



NHẬP MÔN LẬP TRÌNH

Đặng Bình Phương
dbphuong@fit.hcmuns.edu.vn

QUẢN LÝ BỘ NHỚ

1



Nội dung

- 1 Chuyển đổi kiểu (ép kiểu)
- 2 Cấu trúc CT C trong bộ nhớ
- 3 Cấp phát bộ nhớ động
- 4 Các thao tác trên khối nhớ

NMLT - Quản lý bộ nhớ

2



Nhu cầu chuyển đổi kiểu

- ❖ Mọi đối tượng dữ liệu trong C đều có kiểu xác định
 - Biến có kiểu **char**, **int**, **float**, **double**, ...
 - Con trỏ trỏ đến kiểu **char**, **int**, **float**, **double**, ...
- ❖ Xử lý thế nào khi gặp một biểu thức với nhiều kiểu khác nhau?
 - C **tự động** chuyển đổi kiểu (ép kiểu).
 - **Người sử dụng** **tự** chuyển đổi kiểu.

NMLT - Quản lý bộ nhớ

3



Chuyển đổi kiểu tự động

- ❖ Sự tăng cấp (kiểu dữ liệu) trong biểu thức
 - Các thành phần cùng kiểu
 - Kết quả là **kiểu chung**
 - Ví dụ: $\text{int} / \text{int} \rightarrow \text{int}$, $\text{float} / \text{float} \rightarrow \text{float}$
 - Các thành phần khác kiểu
 - Kết quả là **kiểu bao quát nhất**
 - $\text{char} < \text{int} < \text{long} < \text{float} < \text{double}$
 - Ví dụ: $\text{int} / \text{float} \rightarrow \text{float}$, $\text{float} / \text{float} \rightarrow \text{float}$, ...
 - Lưu ý, chỉ chuyển đổi tạm thời (nội bộ).

NMLT - Quản lý bộ nhớ

4



Chuyển đổi kiểu tự động

❖ Phép gán <BT về trái> = <BT về phải>;

- BT ở về phải luôn được tăng cấp (hay giảm cấp) **tạm thời** cho giống kiểu với BT ở về trái.

```
int i;
float f = 1.23;
```

```
i = f;      // → f tạm thời thành int
f = i;      // → i tạm thời thành float
```

- Có thể làm mất tính chính xác của số nguyên khi chuyển sang số thực → hạn chế!

```
int i = 3;
float f;
f = i;      // → f = 2.999995
```

NMLT - Quản lý bộ nhớ

5



Chuyển đổi tường minh (ép kiểu)

❖ Ý nghĩa

- Chủ động chuyển đổi kiểu (tạm thời) nhằm tránh những kết quả sai lầm.

❖ Cú pháp

```
(<kiểu chuyển đổi>)<biểu thức>
```

❖ Ví dụ

```
int x1 = 1, x2 = 2;
float f1 = x1 / x2;           // → f1 = 0.0
float f2 = (float)x1 / x2;    // → f2 = 0.5
float f3 = (float)(x1 / x2);  // → f3 = 0.0
```

NMLT - Quản lý bộ nhớ

6



Cấp phát bộ nhớ tĩnh và động

❖ Cấp phát tĩnh (static memory allocation)

- Khai báo biến, cấu trúc, mảng, ...
- Bắt buộc phải biết trước cần bao nhiêu bộ nhớ lưu trữ → tốn bộ nhớ, không thay đổi được kích thước, ...

❖ Cấp phát động (dynamic memory allocation)

- Cần bao nhiêu cấp phát bấy nhiêu.
- Có thể giải phóng nếu không cần sử dụng.
- Sử dụng vùng nhớ ngoài chương trình (cả bộ nhớ ảo virtual memory).

