CON TRỎ VÀ CẤP PHÁT ĐỘNG

CĐR buổi học

- Sau khi học xong buổi học, sinh viên có khả năng:
  - Hiểu được về con trỏ và cấp phát động.
  - Áp dụng con trỏ trong cấp phát mảng.
  - Áp dụng con trỏ và tham số của hàm.
  - Áp dụng con trỏ và cấu trúc.

Nội dung

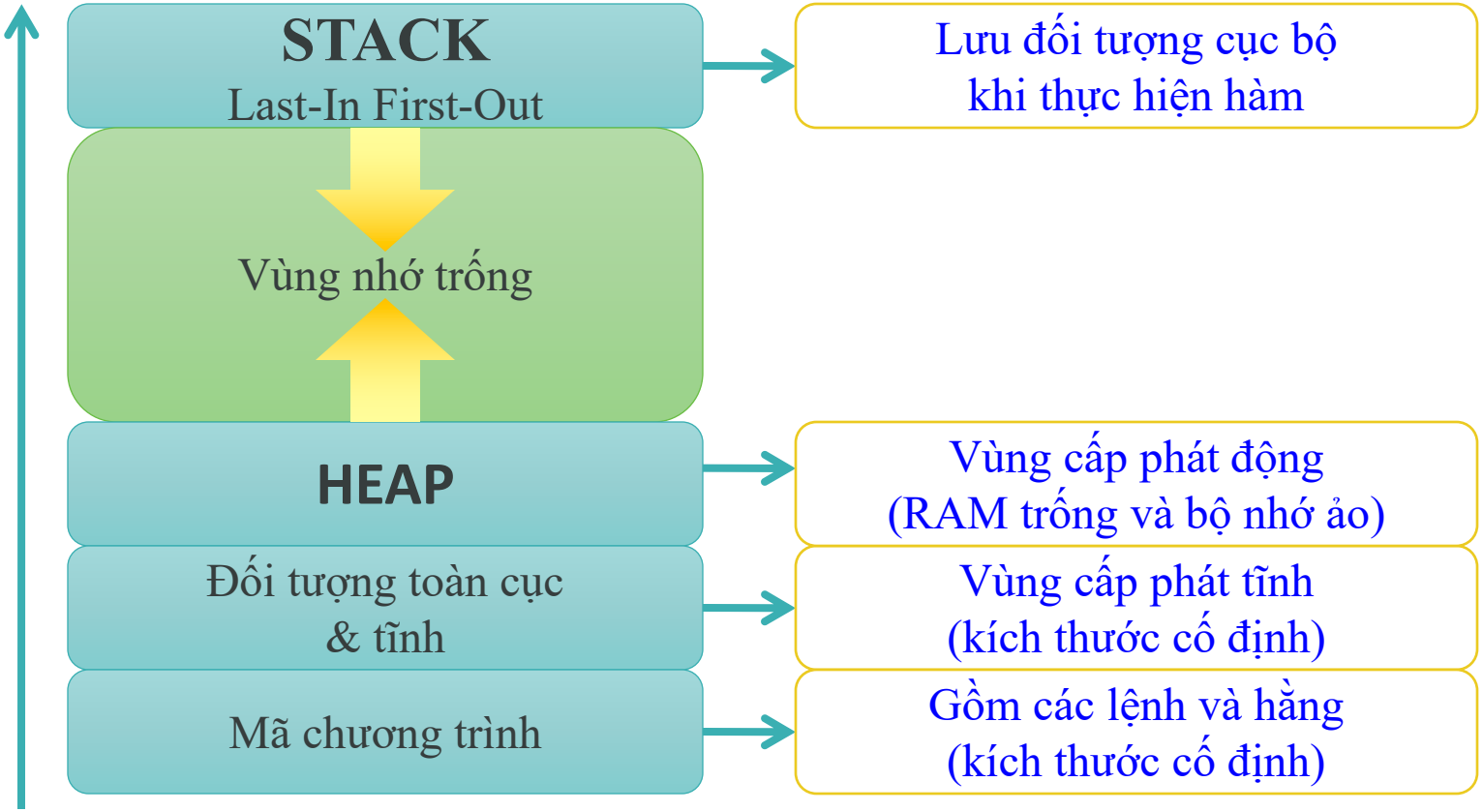
1. Cấp phát động
2. Cấp phát động mảng 1 chiều
3. Cấp phát động mảng 2 chiều
4. Con trỏ và hàm số
5. Con trỏ và cấu trúc
6. Một số vấn đề mở rộng

