dài nhất gồm toàn số lẻ

Bài tập 2

- Viết hàm cho mảng động 2 chiều
  - Lấy mảng con số dòng k, số cột l, tại vị trí (i, j)
  - Lấy mảng con kích thước (k, l) sao cho tổng các phần tử trong mảng con là lớn nhất
  - Sắp xếp lại các dòng theo thứ tự tăng dần tổng các phần tử trên dòng



The End!

Do you have any questions?