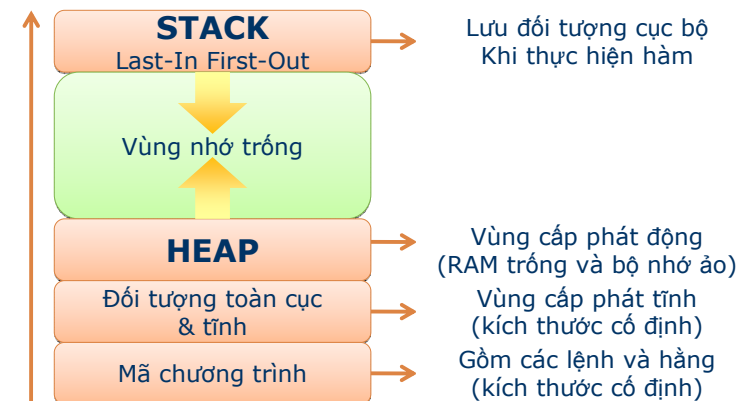
NMLT - Quản lý bộ nhớ

7



Cấu trúc một CT C trong bộ nhớ

- ❖ Toàn bộ tập tin chương trình sẽ được nạp vào bộ nhớ tại vùng nhớ còn trống, gồm 4 phần:



NMLT - Quản lý bộ nhớ

8



Cấp phát bộ nhớ động

❖ Thuộc thư viện `<stdlib.h>` hoặc `<alloc.h>`

- `malloc`
- `calloc`
- `realloc`
- `free`

❖ Trong C++

- `new`
- `delete`



Cấp phát bộ nhớ động

`void *malloc(size_t size)`



Cấp phát trong HEAP một vùng nhớ **size** (bytes)
size_t thay cho unsigned (trong `<stddef.h>`)



- ◆ Con trỏ đến vùng nhớ mới được cấp phát
- ◆ **NULL** nếu không đủ bộ nhớ



```
int *p = (int *)malloc(10*sizeof(int));
if (p == NULL)
    printf("Không đủ bộ nhớ! ☹");
```



Cấp phát bộ nhớ động

`void *calloc(size_t num, size_t size)`



Cấp phát vùng nhớ gồm **num** phần tử trong HEAP, mỗi phần tử kích thước **size** (bytes)



- ◆ Con trỏ đến vùng nhớ mới được cấp phát
- ◆ **NULL** nếu không đủ bộ nhớ



```
int *p = (int *)calloc(10, sizeof(int));
if (p == NULL)
    printf("Không đủ bộ nhớ! ☹");
```



Cấp phát bộ nhớ động

`void *realloc(void *block, size_t size)`



Cấp phát lại vùng nhớ có kích thước **size** do **block** trỏ đến trong vùng nhớ HEAP.
block == NULL → sử dụng `malloc`
size == 0 → sử dụng `free`



- ◆ Con trỏ đến vùng nhớ mới được cấp phát
- ◆ **NULL** nếu không đủ bộ nhớ



```
int *p = (int *)malloc(10*sizeof(int));
p = (int *)realloc(p, 20*sizeof(int));
if (p == NULL)
    printf("Không đủ bộ nhớ! ☹");
```



Cấp phát bộ nhớ động

void *free(void *ptr)



Giải phóng vùng nhớ do **ptr** trỏ đến, được cấp bởi các hàm malloc(), calloc(), realloc(). Nếu ptr là NULL thì không làm gì cả.



♦ Không có



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



Cấp phát bộ nhớ động

<pointer_to_datatype> = new <datatype>[size]



Cấp phát vùng nhớ có kích thước sizeof(<datatype>)*size trong HEAP



♦ Con trỏ đến vùng nhớ mới được cấp phát
♦ **NULL** nếu không đủ bộ nhớ



```
int *a1 = (int *)malloc(sizeof(int));
int *a2 = new int;
int *p1 = (int *)malloc(10*sizeof(int));
int *p2 = new int[10];
```



Cấp phát bộ nhớ động

delete []<pointer_to_datatype>



Giải phóng vùng nhớ trong HEAP do <pointer_to_datatype> trỏ đến (được cấp phát bằng new)



♦ Không có!



```
int *a = new int;
delete a;
int *p = new int[10];
delete []p;
```



Cấp phát bộ nhớ động

❖ Lưu ý

- Không cần kiểm tra con trỏ có NULL hay không trước khi free hoặc delete.
- Cấp phát bằng **malloc**, **calloc** hay **realloc** thì giải phóng bằng **free**, cấp phát bằng **new** thì giải phóng bằng **delete**.
- Cấp phát bằng **new** thì giải phóng bằng **delete**, cấp phát mảng bằng **new []** thì giải phóng bằng **delete []**.