1. Cấp phát động

- Cấp phát bộ nhớ tĩnh (static memory allocation)
  - Khai báo biến, cấu trúc, mảng, …
  - Bắt buộc phải biết trước cần bao nhiều 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).

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:



Cấp phát bộ nhớ

- Cấp phát bộ nhớ
  - Trong C: Hàm **malloc, calloc, realloc** (<stdlib.h> hoặc <alloc.h>)
  - Trong C++: Toán tử **new**
- Giải phóng bộ nhớ
  - Trong C: Hàm **free**
  - Trong C++: Toán tử **delete**

Biến cấp phát động và Biến tự động

- **Biến cục bộ**
  - Khai báo bên trong định nghĩa hàm
  - Sinh ra khi hàm được gọi
  - Hủy đi khi hàm kết thúc
    - Thường gọi là biến tự động nghĩa là được trình biên dịch quản lý một cách tự động
- **Biến cấp phát động**
  - Sinh ra bởi cấp phát động
  - Sinh ra và hủy đi khi chương trình đang chạy
  - Biến cấp phát động hay Biến động là biến con trỏ trước khi sử dụng được cấp phát bộ nhớ.

Toán tử new

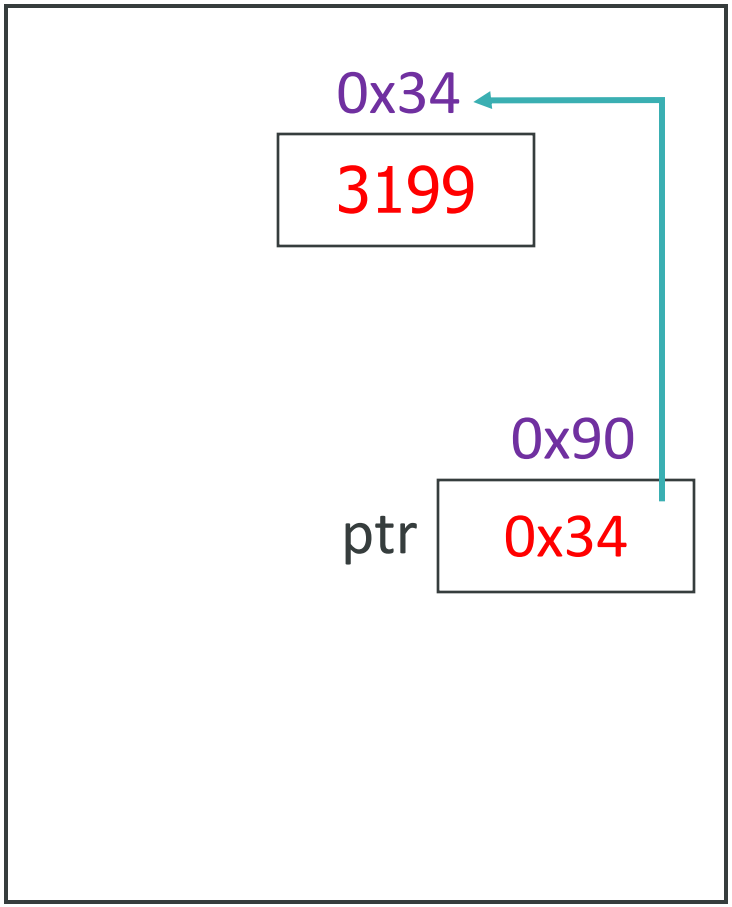
- Vì con trỏ có thể tham chiếu tới biến nhưng không thực sự cần phải có định danh cho biến đó.
- Có thể cấp phát động cho biến con trỏ bằng toán tử new. Toán tử new sẽ tạo ra biến "không tên" cho con trỏ trỏ tới.
- Cú pháp: **<type> *<pointerName> = new <type>**

