LECTURE 12

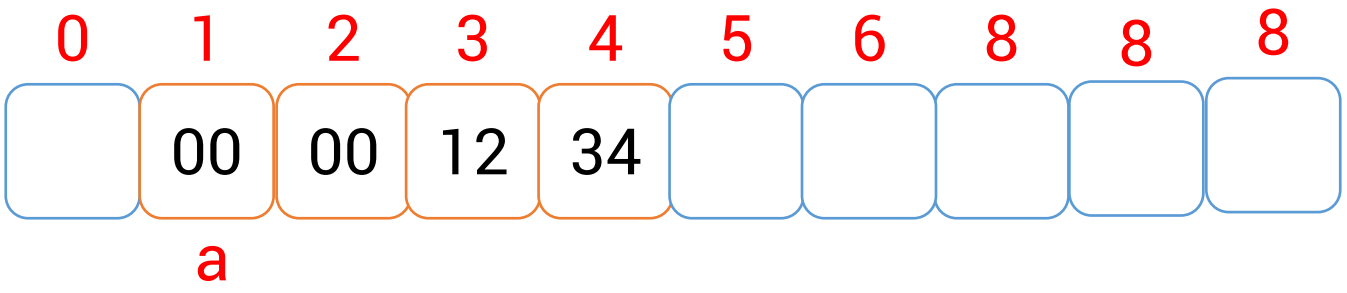
POINTER

Big-O Coding

Website: www.bigocoding.com

Thao tác với biến

- Khi developer khai báo một biến trong source code.
- Một vùng nhớ (memory block) trong bộ nhớ sẽ được cấp phát để lưu giá trị của biến đó.
- Hình ảnh của bộ nhớ, khi khai báo: int a = 1234;

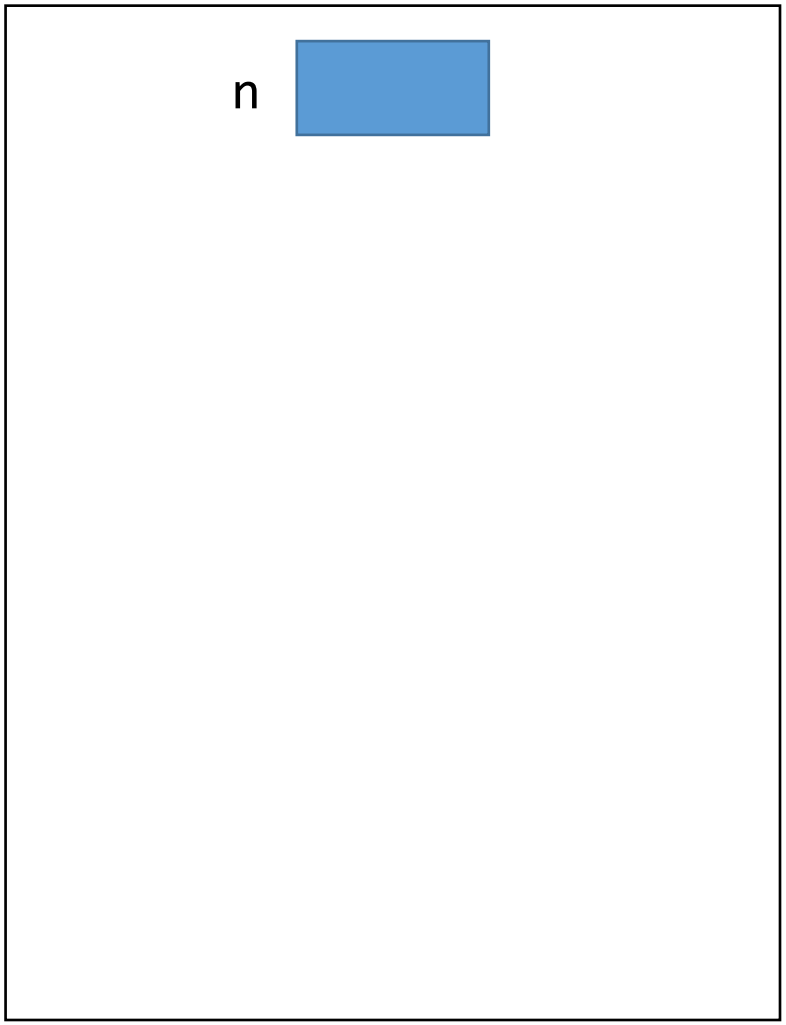


Random Access Memory

- Bộ nhớ RAM trên máy tính được xem là một mảng các ô nhớ (memory cells), mỗi ô nhớ có kích thước 1 byte.
- 16 gigabytes = $2^4 * 2^{10} * 2^{10} * 2^{10}$ bytes.
- Bộ nhớ RAM dùng để lưu trữ hệ điều hành, các lệnh xử lí và dữ liệu cần xử lí.
- Mỗi ô nhớ trong RAM có 1 địa chỉ duy nhất, đánh số từ 0.
- VD: địa chỉ các ô nhớ trong RAM 512 MB đánh số từ 0 đến $2^9 * 2^{10} * 2^{10} = 2^{29} - 1$.

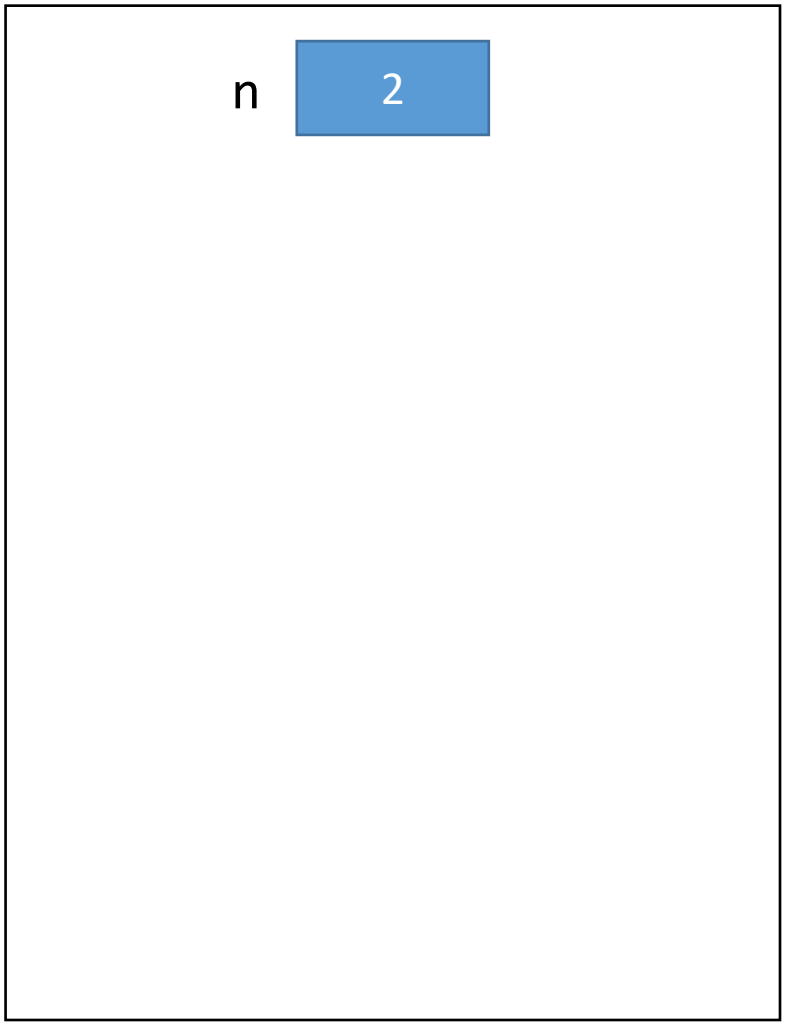
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