Thao tác trên các khối nhớ

❖ Thuộc thư viện <string.h>

- **memset** : gán giá trị cho tất cả các byte nhớ trong khối.
- **memcpy** : sao chép khối.
- **memmove** : di chuyển thông tin từ khối này sang khối khác.



Thao tác trên các khối nhớ

void *memset(void *dest, int c, size_t count)



Gán **count** (bytes) đầu tiên của vùng nhớ mà **dest** trỏ tới bằng giá trị **c** (từ 0 đến 255). Thường dùng cho vùng nhớ kiểu char còn vùng nhớ kiểu khác thường đặt giá trị zero.



♦ dest



```
char buffer[] = "Hello world";
printf("Trước khi memset: %s\n", buffer);
memset(buffer, '*', strlen(buffer));
printf("Sau khi memset: %s\n", buffer);
```



Thao tác trên các khối nhớ

void *memcpy(void *dest, void *src, size_t count)



Sao chép chính xác **count** byte từ khối nhớ **src** vào khối nhớ **dest**. Nếu hai khối nhớ đè lên nhau, hàm sẽ làm việc không chính xác.



♦ dest



```
char src[] = "*****";
char dest[] = "0123456789";
memcpy(dest, src, 5);
memcpy(dest + 3, dest + 2, 5);
```



Thao tác trên các khối nhớ

void *memmove(void *dest, void *src, size_t count)



Sao chép chính xác **count** byte từ khối nhớ **src** vào khối nhớ **dest**. Nếu hai khối nhớ đè lên nhau, hàm vẫn thực hiện chính xác.



♦ dest



```
char src[] = "*****";
char dest[] = "0123456789";
memmove(dest, src, 5);
memmove(dest + 3, dest + 2, 5);
```



Bài tập lý thuyết

- ❖ **Bài 1:** Tại sao cần phải giải phóng khối nhớ được cấp phát động?
- ❖ **Bài 2:** Điều gì xảy ra nếu ta thêm một phần tử vào mảng đã được cấp phát động trước đó mà không cấp lại bộ nhớ?
- ❖ **Bài 3:** Ưu điểm của việc sử dụng các hàm thao tác khối nhớ? Ta có thể sử dụng một vòng lặp kết hợp với một câu lệnh gán để khởi tạo hay sao chép các byte nhớ hay không?



Bài tập lý thuyết

- ❖ **Bài 4:** Ta thường dùng phép ép kiểu trong những trường hợp nào?
- ❖ **Bài 5:** Giả sử c kiểu char, i kiểu int, l kiểu long và f kiểu float. Hãy xác định kiểu của các biểu thức sau:
 - $(c + i + l)$
 - $(i + 32)$
 - $(c + 'A')$
 - $(i + 32.0)$
 - $(100 + 1.0)$



Bài tập lý thuyết

- ❖ **Bài 6:** Việc cấp phát động nghĩa là gì?
- ❖ **Bài 7:** Cho biết sự khác nhau giữa malloc() và calloc()?
- ❖ **Bài 8:** Viết câu lệnh sử dụng hàm malloc() để cấp phát 1000 số kiểu long.
- ❖ **Bài 9:** Giống bài 7 nhưng dùng calloc()
- ❖ **Bài 10:** Cho biết sự khác nhau giữa memcpy và memmove
- ❖ **Bài 11:** Trình bày 2 cách khởi tạo mảng float data[1000]; với giá trị zero.



Bài tập lý thuyết

- ❖ **Bài 12:** Kiểm tra lỗi

```
void func()
{
    int number1 = 100, number2 = 3;
    float answer;
    answer = number1 / number2;
    printf("%d/%d=%f", number1, number2, answer);
}
```
- ❖ **Bài 13:** Kiểm tra lỗi

```
void *p;
p = (float *)malloc(sizeof(float));
*p = 1.23;
```