Con trỏ

Kỹ thuật Lập trình (CO1027)

Ngày 4 tháng 4 năm 2021

ThS. Trần Ngọc Bảo Duy

Khoa Khoa học và Kỹ thuật Máy tính

Trường Đại học Bách Khoa, ĐHQG-HCM

Tổng quan

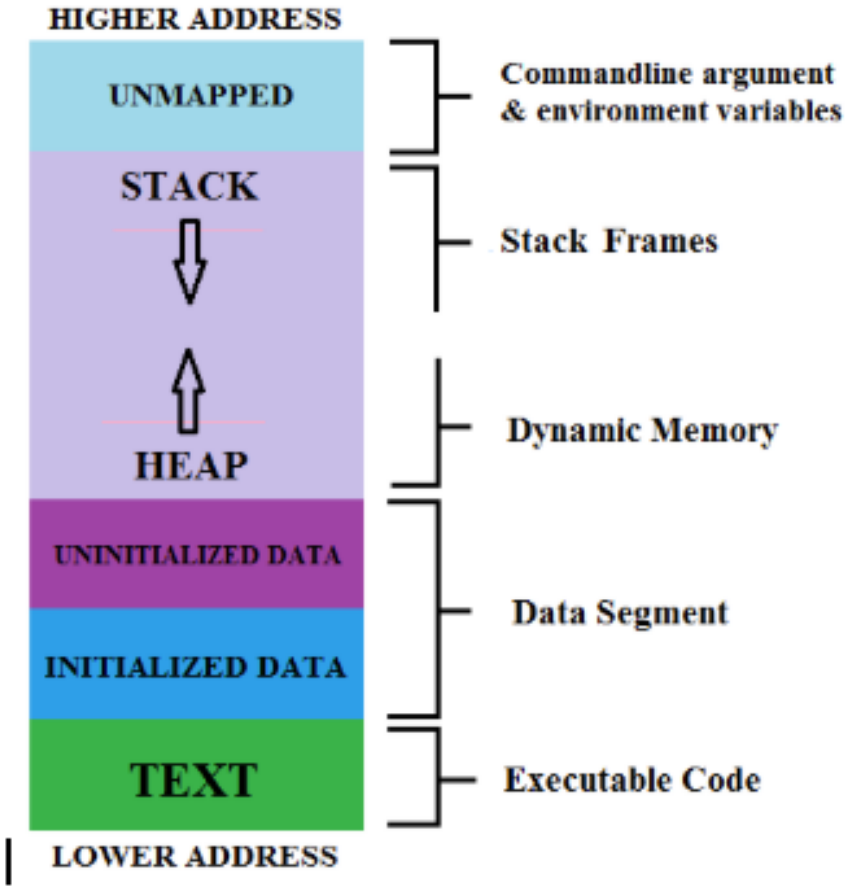
ThS.
Trần Ngọc Bảo Duy

TỔ CHỨC BỘ NHỚ THỰC THI TRONG C/C++

ThS.
Trần Ngọc Bảo Duy

Tổ chức bộ nhớ thực thi

Khi chương trình được lên bộ nhớ để thực thi, hệ thống **tổ chức bộ nhớ** như hình vẽ:



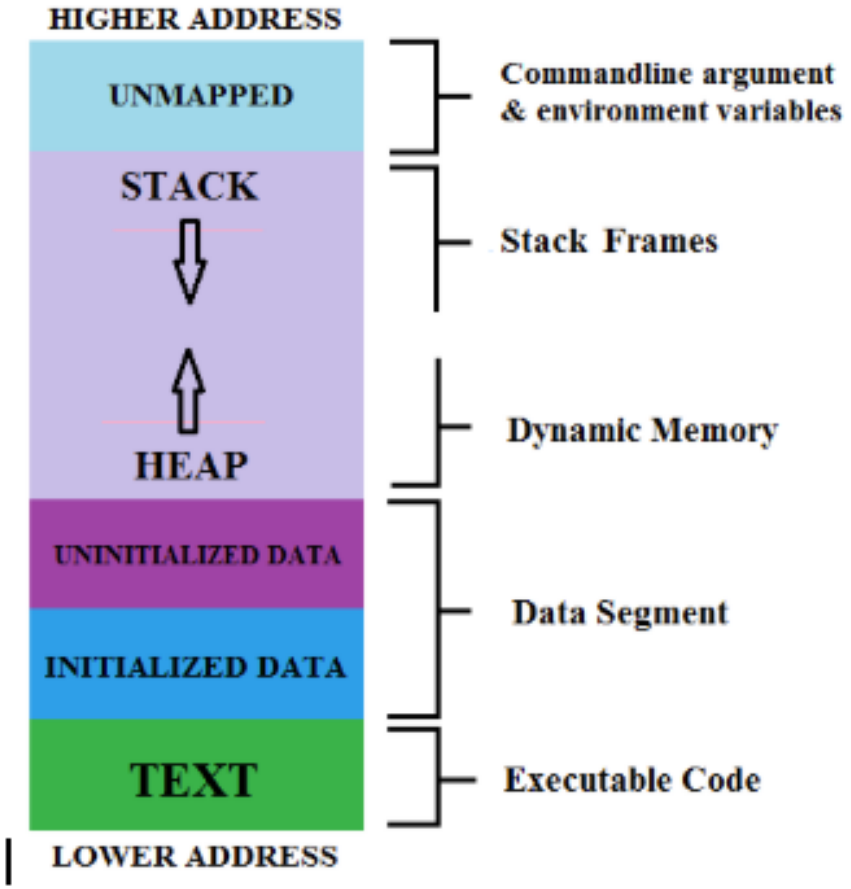
Vùng **TEXT**:

1. Chứa mã thực thi của chương trình.
2. Vùng này chỉ đọc.
3. Vùng này có thể dùng chung trong trường hợp chương trình thực thi thường xuyên.