Ví dụ: `int *ptr = new int;`

  - Tạo ra một biến "không tên" và gán ptr trỏ tới nó
  - Có thể làm việc với biến "không tên" thông qua `*ptr`

Kiểm tra việc cấp phát có thành công không

```
#include <iostream>

using namespace std;


int main() {

	int *p = new int;

	if (p == NULL) {

		cout << "Error: Khong
		du bo nho.\n";

		exit(1);

	}

	*p = 3199;

}
```



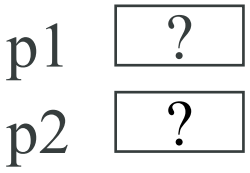
Khởi tạo giá trị trong cấp phát động

- Cú pháp: **<type> pointer = new <type> (value)**
- Ví dụ:

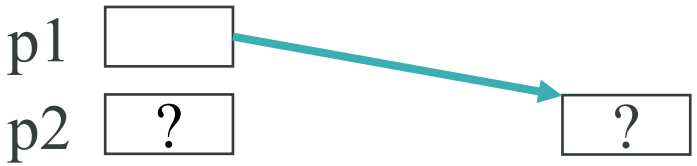
```
#include <iostream>
using namespace std;

int main() {
	int *p;
	p = new int(99); // initialize with 99
	cout << *p; // displays 99
	return 0;
}
```

Ví dụ

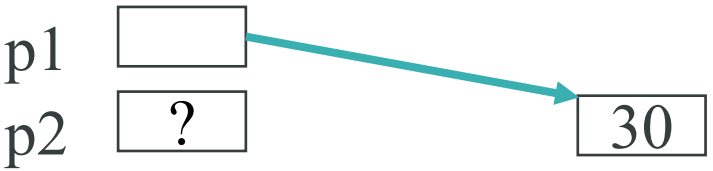
1. `int *p1, *p2;`



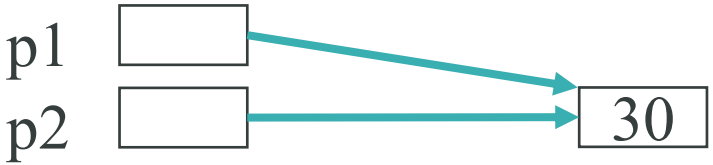
2. `int *p1 = new int;`



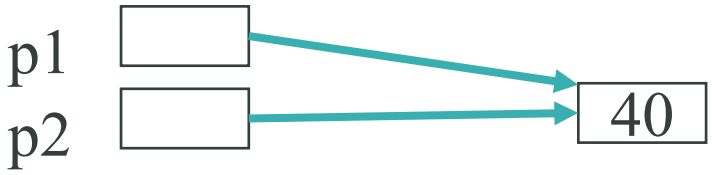
3. `*p1 = 30;`



4. `p2 = p1;`



5. `*p2 = 40;`



6. `p1 = new int;`

7. `*p1 = 50;`

Toán tử delete

- Toán tử **delete** dùng để giải phóng vùng nhớ trong HEAP do con trỏ trỏ tới (con trỏ được cấp pháp bằng toán tử new). Cú pháp:

delete <pointerName>;

- Ghi chú: Sau khi gọi toán tử **delete** thì con trỏ vẫn trỏ tới vùng nhớ trước khi gọi hàm delete. Ta gọi là "con trỏ lạc". Ta vẫn có thể gọi tham chiếu trên con trỏ, tuy nhiên:
  - Kết quả không lường trước được
  - Thường là nguy hiểm

⇒ Hãy tránh con trỏ lạc bằng cách gán con trỏ bằng **NULL** sau khi delete.

- Ví dụ:

```
delete pointer;
pointer = NULL;
```

Từ khóa typedef

- Từ khóa typedef dùng để định nghĩa 1 tên mới hay gọi là một biệt danh (alias) cho tên kiểu dữ liệu có sẵn.