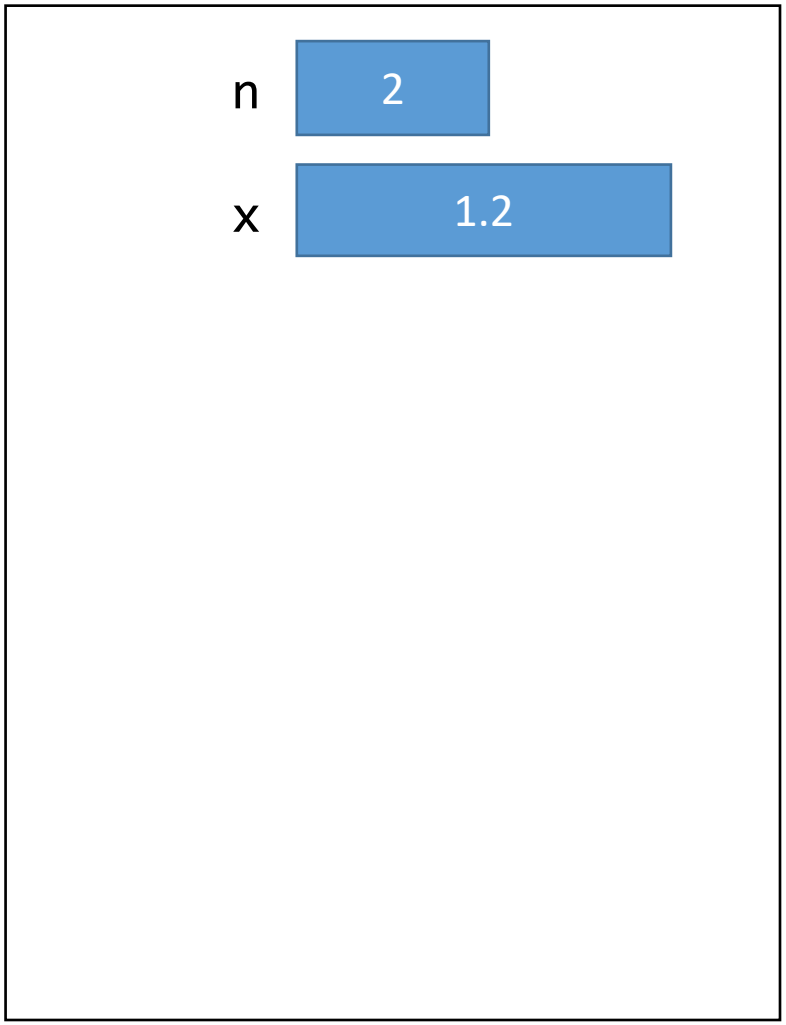
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



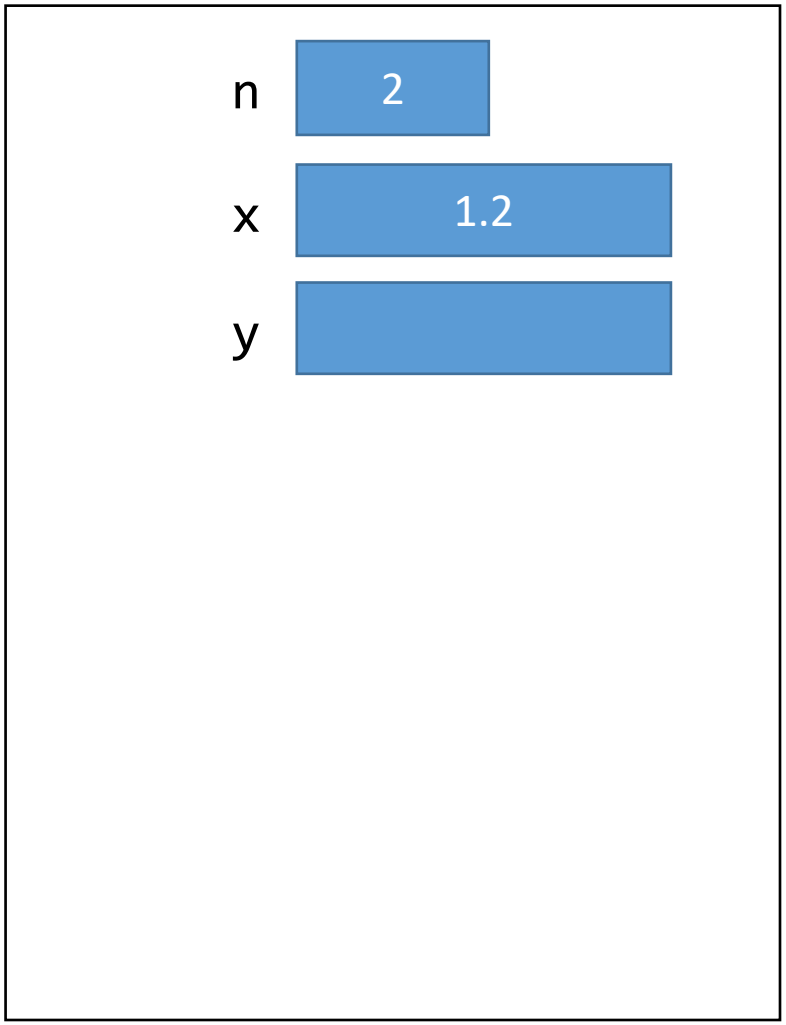
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



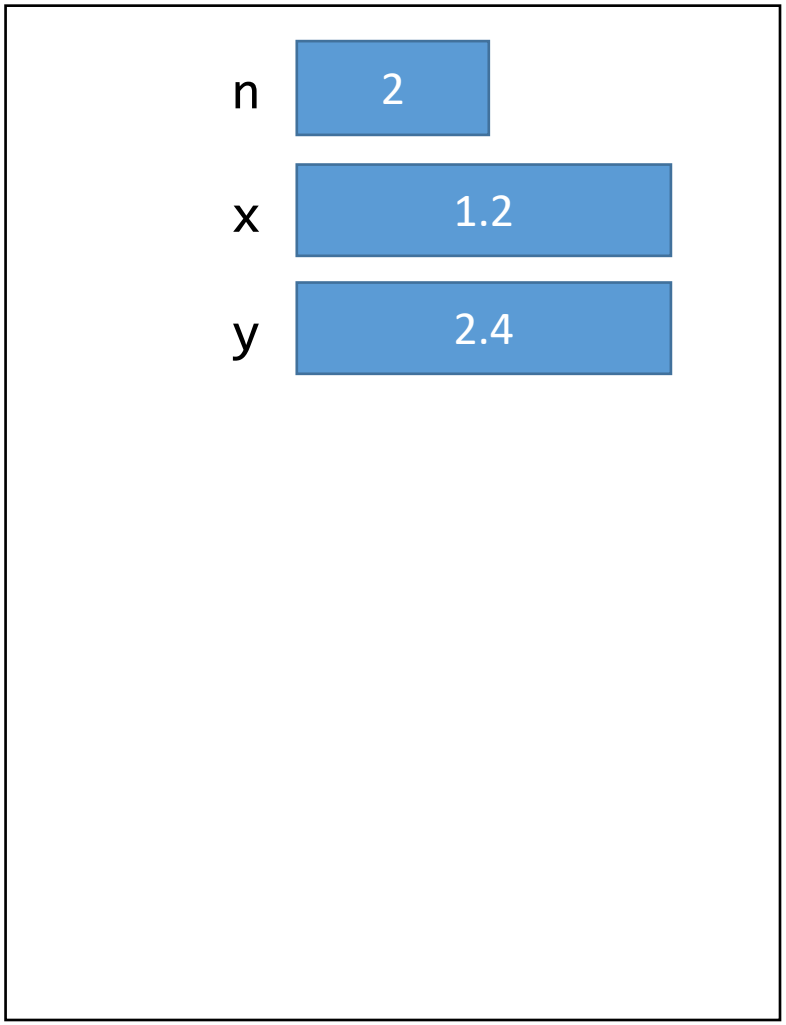
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