ThS.
Trần Ngọc Bảo Duy

Tổ chức bộ nhớ thực thi

Khi chương trình được lên bộ nhớ để thực thi, hệ thống **tổ chức bộ nhớ** như hình vẽ:



Vùng **DATA** gồm:

1. Dữ liệu được khởi động.
2. Dữ liệu không được khởi động.

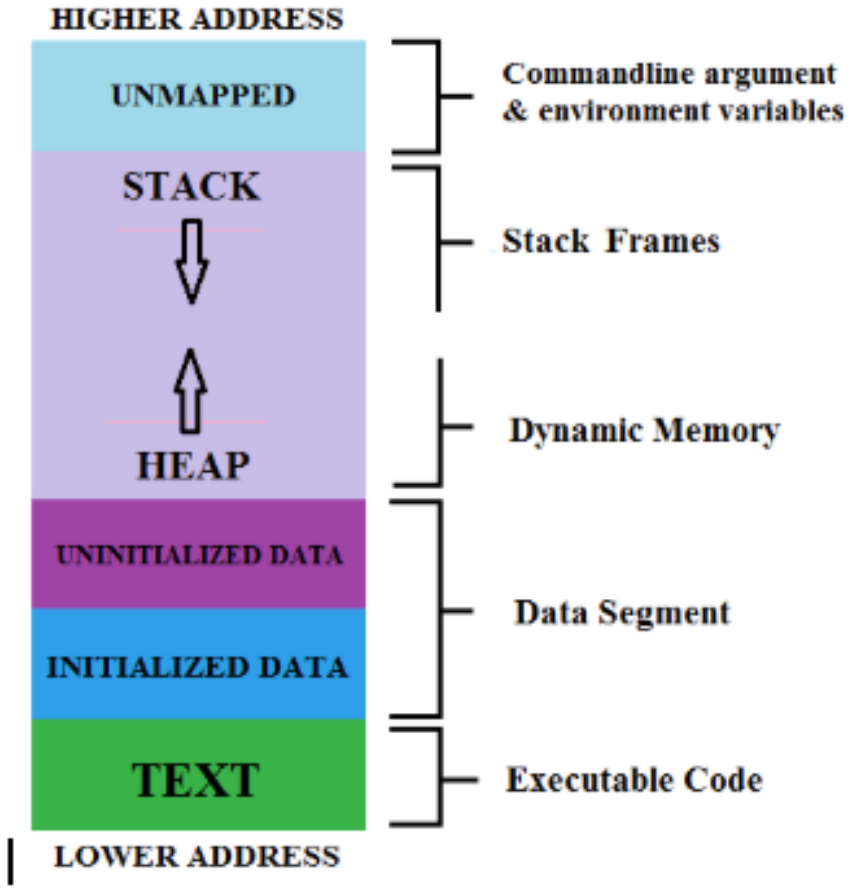
ThS.
Trần Ngọc Bảo Duy

Tổ chức bộ nhớ thực thi

Khi chương trình được lên bộ nhớ để thực thi, hệ thống **tổ chức bộ nhớ** như hình vẽ:

Vùng **dữ liệu được khởi động**: giữ biến toàn cục và biến tĩnh (static), bao gồm 2 vùng con:

1. Chỉ đọc: Hằng chuỗi.
2. Đọc/ ghi: Các biến tĩnh và toàn cục không hằng.

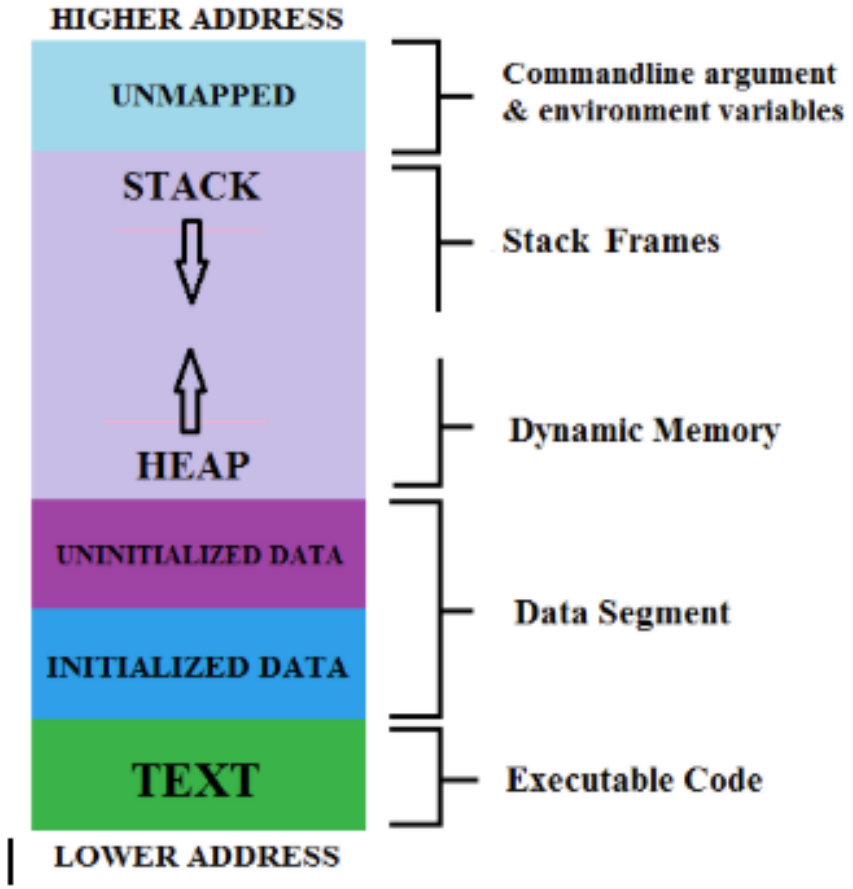


ThS.
Trần Ngọc Bảo Duy

Tổ chức bộ nhớ thực thi

Khi chương trình được lên bộ nhớ để thực thi, hệ thống **tổ chức bộ nhớ** như hình vẽ:

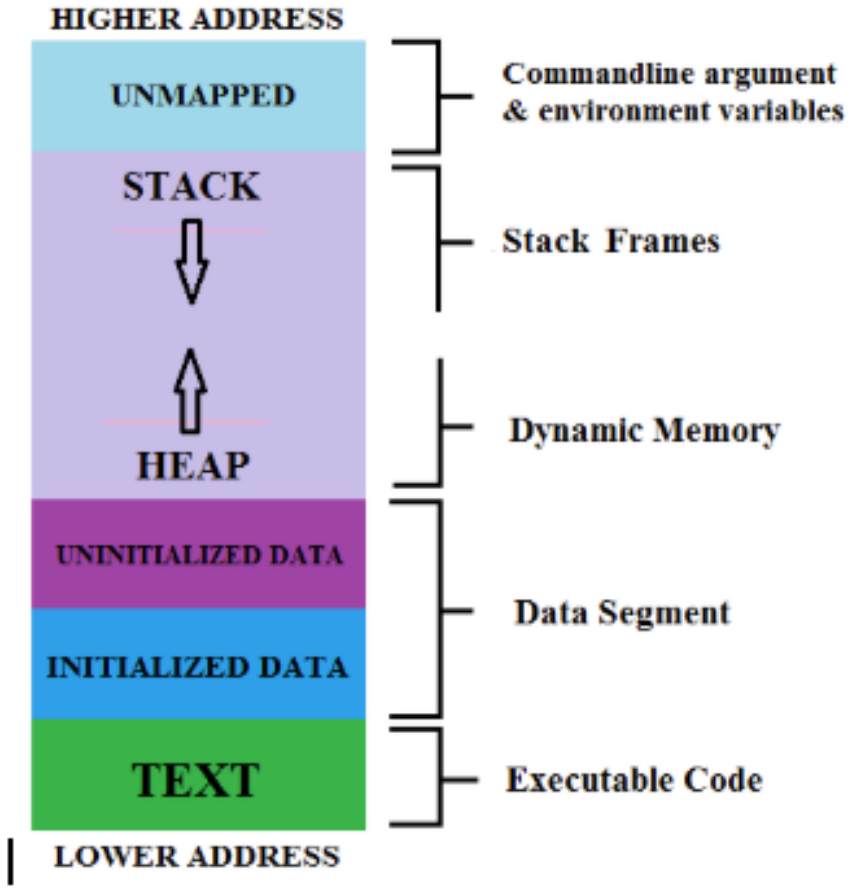
Vùng **dữ liệu không được khởi động**: giữ biến toàn cục và biến tĩnh (static). Hệ thống khởi động về 0 (số) cho các biến không được người lập trình chủ động khởi động.



ThS.
Trần Ngọc Bảo Duy

Tổ chức bộ nhớ thực thi

Khi chương trình được lên bộ nhớ để thực thi, hệ thống **tổ chức bộ nhớ** như hình vẽ:



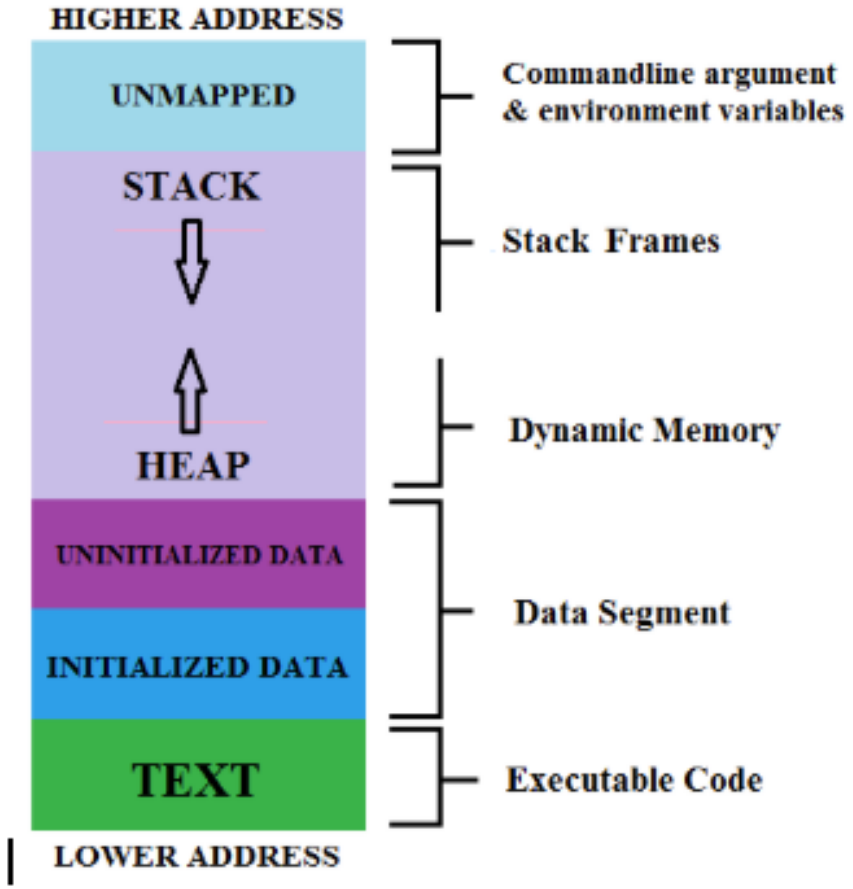
Vùng **HEAP** gồm:

1. Chứa bộ nhớ xin cấp phát động bởi người lập trình.
2. Liên quan đến Kiểu dữ liệu con trỏ nói trong chương này.

ThS.
Trần Ngọc Bảo Duy

Tổ chức bộ nhớ thực thi

Khi chương trình được lên bộ nhớ để thực thi, hệ thống **tổ chức bộ nhớ** như hình vẽ:



Vùng **STACK** gồm:

1. Các biến được khai báo trong chương trình.
2. Thông tin mỗi lần gọi hàm.

ThS.
Trần Ngọc Bảo Duy

CON TRỎ

ThS.
Trần Ngọc Bảo Duy

Bài toán thực tế

Vấn đề

Là một sinh viên trường Đại học Bách Khoa (ĐHQG-HCM), một người khác nhờ bạn hãy giới thiệu về khuôn viên ngôi trường một cách chân thực nhất.

ThS.
Trần Ngọc Bảo Duy

Bài toán thực tế

Vấn đề

Là một sinh viên trường Đại học Bách Khoa (ĐHQG-HCM), một người khác nhờ bạn hãy giới thiệu về khuôn viên ngôi trường một cách chân thực nhất.
Nếu không có các công cụ trực quan, bạn có hai giải pháp:

1. Sao chép một phiên bản khác của trường và đưa cho người đó xem.

ThS.
Trần Ngọc Bảo Duy

Bài toán thực tế

Vấn đề

Là một sinh viên trường Đại học Bách Khoa (ĐHQG-HCM), một người khác nhờ bạn hãy giới thiệu về khuôn viên ngôi trường một cách chân thực nhất.
Nếu không có các công cụ trực quan, bạn có hai giải pháp:

1. Sao chép một phiên bản khác của trường và đưa cho người đó xem.
2. Đưa địa chỉ của trường cho người này, hãy bảo họ tự đến đó mà xem.

Hình: Đ/c: P. Đông Hòa, TP. Dĩ An, tỉnh Bình Dương

ThS.
Trần Ngọc Bảo Duy

Tại sao phải có con trỏ?

1. Như vậy, việc ta tạo thêm một biến mà gán cho nó một giá trị của biến đó giống như ta đang sao chép một phiên bản khác mà chỉ dùng để tham khảo giá trị:

```
1 int a = 5;
2 int b = a;
```


ThS.
Trần Ngọc Bảo Duy

Tại sao phải có con trỏ?

1. Như vậy, việc ta tạo thêm một biến mà gán cho nó một giá trị của biến đó giống như ta đang sao chép một phiên bản khác mà chỉ dùng để tham khảo giá trị:

```
1 int a = 5;
2 int b = a;
```

2. Thay vì mỗi lần muốn tham khảo giá trị bằng một danh hiệu mới, ta phải khai báo và sao chép giá trị thì C/C++ cho phép ta **một kiểu dữ liệu dùng để chứa địa chỉ** của nơi mà ta muốn tham khảo tới.

ThS.
Trần Ngọc Bảo Duy

Tại sao phải có con trỏ?

1. Như vậy, việc ta tạo thêm một biến mà gán cho nó một giá trị của biến đó giống như ta đang sao chép một phiên bản khác mà chỉ dùng để tham khảo giá trị:

```
1 int a = 5;
2 int b = a;
```

2. Thay vì mỗi lần muốn tham khảo giá trị bằng một danh hiệu mới, ta phải khai báo và sao chép giá trị thì C/C++ cho phép ta **một kiểu dữ liệu dùng để chứa địa chỉ** của nơi mà ta muốn tham khảo tới.
3. Ta chỉ cần chứa địa chỉ chứa không cần sao chép lại đối tượng.

ThS.
Trần Ngọc Bảo Duy

Tại sao phải có con trỏ?

1. Như vậy, việc ta tạo thêm một biến mà gán cho nó một giá trị của biến đó giống như ta đang sao chép một phiên bản khác mà chỉ dùng để tham khảo giá trị:

```
1 int a = 5;
2 int b = a;
```

2. Thay vì mỗi lần muốn tham khảo giá trị bằng một danh hiệu mới, ta phải khai báo và sao chép giá trị thì C/C++ cho phép ta **một kiểu dữ liệu dùng để chứa địa chỉ** của nơi mà ta muốn tham khảo tới.
3. Ta chỉ cần chứa địa chỉ chứa không cần sao chép lại đối tượng.