Ví dụ: `typedef int SONGUYEN;`

Các khai báo sau tương đương:

```
int a;
SONGUYEN a;
```

Định nghĩa kiểu dữ liệu con trỏ

- Có thể đặt tên cho kiểu dữ liệu con trỏ
- Để có thể khai báo biến con trỏ như các biến khác
  - Loại bỏ * trong khai báo con trỏ

Ví dụ: `typedef int* IntPtr;`

- Định nghĩa một tên khác cho kiểu dữ liệu con trỏ
- Các khai báo sau tương đương:

```
IntPtr p;

int *p;
```

Con trỏ và hàm

- Con trỏ là kiểu dữ liệu hoàn chỉnh có thể dùng nó như các kiểu khác
- Con trỏ có thể là tham số của hàm
- Có thể là kiểu trả về của hàm

Ví dụ: `int* findOtherPointer(int* p);`

Hàm này khai báo:

- Có tham số kiểu con trỏ trỏ tới int
- Trả về biến con trỏ trỏ tới int

Ví dụ

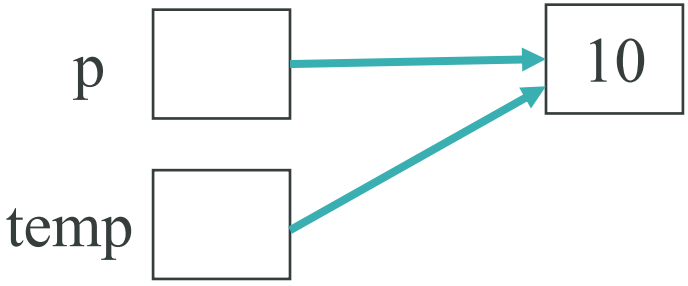
```
typedef int *IntPointer;
void Input(IntPointer temp) {
    *temp = 20;
    cout << "Trong ham goi *temp = " << *temp << endl;
}
int main() {
    IntPointer p = new int;
    *p = 10;
    cout << "Truoc khi goi ham, *p = " << *p << endl;
    Input(p);
    cout << "Sau khi ket thuc ham, *p = " << *p << endl;
}
```

```
Truoc khi goi ham, *p = 10
Trong ham goi *temp = 20
Sau khi ket thuc ham, *p = 20
```

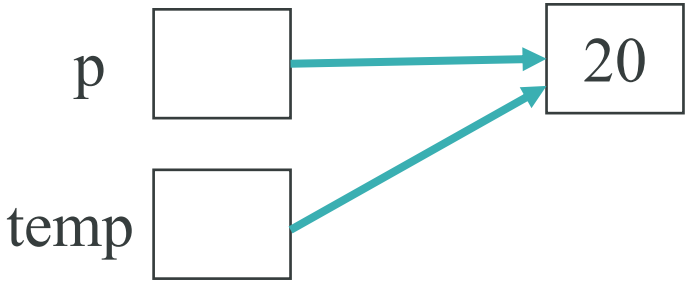
Ví dụ

1. Trước khi gọi hàm Input

2. Giá trị của p sẽ được truyền vào temp



3. Thay đổi giá trị *temp



4. Sau khi kết thúc gọi hàm Input

Bài tập

- Viết hàm cấp phát và nhập giá trị cho 1 con trỏ theo 2 cách.

Bài giải

```cpp
// Cách 1
#include <iostream>
using namespace std;

typedef int *IntPointer;
IntPointer Input() {
    IntPointer temp = new int;
    *temp = 20;
    return temp;
}
int main() {
    IntPointer p;
    p = Input();
    cout << "Sau khi ket thuc
    ham, *p = " << *p << endl;
}
```

```cpp
// Cách 2
#include <iostream>
using namespace std;

typedef int *IntPointer;
IntPointer Input(IntPointer &temp) {
      temp = new int;
      *temp = 20;
      return temp;
}
int main() {
      IntPointer p;
      Input(p);
      cout << "Sau khi ket thuc ham,
      *p = " << *p << endl;
}
```

2. Cấp phát động và mảng 1 chiều

Nhắc lại

- Mảng lưu trong các ô nhớ liên tiếp trong bộ nhớ máy tính
- Biến mảng tham chiếu tới phần tử đầu tiên
- Biến mảng là một biến hằng con trỏ

Nhắc lại

- Ví dụ:

```
int a[10];
typedef int* IntPtr;
IntPtr p;
```

⇒ a và p là các biến con trỏ.