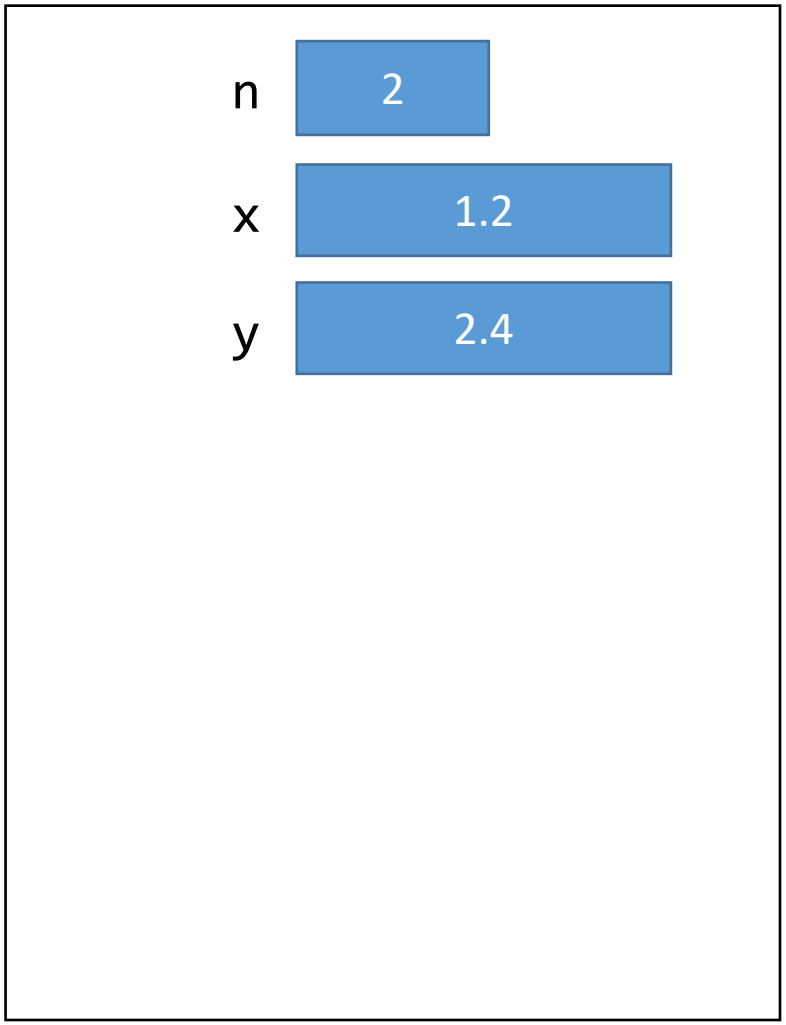
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



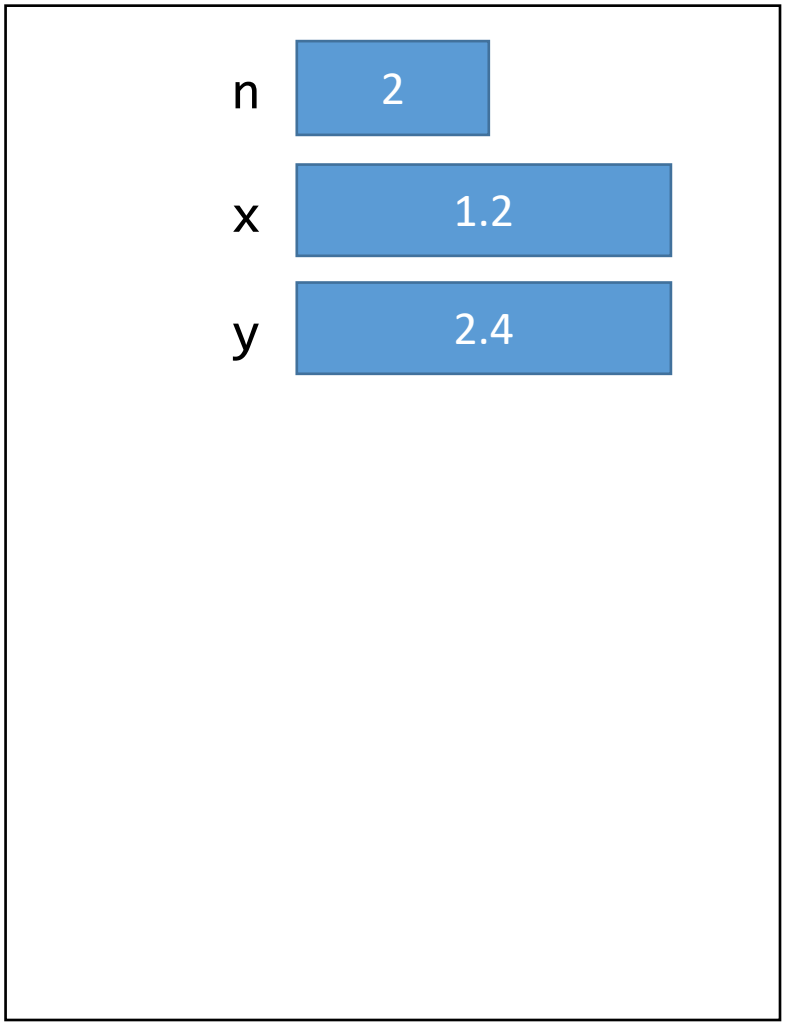
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



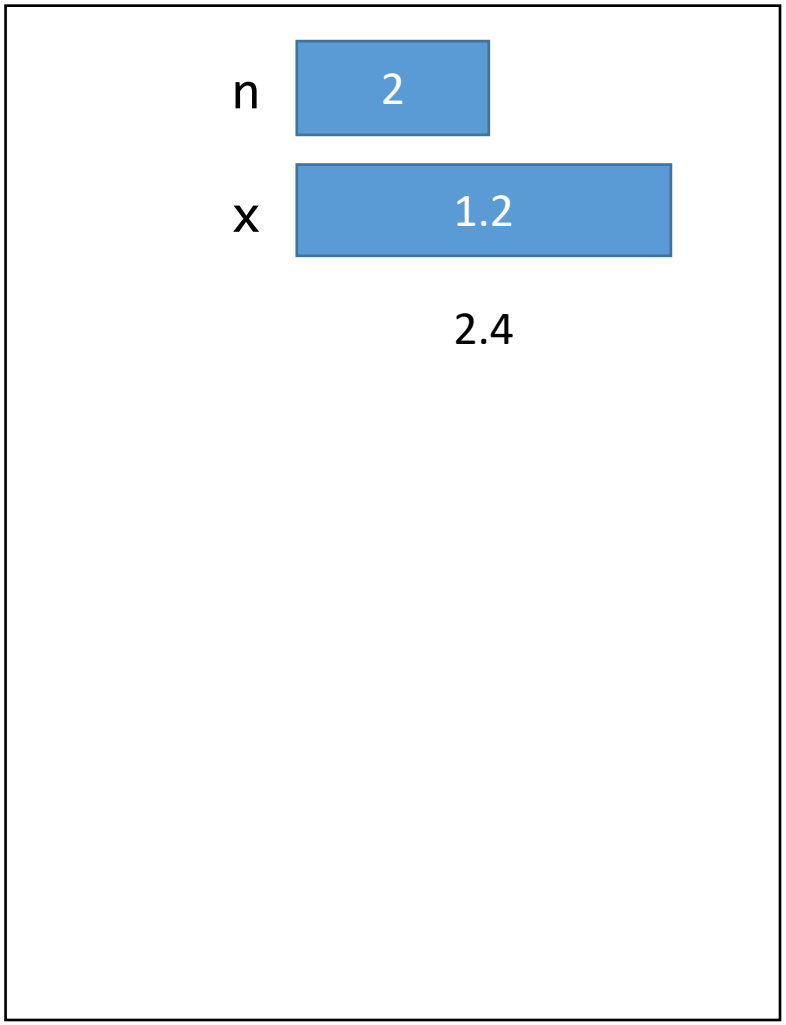
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



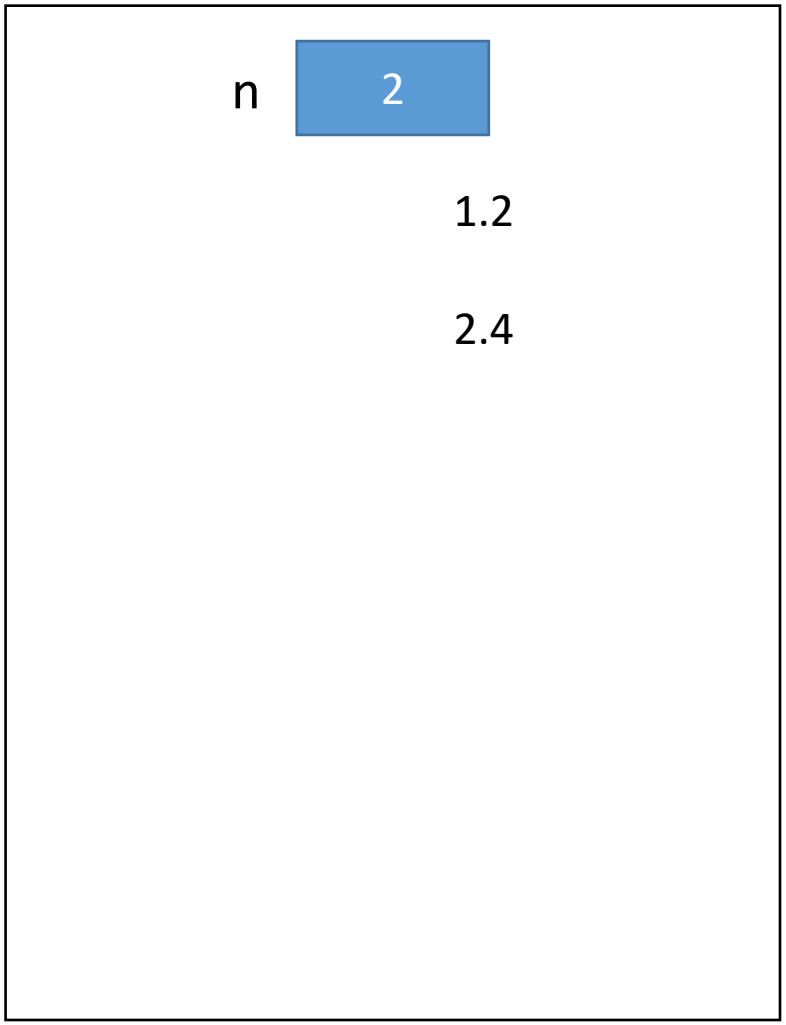
Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```