$\Rightarrow$ **KIỂU DỮ LIỆU ĐÓ LÀ CON TRỎ.**

ThS.
Trần Ngọc Bảo Duy

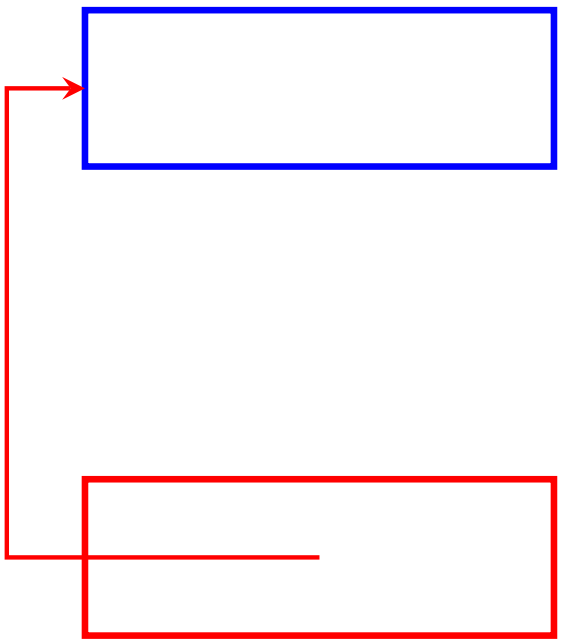
Mô hình con trỏ

Biến x: **Biến bất kỳ**
Ô nhớ bắt đầu của x
có địa chỉ là: **0x1234FFFF**

Biến p: **Con trỏ**
Chứa địa chỉ của biến x,
nghĩa là giá trị **0x1234FFFF**

ThS.
Trần Ngọc Bảo Duy

Mô hình con trỏ



Biến x: **Biến bất kỳ**
Ô nhớ bắt đầu của x
có địa chỉ là: **0x1234FFFF**

Biến p: **Con trỏ**
Chứa địa chỉ của biến x,
nghĩa là giá trị **0x1234FFFF**

Minh hoạ con trỏ bởi tên từ ô nhớ biến p
CHỈ ĐẾN (point to) ô nhớ biến x.

ThS.
Trần Ngọc Bảo Duy

Toán tử &

Toán tử & trả về địa chỉ của một biến.

```
1  #include <iostream>
2  using namespace std;
3  struct Point { double x, y, z; };
4  int main()
5  {
6      int a = 5;
7      cout << a << endl;
8      cout << &a << endl;
9
10     Point p1 = { 1.0, 2.0, 3.0 };
11     cout << p1.x << endl;
12     cout << &p1 << "␣" << &p1.x << endl;
13     cout << &p1.y << "␣" << &p1.z << endl;
14     return 0;
15 }
```

1. Dòng số 9: in ra giá trị của biến nguyên `a`.
2. Dòng số 11: in ra giá trị của thành phần `x` của biến cấu trúc `p1`.

ThS.
Trần Ngọc Bảo Duy

Toán tử &

Toán tử & trả về địa chỉ của một biến.

```
1  #include <iostream>
2  using namespace std;
3  struct Point { double x, y, z; };
4  int main()
5  {
6      int a = 5;
7      cout << a << endl;
8      cout << &a << endl;
9
10     Point p1 = { 1.0, 2.0, 3.0 };
11     cout << p1.x << endl;
12     cout << &p1 << "␣" << &p1.x << endl;
13     cout << &p1.y << "␣" << &p1.z << endl;
14     return 0;
15 }
```

3. Dòng số 8: in ra địa chỉ của biến nguyên a.
4. Dòng số 12, 13: in ra địa chỉ của các biến hoặc thành phần trong biến cấu trúc.

ThS.
Trần Ngọc Bảo Duy

Cú pháp khai báo

Các kiểu khai báo

```
<Tên kiểu>* <Tên biến>;
<Tên kiểu>* <Tên biến a> = NULL;
<Tên kiểu>* <Tên biến a> = &<Tên biến b>;
```


ThS.
Trần Ngọc Bảo Duy

Cú pháp khai báo

Các kiểu khai báo

```
<Tên kiểu>* <Tên biến>;
<Tên kiểu>* <Tên biến a> = NULL;
<Tên kiểu>* <Tên biến a> = &<Tên biến b>;
```

trong đó:

- `<tên biến b>`: Phải có kiểu `<Tên kiểu>`, hoặc có kiểu chuyển đổi qua được `<Tên kiểu>`.
- `NULL`: Hằng số 0, khi đó, con trỏ không giữ bất kỳ địa chỉ nào.

ThS.
Trần Ngọc Bảo Duy

Khai báo con trỏ: Ví dụ

```
1  int a;
2  int *p1;
3  int *p2 = 0;
4  int *p3 = &a;
5
6  double d;
7  double *pd1;
8  double *pd2 = 0;
9  double *pd3 = &d;
10
11 Point p1 = {1.0f, 2.0f, 3.0f};
12 Point *pp1;
13 Point *pp2 = 0;
14 Point *pp3 = &p1;
```


ThS.
Trần Ngọc Bảo Duy

Toán tử *

Toán tử * lấy giá trị (tham khảo) tại một địa chỉ.

ThS.
Trần Ngọc Bảo Duy

Toán tử *

Toán tử * lấy giá trị (tham khảo) tại một địa chỉ.

```
1 #include <iostream>
2 int main() {
3     int a = 100;
4
5     cout << a;
6     cout << *&a; // *(&a)
7     cout << *&*&a; // *(&(*a))
8     cout << *&*&*&a; // *(&(*(&(*a))))
9 }
```


ThS.
Trần Ngọc Bảo Duy

Các phép toán trên con trỏ

- Tăng và giảm: ++ và --.
- Cộng và trừ: + và -.
- Cộng và trừ gán: += và -=.
- So sánh: == và !=.

ThS.
Trần Ngọc Bảo Duy

Các phép toán trên con trỏ

- Tăng và giảm: `++` và `--`.
- Cộng và trừ: `+` và `-`.
- Cộng và trừ gán: `+=` và `-=`.
- So sánh: `==` và `!=`.