- Phép gán hợp lệ `p = a;`
  - p bây giờ sẽ trỏ tới nơi a trỏ, tức là tới phần tử đầu tiên của mảng a
- Phép gán không hợp lệ `a = p;`
  - Bởi con trỏ mảng là con trỏ hằng.

Hạn chế của mảng chuẩn

- Hạn chế của mảng chuẩn
  - Bắt buộc phải biết trước cần bao nhiêu bộ nhớ lưu trữ => tốn bộ nhớ, không thay đổi được kích thước, …

  ⇒ **Dùng Mảng động**

- Mảng động
  - **Kích thước không xác định ở thời điểm lập trình**
  - **Mà xác định khi chạy chương trình**

Tạo mảng động bằng toán tử new

- Cấp phát động cho biến con trỏ
- Sau đó dùng con trỏ như mảng chuẩn
- Cú pháp:

<type> <pointer> = new <type> [<number_of_elements>]

Ví dụ:

```
typedef double * doublePtr;
doublePtr d;
d = new double[10];
```

⇒ Tạo biến mảng cấp phát động d có 10 phần tử, kiểu cơ sở là double.

Xóa mảng động

- Dùng toán tử `delete[]` để xóa mảng động.

Ví dụ:

```
double *d = new double[10];
//... Processing
delete[] d;
```

⇒ Giải phóng tất cả vùng nhớ của mảng động này

⇒ Cặp ngoặc vuông báo hiệu có mảng

⇒ Nhắc lại: d vẫn trỏ tới vùng nhớ đó. Vì vậy sau khi `delete`, cần gán `d = NULL;`

Bài tập

Hãy viết chương trình tạo mảng 1 chiều có n phần tử bằng 2 cách:

- Cách 1: Bằng mảng chuẩn
- Cách 2: Bằng cấp phát động

Lời giải

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

int main() {
	int arr[10];
	return 0;
}
```

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

int main() {
	int *arr;
	arr = new int[10];
	return 0;
}
```

Hàm trả về kiểu mảng

- Ta không được phép trả về kiểu mảng trong hàm.

Ví dụ:

```
int[] someFunction(); // Không hợp lệ!
```

- Có thể thay bằng trả về con trỏ tới mảng có cùng kiểu cơ sở:

```
int* someFunction(); // Hợp lệ!
```

Bài tập

- Hãy viết HÀM tạo mảng 1 chiều có n phần tử bằng cấp phát động.
- Viết hàm xuất mảng 1 chiều đã tạo.
- Viết hàm đếm số phần tử âm trong mảng 1 chiều.

Lời giải

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

int* Input(int n){
	int *p;
	p = new int[n];
	for (int i = 0; i < n; i++){
		cin >> p[i];
	}
	return p;
}

int main() {
	int *arr, n;
	cout << "Nhap n: ";
	cin >> n;
	arr = Input(n);
}
```

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

void Input(int *&p, int n){
	p = new int[n];
	for (int i = 0; i < n; i++){
		cin >> p[i];
	}
}

int main() {
	int *arr, n;
	cout << "Nhap n: ";
	cin >> n;
	Input(arr, n);
}
```

Lời giải

```
// Hàm xuất mảng
void Output(int *p, int n) {
      cout << "\n Xuat mang 1 chieu: ";
      for (int i = 0; i < n; i++) {
            cout << p[i] << " " ;
      }
}
```

3. Mảng động 2 chiều

- Là mảng của mảng
- Sử dụng định nghĩa kiểu con trỏ giúp hiểu rõ hơn:

```
typedef int* IntArrayPtr;
IntArrayPtr *m = new IntArrayPtr[3];
```

⇒ Tạo ra mảng 3 con trỏ

⇒ Sau đó biến mỗi con trỏ này thành mảng 4 biến int

```
for (int i = 0; i < 3; i++)
    m[i] = new int[4];
```

⇒ Kết quả là mảng động 3 x 4

Bài tập

Tạo mảng 2 chiều bằng con trỏ.