Ví dụ

```
int main(){
    int n;
    n = 2;
    float x = 1.2;
    float y;
    y = x * n;
    cout << y << endl;
    return 0;
}
```

```
2

1.2

2.4
```

Khai báo biến với nhiều mục đích khác nhau

- Cần lưu trữ tuổi của một người.
  - `int age;`
- Cần lưu GPA của một sinh viên.
  - `double gpa;`
- Cần lưu giá trị true/false.
  - `bool isPrimeA;`

Khai báo biến với nhiều mục đích khác nhau

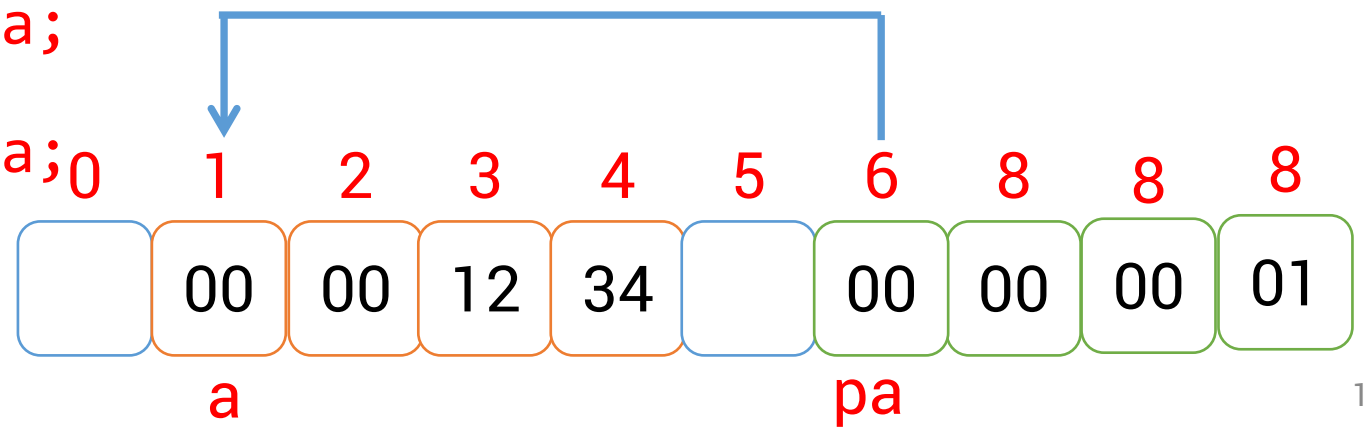
- Cần lưu chiều cao (cm) tất cả các bạn trong lớp.
  - `int heights[100];`
- Cần lưu giá (USD) các mặt hàng trong giỏ hàng.
  - `double prices[100];`
- Cần lưu tên học viên.
  - `string fullName;`

Khai báo biến với nhiều mục đích khác nhau

- Cần lưu 1 phân số.
  - `Fraction p1;`
- Cần lưu thông tin 1 sản phẩm.
  - `Product p1;`
- Cần lưu danh sách các sản phẩm trong giỏ hàng.
  - `Product a[100];`
- Cần lưu 1 mảng 2 chiều các số nguyên.
  - `int a[100][100];`

Pointer là gì?

- Nhắc lại, mỗi biến khai báo ra đều chiếm 1 vùng nhớ nào đó. Mỗi vùng nhớ đều có địa chỉ xác định.
- Do đó, cần lưu địa chỉ của 1 biến kiểu int, ta sử dụng con trỏ.
  - `int a = 1234;`
  - `int *pa;`
  - `pa = &a;`



Khai báo con trỏ

- Cú pháp:
  - `<kiểu dữ liệu> *<tên biến>;`
- Ví dụ:
  - Khai báo con trỏ, trỏ đến 1 vùng nhớ lưu 1 kí tự.
    - char *p;
  - Khai báo con trỏ, trỏ đến 1 vùng nhớ lưu 1 số nguyên.
    - int *p;

Khai báo con trỏ

- Ví dụ:
  - Khai báo con trỏ, trỏ đến 1 vùng nhớ lưu số thực
    - float *p;
  - Khai báo con trỏ, trỏ đến 1 vùng nhớ lưu thông tin sinh viên
    - Student *p;

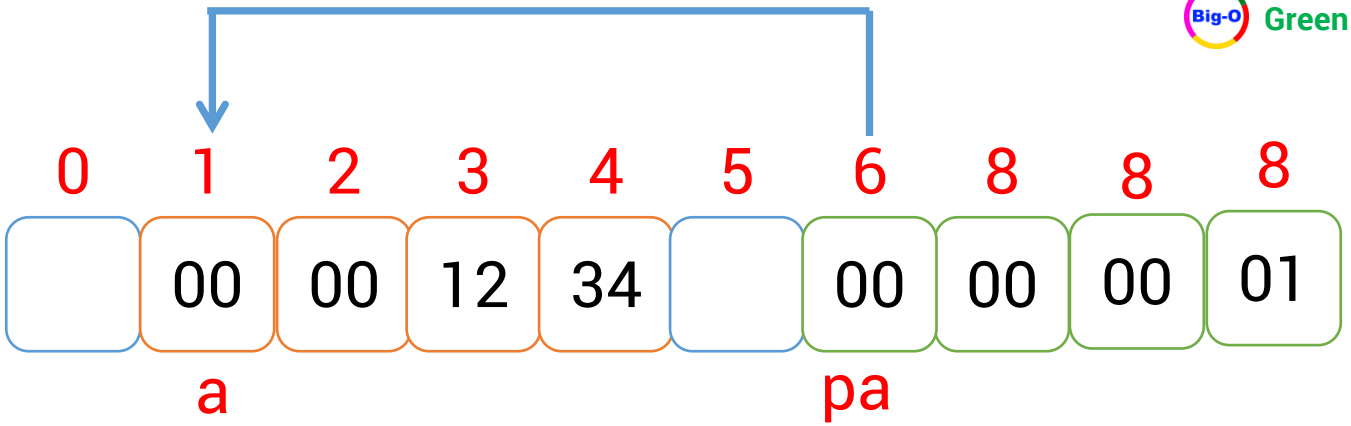
NULL pointer

- NULL pointer là con trỏ không trỏ đến đâu cả (point to nowhere).
- NULL pointer khác với unrefenced pointer.
- Ví dụ:
  - `int n = 3;`
  - `int *p1 = &n; // p1 trỏ đến biến n`
  - `int *p2; // unreferenced pointer`
  - `int *p3 = NULL; // NULL pointer`

Operator & và operator *

- Operator &: lấy địa chỉ của một biến
  - Operator này có thể là biến thông thường hay biến con trỏ đều được.
- Operator *: lấy giá trị tại vùng nhớ mà con trỏ đang trỏ đến.
  - Operator này chỉ áp dụng cho biến con trỏ.