Ý nghĩa

Gọi `p` là con trỏ có kiểu `T`: `T *p;`. Các phép cộng và trừ: làm con trỏ `p` tăng hay giảm một **bội số** của kích thước kiểu `T`.

ThS.
Trần Ngọc Bảo Duy

CẤP PHÁT BỘ NHỚ ĐỘNG

ThS.
Trần Ngọc Bảo Duy

Tại sao phải cấp phát động?

Mảng trong C:

- Phải biết trước số lượng phần tử tại thời điểm viết chương trình (*cấp phát tĩnh*).
- Do đó, cần phải khai báo một số lượng lớn các ô nhớ để sẵn. Tuy nhiên, tại một thời điểm nào đó, chương trình có thể sẽ sử dụng ít hơn rất nhiều $\rightarrow$ lãng phí.

ThS.
Trần Ngọc Bảo Duy

Tại sao phải cấp phát động?

Mảng trong C:

- Phải biết trước số lượng phần tử tại thời điểm viết chương trình (*cấp phát tĩnh*).
- Do đó, cần phải khai báo một số lượng lớn các ô nhớ để sẵn. Tuy nhiên, tại một thời điểm nào đó, chương trình có thể sẽ sử dụng ít hơn rất nhiều $\rightarrow$ lãng phí.

Yêu cầu: Có thể nào dùng mảng với số lượng phần tử chỉ cần biết lúc chương trình đang chạy?

$\rightarrow$ Cần cơ chế cấp phát động, buộc phải sử dụng KIỂU DỮ LIỆU **CON TRỎ**.

ThS.
Trần Ngọc Bảo Duy

Con trỏ và mảng

Con trỏ và mảng có nhiều điểm giống nhau:

1. Con trỏ và mảng: giữ địa chỉ của ô nhớ.
   - Con trỏ: giữa địa chỉ của ô nhớ nào đó (của biến khác, của bộ nhớ động).
   - Mảng: giữ địa chỉ của phần tử đầu tiên.

   → Có thể gán mảng vào con trỏ. Tuy nhiên, gán con trỏ vào mảng là không được.

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     int a[5];
5     int *p = a;
6     cout << a << endl;
7     cout << p << endl;
8     return 0;
9 }
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ và mảng

Con trỏ và mảng có nhiều điểm giống nhau:

2. Có cùng cách truy cập các ô nhớ: Dùng toán tử `[]` hoặc toán tử `*`, `+`.

```
1 int a[5];
2 int *p = a;
3 int id = 2;
4
5 a[id] = 100;
6 p[id] = 100;
7
8 *(a + id) = 100;
9 *(p + id) = 100;
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ và mảng

Con trỏ và mảng cũng có điểm khác nhau:

1. Mảng: các phần tử của mảng nằm trên **STACK** của chương trình.
2. Con trỏ: Các phần tử mảng con trỏ chỉ đến có thể trên **STACK** hay **HEAP**.

ThS.
Trần Ngọc Bảo Duy

Cấp phát bộ nhớ động

Định nghĩa

- Giúp người lập trình tạo ra mảng động. Không cần xác định số lượng phần tử của mảng động tại thời điểm biên dịch như mảng tĩnh.

ThS.
Trần Ngọc Bảo Duy

Cấp phát bộ nhớ động

Định nghĩa

- Giúp người lập trình tạo ra mảng động. Không cần xác định số lượng phần tử của mảng động tại thời điểm biên dịch như mảng tĩnh.
- Mảng động sẽ được cấp phát trên **HEAP**.

ThS.
Trần Ngọc Bảo Duy

Cấp phát bộ nhớ động

ThS.
Trần Ngọc Bảo Duy

Định nghĩa

- Giúp người lập trình tạo ra mảng động. Không cần xác định số lượng phần tử của mảng động tại thời điểm biên dịch như mảng tĩnh.
- Mảng động sẽ được cấp phát trên **HEAP**.

GHI CHÚ: Mỗi khi xin cấp phát bộ nhớ CẦN PHẢI giải phóng vùng nhớ xin được khi dùng xong.

Toán tử cấp phát động

1. Toán tử xin bộ nhớ: `new`.
2. Toán tử giải phóng bộ nhớ: `delete`.

```
1  #include <iostream>
2  using namespace std;
3  struct Point { float x, y, z; };
4
5  int main() {
6      int *p1;
7      float *p2;
8      Point *p3;
9
10     int num = 10 * 10;
11
12     p1 = new int[num];
13     p2 = new float[num];
14     p3 = new Point[num];
15
16     delete[] p1;
17     delete[] p2;
18     delete[] p3;
19 }
```


ThS.
Trần Ngọc Bảo Duy

Lưu ý khi cấp phát động

```
1  int num = 100;
2  int *p1 = new int[num];
3
4  if (p1 == NULL) {
5      cout << "Allocation: Failed" << endl;
6      return 1;
7  }
8  else {
9      // Do something with p1
10     delete[] p1;
11 }
12 return 0;
```


ThS.
Trần Ngọc Bảo Duy

Lưu ý khi cấp phát động

```
1  int num = 100;
2  int *p1 = new int[num];
3
4  if (p1 == NULL) {
5      cout << "Allocation: Failed" << endl;
6      return 1;
7  }
8  else {
9      // Do something with p1
10     delete[] p1;
11 }
12 return 0;
```

Toán tử `new` trả về NULL nếu không xin được.

Lúc đó, không thể sử dụng được bộ nhớ. Vì vậy, **LUÔN LUÔN** kiểm tra xem `new` có trả về `NULL` hay không?

ThS.
Trần Ngọc Bảo Duy

Cấp phát một phần tử

C/C++ cho phép việc cấp phát 1 phần tử, khi đó, ta không cần chỉ định rõ số lượng phần tử cần cấp phát.

ThS.
Trần Ngọc Bảo Duy

Cấp phát một phần tử

C/C++ cho phép việc cấp phát 1 phần tử, khi đó, ta không cần chỉ định rõ số lượng phần tử cần cấp phát.