Lời giải

```
int main() {
    int row=2, col=3;

    // Cấp phát vùng nhớ
    int **p = new int*[row];
    if (p == NULL) exit(1);
    for (int i = 0; i < row; i++) {
        p[i] = new int[col];
        if (p[i] == NULL) exit(1);
    }


    // Giải phóng vùng nhớ
    for (int i = 0; i < row; i++) {
         delete[] p[i];
    }

    delete[] p;

    return 0;

}
```

4. Con trỏ và hàm số

- Tham số của hàm là 1 biến con trỏ
  - Trường hợp thay đổi giá trị của đối số

```
void Hoanvi(int *x, int *y)
{
        int z = *x;
        *x=*y;
        *y=z;
}
void main()
{
        int a=1,b=2;
        cout<<a<<b; // 1 2
        Hoanvi(&a,&b);
        cout<<a<<b; // 2 1
}
```

4. Con trỏ và hàm số

- Tham số của hàm là 1 biến con trỏ
  - Trường hợp không thay đổi giá trị của đối số

```
void Capphat(int *a)
{       a = new int[5];
        for(int i=0; i<5; i++)
        {       a[i]=i+1;
                cout<< a[i];
        }
}
void main()
{       int n=5;
        int *b = &n;
        cout<<*b; // 5
        Capphat(b); // 1 2 3 4 5
        cout<<*b; // 5
}
```

4. Con trỏ và hàm số

- Kiểu trả về của hàm là 1 con trỏ

```
int* GetArray()
{
        int *a = new int [5];
        for(int i=0; i<5; i++)
                a[i]=i+1;
        return a;
}
```

4. Con trỏ và cấu trúc

- Truy xuất các các thuộc tính dùng con trỏ:

  tên_biến_con_trỏ → tên_thuộc_tính

- Ví dụ cấu trúc phân số

```
struct PhanSo
{       int TuSo;
        int MauSo;
};
PhanSo x;
x.Tuso=1; x.MauSo=2;
PhanSo *p, *q;
p = &x;
P→Tuso=3; p→ MauSo=4; // x(3,4)

q = new PhanSo();
q->TuSo = 1;  // giống (*q).TuSo=1
q->MauSo = 2;// giống (*q).MauSo=2
```

4. Con trỏ và cấu trúc

- Cấu trúc đệ quy (tự trỏ)

```
struct PERSON
{
        char hoten[30];
        struct PERSON *father, *mother;
};

struct NODE
{
        int value;
        struct NODE *pNext;
};
```

Bài tập

- Bài 1: Tại sao cần phải giải phóng khối nhớ được cấp phát động?

⇒ Khối nhớ không tự giải phóng sau khi sử dụng nên sẽ làm giảm tốc độ thực hiện chương trình hoặc tràn bộ nhớ nếu tiếp tục cấp phát

- Bài 2: Điều gì xảy ra nếu ta nối thêm một số ký tự vào một chuỗi (được cấp phát động trước đó) mà không cấp phát lại bộ nhớ cho nó?

⇒ Nếu chuỗi đủ lớn để chứa thêm thông tin thì không cần cấp phát lại. Ngược lại phải cấp phát lại để có thêm vùng nhớ.

Bài tập

- Bài 3: Ta thường dùng phép ép kiểu trong những trường hợp nào?

⇒ Lấy phần nguyên của số thực hoặc lấy phần thực của phép chia hai số nguyên, …

- Bài 4: Giả sử c kiểu char, i kiểu int, l kiểu long. Hãy xác định kiểu của các biểu thức sau:
  - `(c + i + l)`
  - `(i + ‘A’)`
  - `(i + 32.0)`
  - `(100 + 1.0)`

Bài tập

- Bài 5: Việc cấp phát động nghĩa là gì?