Ví dụ



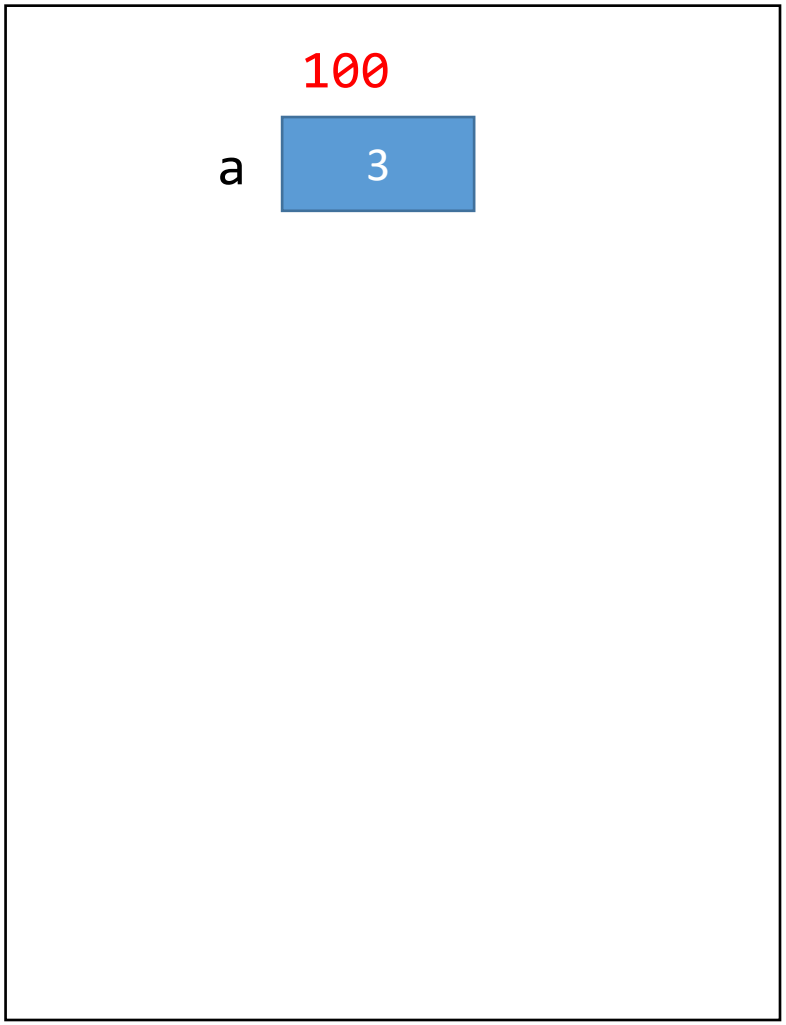
- `int a = 1234;`
- `int *pa;`
- `pa = &a; // lấy địa chỉ vùng nhớ của biến a và gán vào pa`
- `cout << a; // giá trị biến x`
- `cout << &a; // địa chỉ biến a`
- `cout << *a; //` ~~vùng nhớ mà a trỏ đến~~
- `cout << pa; // giá trị biến pa`
- `cout << &pa; // địa chỉ biến pa`
- `cout << *pa; // vùng nhớ mà pa đang trỏ đến`

Operator new

- Operator new là toán tử, dùng cho con trỏ xin phép hệ điều hành cấp phát 1 vùng nhớ nào đó cho con trỏ quản lí.
- Hệ điều hành có thể đồng ý cấp phát hoặc không đồng ý.
  - Nếu đồng ý, hệ điều hành trả về địa chỉ của vùng nhớ mới được cấp phát.
  - Nếu không đồng ý, hệ điều hành trả về NULL.

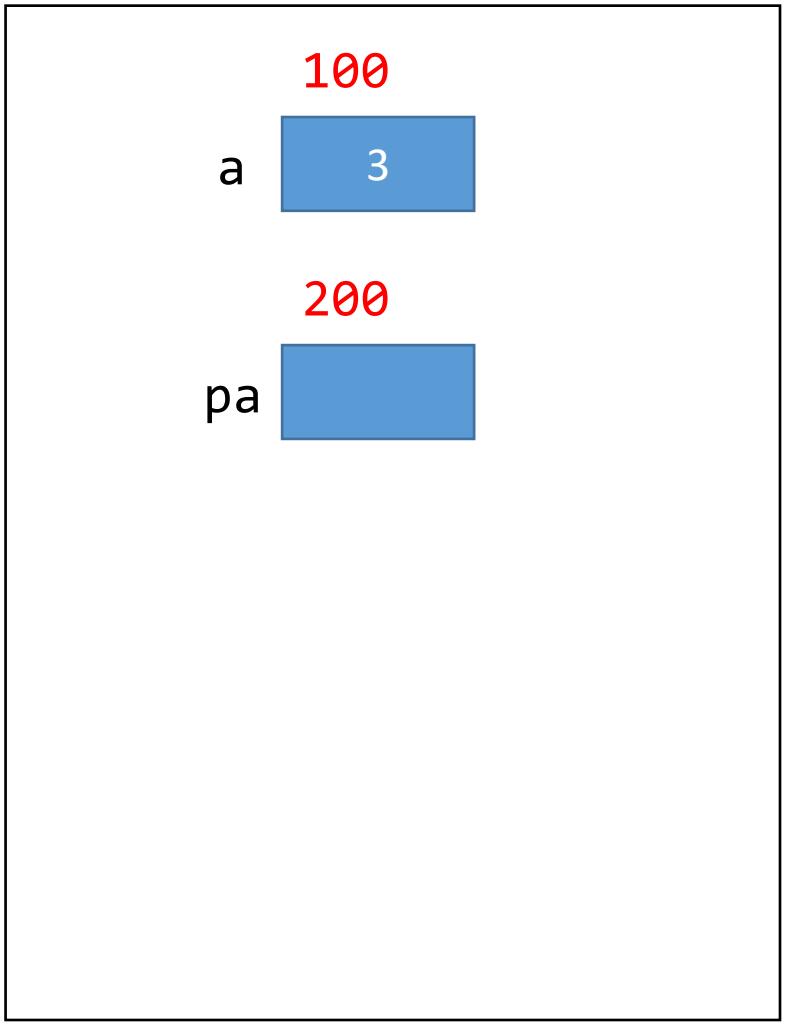
Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
        // NULL pointer
    }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
}
```



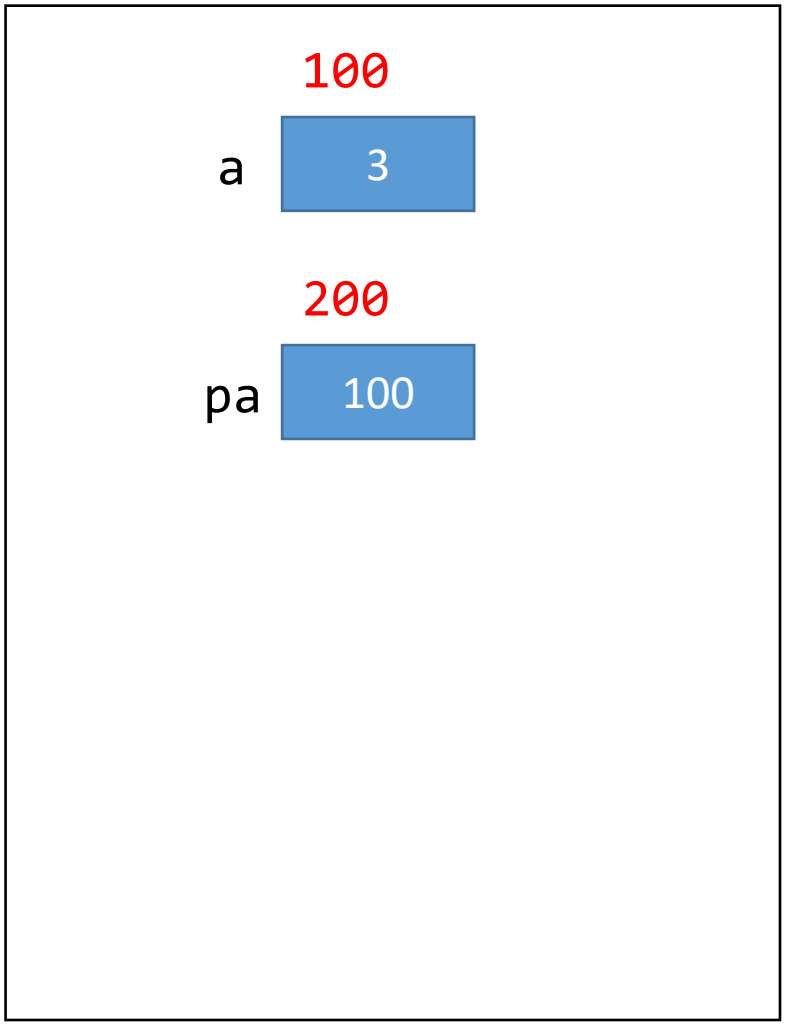
Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
        // NULL pointer
    }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
}
```