```
1  int *p1 = new int;
2
3  if (p1 == NULL) {
4      cout << "Allocation: Failed" << endl;
5      return 1;
6  }
7  else {
8      // Do something with p1
9      delete[] p1;
10 }
11 return 0;
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ và cấu trúc

```
1  struct Point { float x, y, z; };
2  int main() {
3      Point *ptr = new Point;
4      (*ptr).x = 4.5f;
5      (*ptr).y = 3.5f;
6      (*ptr).z = 5.5f;
7
8      ptr->x = 4.5f;
9      ptr->y = 3.5f;
10     ptr->z = 5.5f;
11     return 0;
12 }
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ và cấu trúc

```
1  struct Point { float x, y, z; };
2  int main() {
3      Point *ptr = new Point;
4      (*ptr).x = 4.5f;
5      (*ptr).y = 3.5f;
6      (*ptr).z = 5.5f;
7
8      ptr->x = 4.5f;
9      ptr->y = 3.5f;
10     ptr->z = 5.5f;
11     return 0;
12 }
```

Tổng quát cách truy cập

<Con trỏ>-><Biến thành viên của cấu trúc>

ThS.
Trần Ngọc Bảo Duy

MỘT SỐ VẤN ĐỀ KHÁC VỀ CON TRỎ

ThS.
Trần Ngọc Bảo Duy

Thứ tự của các toán tử

- `*p++` $\Leftrightarrow$ `*(p++)`
- `*++p` $\Leftrightarrow$ `*(++p)`
- `++*p` $\Leftrightarrow$ `++(*p)`
- `(*p)++` $\Leftrightarrow$ Tăng giá trị tại vùng nhớ mà `p` trỏ tới.

ThS.
Trần Ngọc Bảo Duy

Thứ tự của các toán tử

- `*p++` $\Leftrightarrow$ `*(p++)`
- `*++p` $\Leftrightarrow$ `*(++p)`
- `++*p` $\Leftrightarrow$ `++(*p)`
- `(*p)++` $\Leftrightarrow$ Tăng giá trị tại vùng nhớ mà `p` trỏ tới.

Khi nghi ngờ, hoặc không nhớ... hãy dùng toán tử () để phân giải độ ưu tiên.

ThS.
Trần Ngọc Bảo Duy

Con trỏ và const

```
1 int a = 20;
2 int b = 30;
3 int c = 40;
4
5 const int * ptr1 = &a;
6 int const * ptr1 = &a;
7
8 int* const ptr2 = &b;
```

ptr1: có thể thay đổi được.
Giá trị mà ptr1 chỉ đến không thể thay đổi được.
⇒ ptr1 là **con trỏ hằng** (pointer to constant).
Đặc điểm:

1. Con trỏ hằng có thể trỏ đến các ô nhớ khác nhau.
2. Không thể thay đổi giá trị tại ô nhớ mà nó đang trỏ đến.
3. Từ khóa const nằm phía trước dấu *.

ThS.
Trần Ngọc Bảo Duy

Con trỏ và const

```
1 int a = 20;
2 int b = 30;
3 int c = 40;
4
5 const int * ptr1 = &a;
6 int const * ptr1 = &a;
7
8 int* const ptr2 = &b;
```

ptr2: Không thể thay đổi được giá trị của ptr2 = không thể làm ptr2 chỉ đến ô nhớ nào khác sau dòng này.
Giá trị mà ptr2 chỉ đến có thể thay đổi được qua *ptr2.
⇒ ptr2 là **hằng con trỏ** (constant pointer).
Đặc điểm:

1. Khi khai báo hằng con trỏ cần khởi tạo giá trị địa chỉ cho nó.
2. Địa chỉ khởi tạo đó là cố định, không bao giờ thay đổi.
3. Có thể thay đổi được giá trị tại địa chỉ đã khởi gán ban đầu.
4. Từ khóa const nằm phía sau dấu *.

ThS.
Trần Ngọc Bảo Duy

Con trỏ và const: Một số lỗi thông dụng

```
1 int a = 20;
2
3 const int * ptr1;
4 int * const ptr2 = &a;
5
6 int* const ptr3;
```

```
Error: const variable ptr3 requires an initializer.
```

Lỗi: ptr3 là hằng con trỏ nhưng không được khởi động tương tự như cho ptr3

ThS.
Trần Ngọc Bảo Duy

Con trỏ và const: Một số lỗi thông dụng

```
1 int a = 20;
2
3 const int * ptr1;
4 int * const ptr2 = &a;
5
6 *ptr1 = 100;
```

```
Error: expression must be a modifiable value.
```

Lỗi: Giá trị mà `ptr1` chỉ đến không thay đổi được qua con trỏ `ptr1`. Do đó, nó không thể nằm bên trái biểu thức gán.

ThS.
Trần Ngọc Bảo Duy

Con trỏ và `const`: Một số lỗi thông dụng

```
1 int a = 20;
2 int b = 30;
3
4 const int * ptr1;
5 int * const ptr2 = &a;
6
7 ptr2 = &b;
```