⇒ Bộ nhớ được cấp phát động là bộ nhớ được cấp phát trong khi chạy chương trình và có thể thay đổi độ lớn vùng nhớ.

- Bài 6: Cho biết sự khác nhau giữa malloc và calloc?

⇒ malloc: cấp phát bố nhớ cho một đối tượng.

⇒ calloc: cấp phát bộ nhớ cho một nhóm đối tượng.

Bài tập

- Bài 7: Viết câu lệnh sử dụng hàm malloc để cấp phát 1000 số kiểu long.

⇒ `long *ptr;`

⇒ `ptr = (long *)malloc(1000 * sizeof(long));`

- Bài 8: Giống bài 7 nhưng dùng calloc

⇒ `long *ptr;`

⇒ `ptr = (long *)calloc(1000, sizeof(long));`

⇒ `ptr = (long *)calloc(sizeof(long), 1000);` !!!

Bài tập

- Bài 9: Kiểm tra kết quả

```
void func() {
    int n1 = 100, n2 = 3;
    float ketqua = n1 / n2;
    printf("%d / %d = %f", n1, n2, ketqua);
}
```

- Bài 10: Kiểm tra lỗi

```
int main() {
    void *p;
    p = (float *)malloc(sizeof(float));
    *p = 1.23;
}
```

Bài tập

- Bài 11: Cho biết kết quả chương trình sau. Giải thích tại sao ta có được kết quả này.

```
void hamf(int*a)
{       a= new int[5];
        for(int i=0; i<5; i++)
                a[i]=i+1;
}
void main()
{       int n=5;
        int *a= &n;
        cout<<“Giá trị *a = "<<*a;
        hamf(a);
        cout<<“Giá trị *a = "<<*a;
}
```

Bài tập

- Bài 12: Cho biết kết quả chương trình sau. Giải thích tại sao ta có được kết quả này.

```
void hamf(int*&a)
{       a= new int[5];
        for(int i=0; i<5; i++)
                a[i]=i+1;
}
void main()
{       int n=5;
        int *a= &n;
        cout<<“Giá trị *a = "<<*a;
        hamf(a);
        cout<<“Giá trị *a = "<<*a;
}
```

Bài tập bắt buộc (1/2)

1. Cho biết ý nghĩa của các khai báo và câu lệnh; Tìm lỗi sai trong đoạn code và giải thích (t.t) → xem các bài tập từ 1 đến 12 trong phần trước.
2. Viết chương trình nhập một dãy số hữu tỉ tùy ý (sử dụng con trỏ và sự cấp phát động), xuất ra dãy gồm tất cả các số nhỏ hơn 1 có trong dãy được nhập vào, tính tổng và tích của dãy số hữu tỉ.
3. Viết chương trình khai báo mảng hai chiều có 12x12 phần tử kiểu char. Gán ký tự 'X' cho mọi phần tử của mảng này. Sử dụng con trỏ đến mảng để in giá trị các phần tử mảng lên màn hình ở dạng lưới.

Bài tập bắt buộc (2/2)

4. Viết chương trình khai báo mảng 10 con trỏ đến kiểu float, nhận 10 số thực từ bàn phím, sắp xếp lại và in ra màn hình dãy số đã sắp xếp.
5. Chương trình cho phép người dùng nhập các dòng văn bản từ bàn phím đến khi nhập một dòng trống. Chương trình sẽ sắp xếp các dòng theo thứ tự alphabet rồi hiển thị chúng ra màn hình.
6. Làm lại các bài tập về ma trận dùng con trỏ

6. Vấn đề mở rộng

a) Các thao tác trên khối nhớ

b) Tham khảo cấp phát động bằng hàm `malloc`

6.a) 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.

6.a) Thao tác trên các khối nhớ (tt)

```
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.

Trả về: Con trỏ dest.

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

```
Truoc khi memset: Hello world
Sau khi memset: ***********
```

6.a) Thao tác trên các khối nhớ (tt)

```
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.