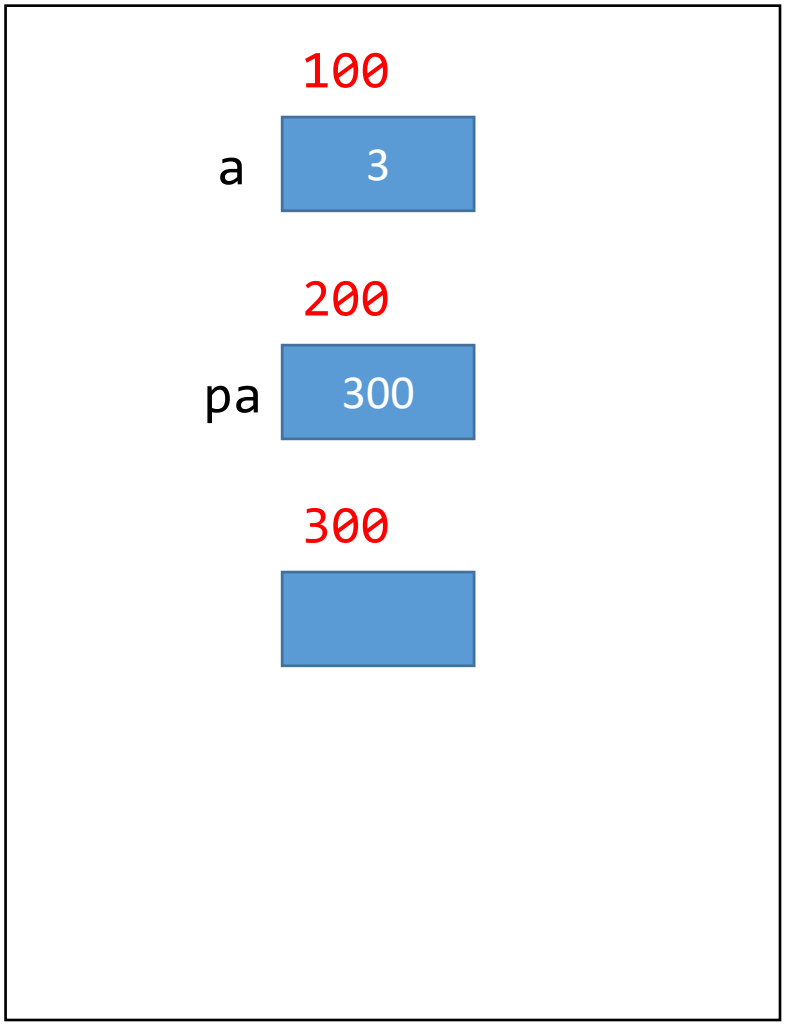
Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
        // NULL pointer
    }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
}
```



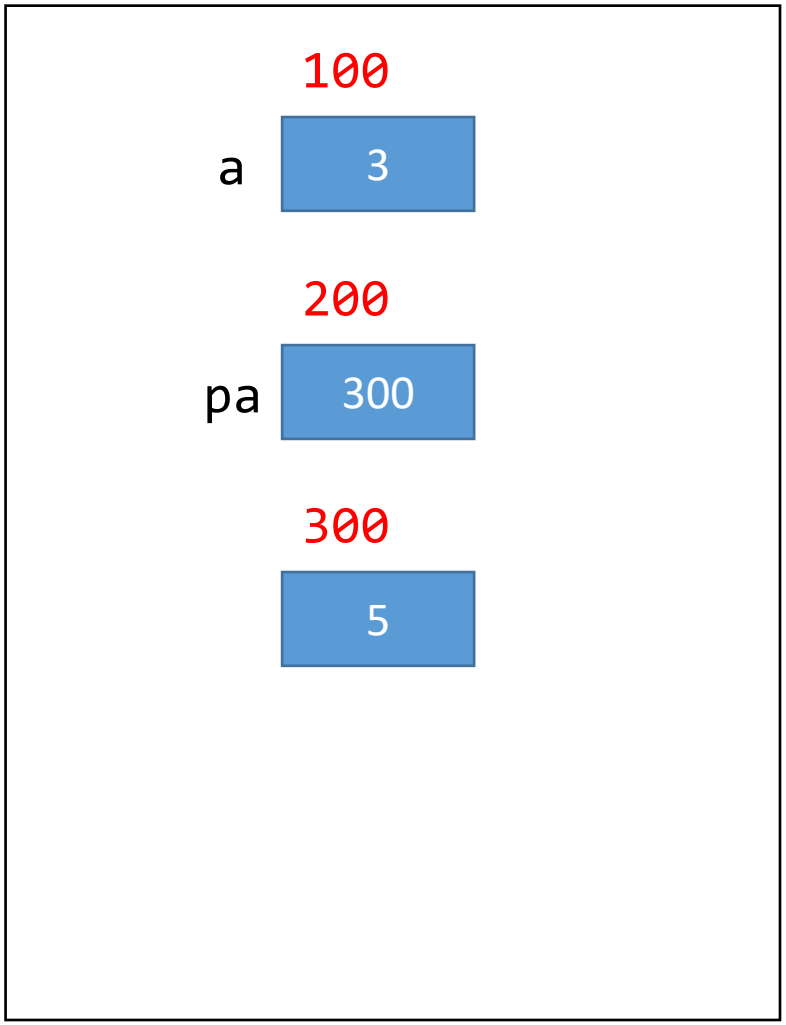
Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
        // NULL pointer
    }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
}
```



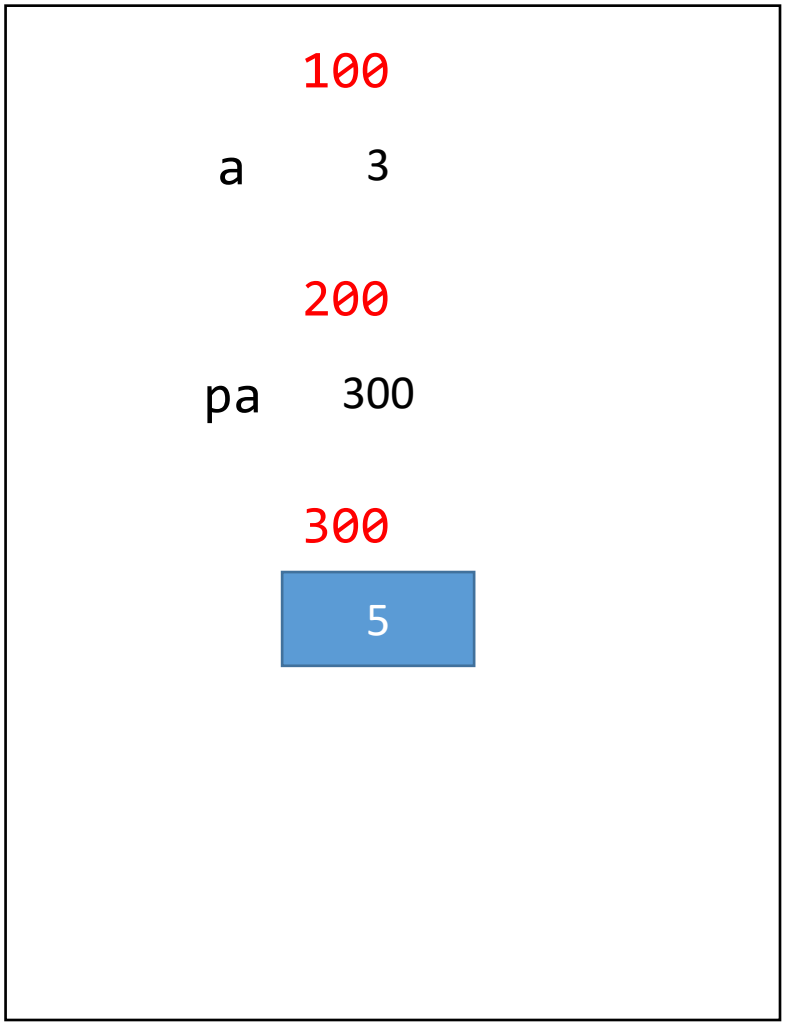
Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
        // NULL pointer
  }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
}
```



Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
       // NULL pointer
  }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
}
```