```
Error: expression must be a modifiable value.
```

Lỗi: Con trỏ `ptr2` là hằng con trỏ, nó chỉ nhận giá trị khởi động. Sau đó, không thể làm `ptr2` chỉ đến đối tượng nào khác.

ThS.
Trần Ngọc Bảo Duy

Con trỏ và const: Một số lỗi thông dụng

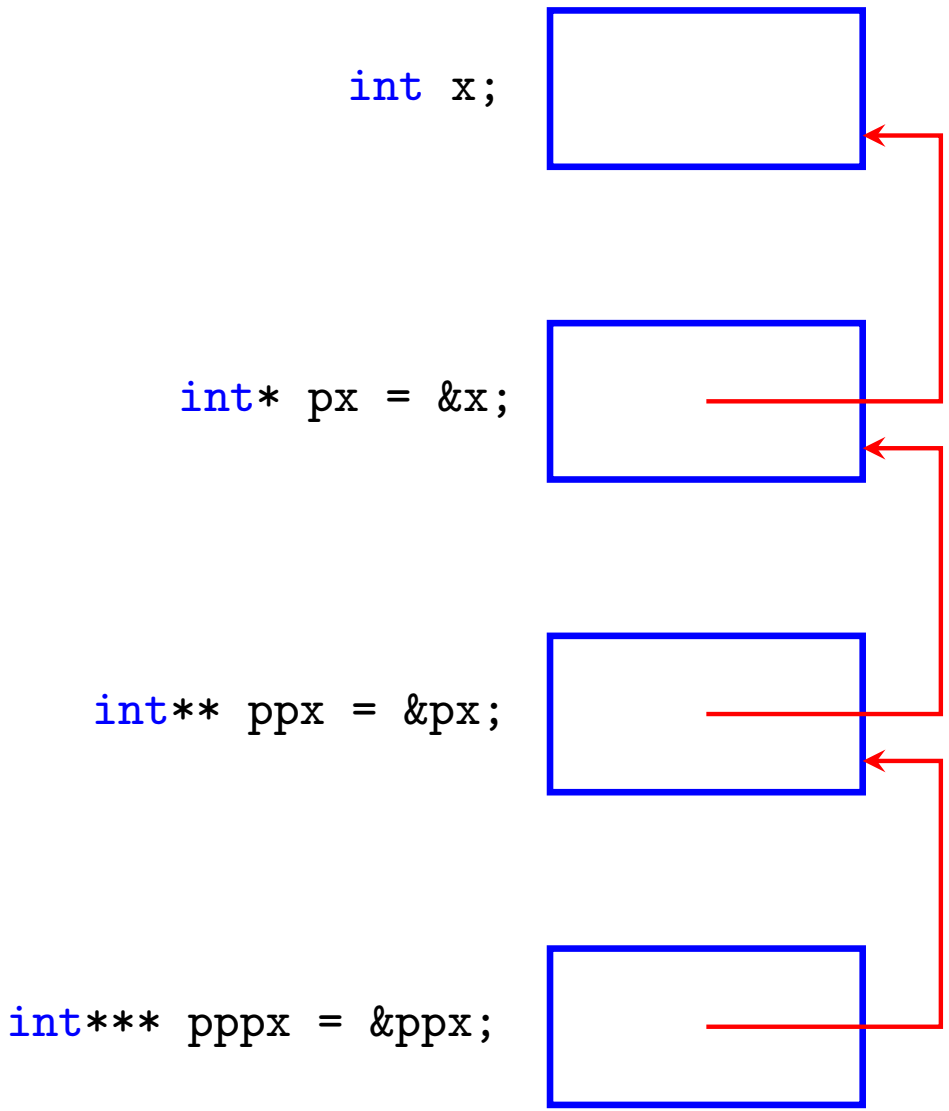
```
1 int a = 20;
2 int b = 30;
3
4 const int * ptr1;
5 int * const ptr2 = &a;
6 int * ptr3 = &b;
7
8 ptr1 = ptr3;
9 ptr3 = ptr1;
```

```
Error: a value of type const int* cannot be assigned to
                    an entity of type int*.
```

ptr3: con trỏ bình thường, thay đổi được nó và giá trị nó chỉ đến.
Gán con trỏ ptr1 vào ptr3: khiến cho giá trị mà ptr1 chỉ đến có thể thay đổi được.
Bộ biên dịch không cho phép, vì nếu cho phép thì ý nghĩa của ptr1 không còn.

ThS.
Trần Ngọc Bảo Duy

Con trỏ trỏ đến con trỏ



ThS.
Trần Ngọc Bảo Duy

Con trỏ void

`void* ptr`: là con trỏ chưa định kiểu, hay là con trỏ tổng quát.

- Có thể được ép kiểu về kiểu mong muốn:

```
1  int a = 10;
2  void *ptr = &a;
3  // cout << *ptr; -> ERROR
4
5  cout << *((int*) ptr); // Print out 10
6
7  char b = 'a';
8  ptr = &b;
9
10 cout << *((char*) ptr); // Print out a
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ `void`

`void* ptr`: là con trỏ chưa định kiểu, hay là con trỏ tổng quát.

- Có thể được ép kiểu về kiểu mong muốn:

```
1  int a = 10;
2  void *ptr = &a;
3  // cout << *ptr; -> ERROR
4
5  cout << *((int*) ptr); // Print out 10
6
7  char b = 'a';
8  ptr = &b;
9
10 cout << *((char*) ptr); // Print out a
```

- Con trỏ `void` giúp chương trình uyển chuyển, Nhưng rủi ro đi kèm: bộ biên dịch không thể kiểm tra tương thích kiểu tại thời điểm biên dịch.

ThS.
Trần Ngọc Bảo Duy