Trả về: Con trỏ dest.

```
char src[] = "******";
char dest[] = "0123456789";
memcpy(dest, src, 5);
memcpy(dest + 3, dest + 2, 5);
printf("dest: %s\n", dest);
```

```
dest: ******5689
```

6.a) Thao tác trên các khối nhớ (tt)

```
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.

Trả về: Con trỏ dest.

```
char src[] = "******";
char dest[] = "0123456789";
memmove(dest, src, 5);
memmove(dest + 3, dest + 2, 5);
printf("dest: %s\n", dest);
```

Kết quả đoạn code: dest: ******5689

6.b) Tham khảo cấp phát động bằng hàm malloc trong C

```
void *malloc(size_t size)
```

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

Trả về:
- Thành công: Con trỏ đến vùng nhớ mới được cấp phát.
- Thất bại: NULL (không đủ bộ nhớ).

```
int *p = (int *)malloc(sizeof(int));
// int *p = new int;

int *p = (int *)malloc(10 * sizeof(int));
// int *p = new int[10];
if (p == NULL) printf("Khong du bo nho!");
```

6.b) Tham khảo cấp phát động bằng hàm malloc (tt)

```
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)

Trả về:
- Thành công: Con trỏ đến vùng nhớ mới được cấp phát.
- Thất bại: NULL (không đủ bộ nhớ).

```
int *p = (int *)calloc(10, sizeof(int));
if (p == NULL)
    printf("Khong du bo nho!");
```

6.b) Tham khảo cấp phát động bằng hàm malloc (tt)

```
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

Trả về:
- Thành công: Con trỏ đến vùng nhớ mới được cấp phát.
- Thất bại: NULL (không đủ bộ nhớ).

```
int *p = (int *)malloc(10 * sizeof(int));
p = (int *)realloc(p, 20 * sizeof(int));
if (p == NULL)
      printf("Khong du bo nho!");
```

6.b) Tham khảo cấp phát động bằng hàm malloc (tt)

```
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.

Trả về: Không có.

```
int *p = (int *)malloc(10 *  sizeof(int));
free(p);
// Tương đương:
int *p = new int[10];
delete []p;
```

Bài tập

Tạo mảng 2 chiều bằng con trỏ.

Lời giải (sử dụng hàm malloc)

```
int main() {
      int m = 4, n = 4;
      int kt;
      int **a = (int **)malloc(m * sizeof(int *));
      if (a != NULL) {/* kiểm tra sự cấp phát thành công */
            kt = 0;
            for (int i = 0; i < m; i++) {
                if (kt == 1) break;
                a[i] = (int *) malloc(n*sizeof(int));
                if (a[i] == NULL) kt = 1;
             }
       }
}
```

Lời giải (sử dụng hàm malloc)

```
if (kt == 0) {
    /* sử dụng được a[i][j] */
    // Giải phóng vùng nhớ được cấp phát bằng
malloc
     for (int i = 0; i<m; i++)
          if (a[i] != NULL)
               free(a[i]);
     free(a);
}
```

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 []`.

Bài tập

- Bài 1: 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?

⇒ Việc sử dụng các hàm thao tác khối nhớ như memset, memcpy, memmove giúp khởi tạo hay sao chép/di chuyển vùng nhớ nhanh hơn.

⇒ Trong một số trường hợp chỉ có thể sử dụng vòng lặp kết hợp với lệnh gán để khởi tạo nếu như các byte nhớ cần khởi tạo khác giá trị.

Bài tập

- Bài 2: Cho biết sự khác nhau giữa memcpy và memmove

⇒ Hàm memmove cho phép sao chép hai vùng nhớ chồng lên nhau trong khi hàm memcpy làm việc không chính xác trong trường hợp này

- Bài 3: Trình bày 2 cách khởi tạo mảng `float data[1000];` với giá trị 0.

⇒ C1: `for (int i=0; i<1000; i++) data[i] = 0;`

⇒ C2: `memset(data, 0, 1000*sizeof(float));`