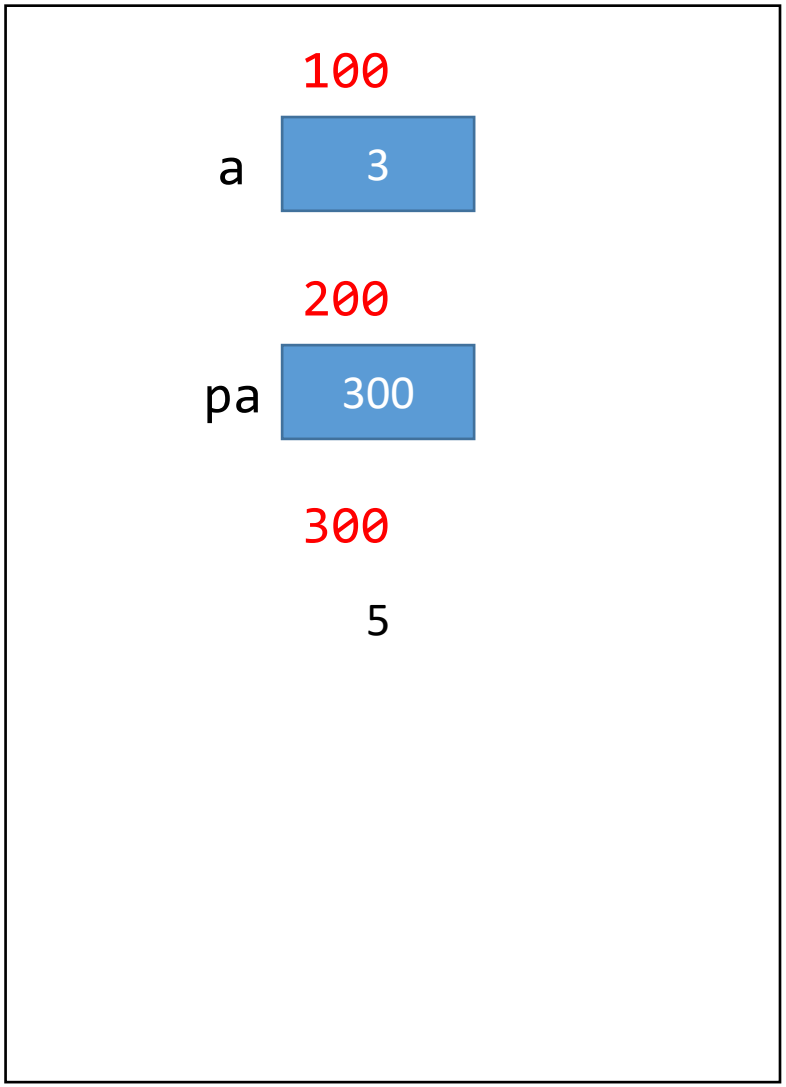


Operator delete

- Với các vùng nhớ được cấp phát cho con trỏ, hệ điều hành sẽ không thực hiện việc thu hồi.
- Như vậy, nếu trong chương trình, ta new quá nhiều, thì đến khi kết thúc chương trình, sẽ có nhiều vùng nhớ không được thu hồi như vùng nhớ 300 ở trên.
- Do đó, operator delete giúp developer xin hủy/xin trả lại hệ điều hành vùng nhớ đã xin cấp phát new (trước đó).

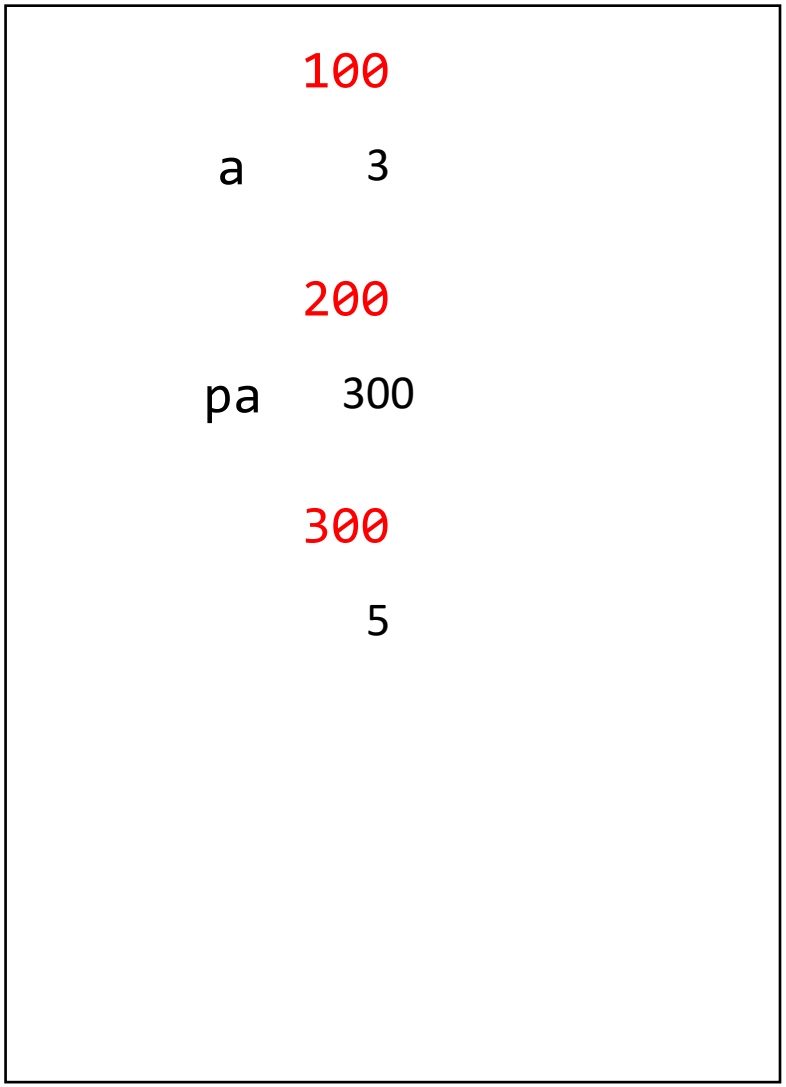
Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
       // NULL pointer
    }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
    delete pa;
}
```



Ví dụ

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    pa = new int;
    if(pa !=NULL){
        *pa = 5;
    }
    else{
       // NULL pointer
    }
  // Lưu ý sự khác biệt
giữa operator & và
operator new
    delete pa;
}
```



Lưu ý

- Vùng nhớ nào mình new, thì phải được delete.

```
int main(){
    int *pa;
    pa = new int;
    // Do sth
    delete pa; // MUST HAVE!!!
}
```

Lưu ý

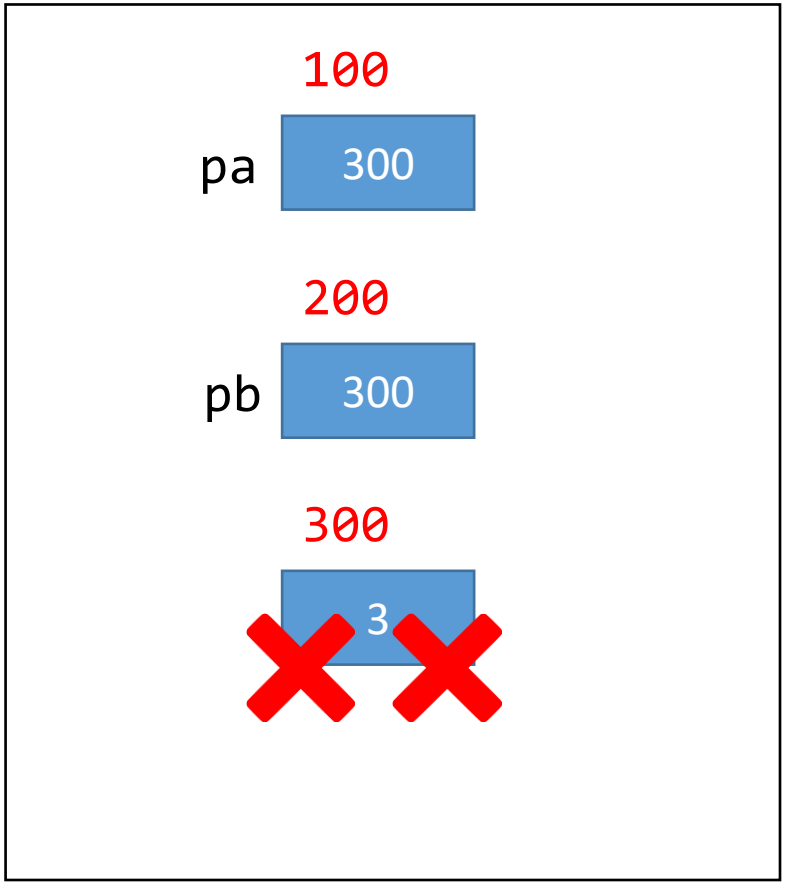
- Vùng nhớ nào mình không new, thì đừng tự tiện delete.

```
int main(){
    int a = 3;
    int *pa;
    pa = &a;
    delete pa; // ERROR!!!
}
```

Lưu ý

- Đừng delete vùng nhớ 2 lần.

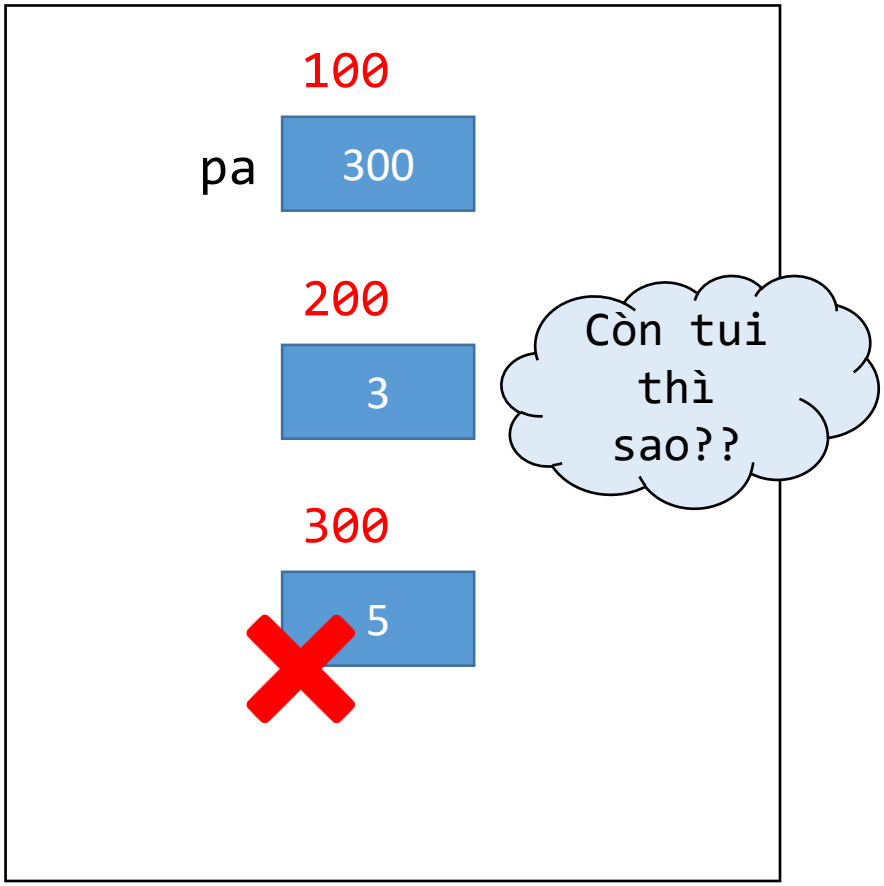
```
int main(){
    int *pa;
    int *pb;
    pa = new int(3);
    pb = pa;
    delete pb; // OK
    delete pa; // ERROR!!!
}
```



Lưu ý

- Đừng delete vùng nhớ quá trễ.

```
int main(){
    int *pa;
    pa = new int(3);

    pa = new int(5);
    delete pa;
}
```



Lưu ý

- Đừng chơi với unreferenced variable, unreferenced pointer.

```
int main(){
    int a;
    cout << a << endl; // ERROR!!!

    int *pa;
    cout << *pa << endl; // ERROR!!!
}
```

Lưu ý

- Đừng lấy giá trị của NULL pointer

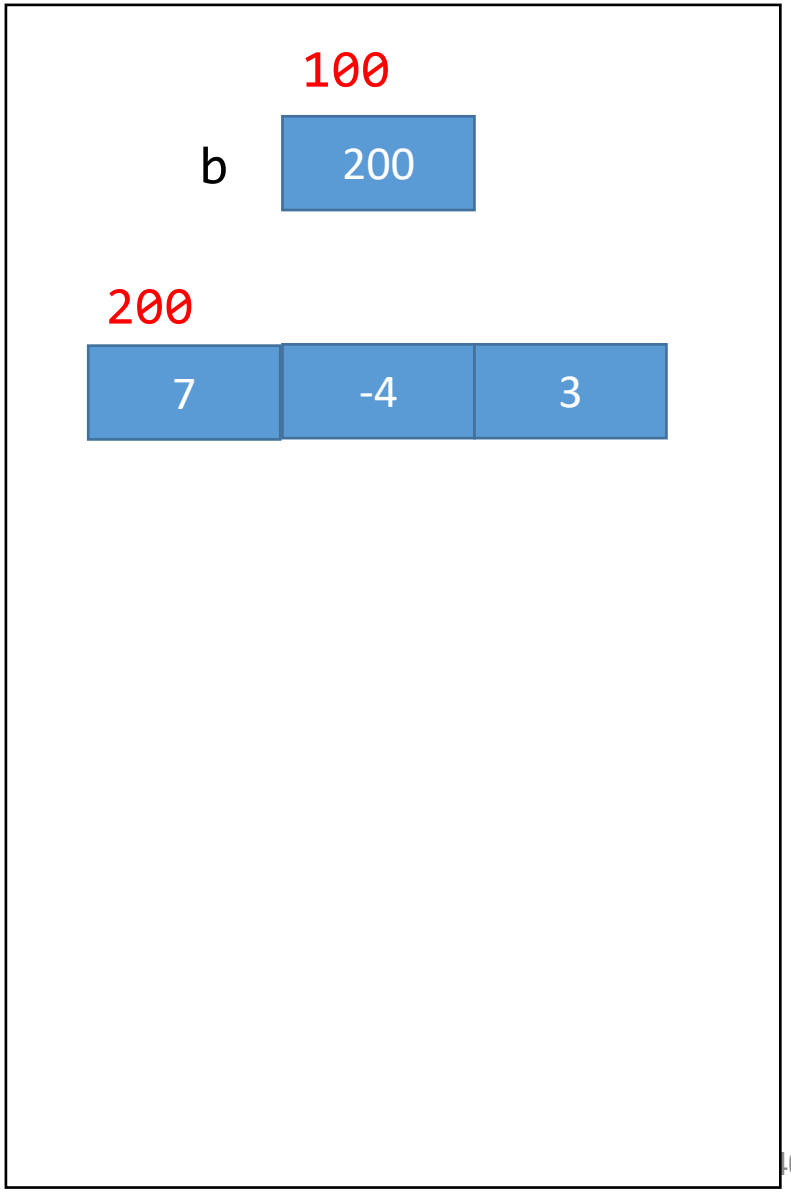
```
int main(){
    int *pa;
    pa = NULL;
    cout << *pa << endl; // ERROR!!!
}
```

Sử dụng pointer để quản lí 1 biến đơn

```
int main(){
    int *pa;
    pa = new int;
    *pa = 3;
    *pa = *pa + 5;
    cout << *pa << endl;
}
```

Sử dụng pointer để quản lí mảng 1 chiều

```
int main(){
    int a[1000]; // hoang phí

    int *b;
    int n;
    cin >> n;
    b = new int[n];
    for(int i = 0; i < n; i++)
        cin >> b[i];

    cout << b[0] << endl;
    cout << *(b) << endl;

    cout << b[1] << endl;
    cout << *(b+1) << endl;

    cout << b+1 << endl;

    delete []b;
}
```



Sử dụng pointer để quản lí mảng 2 chiều

```
int main(){
    int a[1000][1000]; // hoang phí

    int **b;
    int m, int n;
    cin >> m >> n;
    b = new int*[m];
    for(int i = 0; i < n; i++)
        b[i] = new int [n];

    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
            cin >> a[i][j];

    for(int i = 0; i < n; i++)
        delete [] b[i];
    delete []b;
}
```

Sử dụng pointer để quản lí linked list

- Sẽ học ở buổi tiếp theo.

```
struct SinglyLinkedListNode{
    int data;
    SinglyLinkedListNode *next;
};
```

Sử dụng pointer để quản lí tree

- Sẽ học ở tuần sau.

```
struct TreeNode{
    int data;
    TreeNode *left;
    TreeNode *right;
};
```

Kích thước của pointer

- Vì pointer lưu địa chỉ của vùng nhớ, nên kích thước pointer không phụ thuộc vào kiểu dữ liệu của vùng nhớ mà pointer trỏ đến.
  - sizeof(char *) == sizeof(int *) == sizeof(float *) == sizeof(Student *)
- Kích thước pointer phụ thuộc vào hệ điều hành:
  - 16-bit system: 2 bytes.
  - 32-bit system: 4 bytes.
  - 64-bit system: 8 bytes.

Hỏi đáp