Chương 6

Kỹ thuật sử dụng con trỏ

Giảng viên: Nguyễn Chiến Thắng
Email: thangnc.haui@gmail.com

Nội dung

1. Khái niệm con trỏ
2. Định nghĩa kiểu con trỏ
3. Khai báo biến con trỏ
4. Các toán tử con trỏ
5. Con trỏ và mảng một chiều
6. Cấp phát bộ nhớ động
7. Giải phóng bộ nhớ động
8. Toán tử new và delete
9. Kỹ thuật sử dụng con trỏ

6.1. Khái niệm con trỏ

- Con trỏ là một biến, nó **chứa địa chỉ ô nhớ** của một biến khác.
- Nếu một biến chứa địa chỉ của một biến khác, thì biến này được gọi là con trỏ ***trỏ đến*** biến thứ hai.
- Con trỏ cung cấp phương thức truy xuất gián tiếp đến giá trị của một phần tử dữ liệu.
- Các con trỏ có thể trỏ đến các biến có kiểu dữ liệu cơ bản như **int, char, double, …** hay dữ liệu tập hợp như **mảng, chuỗi** hoặc **cấu trúc.**

Khái niệm con trỏ (tt)

- Các tình huống con trỏ có thể được sử dụng:
  - ✓ Để trả về nhiều hơn một giá trị từ một hàm.
  - ✓ Để truyền mảng và chuỗi từ một hàm đến một hàm khác thuận tiện hơn.
  - ✓ Để làm việc với các phần tử của mảng thay vì truy xuất trực tiếp vào các phần tử này.
  - ✓ Để cấp phát bộ nhớ và truy xuất bộ nhớ (Cấp phát bộ nhớ trực tiếp hay cấp phát động).
  - ✓ Khắc phục tình trạng không gian nhớ hữu hạn khi sử dụng mảng cố định (lãng phí hoặc thiếu bộ nhớ khi dữ liệu thay đổi thương xuyên).

6.2. Định nghĩa kiểu con trỏ

- Cú pháp khai báo tổng quát:

```
<typedef> <Kiểu_dữ_liệu> *<Kiểu_con_trỏ>;
```

- Ví dụ:

```
typedef int *intPointer;
typedef double *doublePointer;
```

6.3. Khai báo biến con trỏ

- Cú pháp 1:

```
<kiểu_dữ_liệu> *<tên_con_trỏ>;
```

- Ví dụ:

```
int *iPt;
```

- Cú pháp 2:

```
<kiểu_con_trỏ> <tên_con_trỏ>;
```

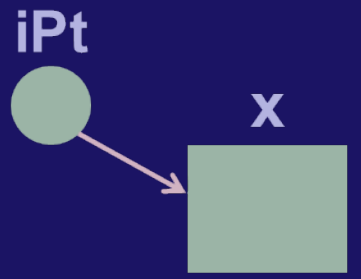
- Ví dụ:

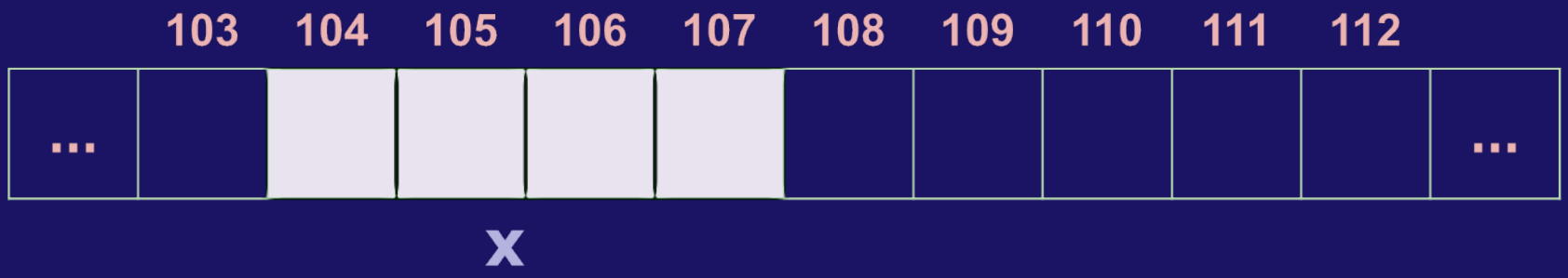
```
intPointer iPt;
```

6.4. Các toán tử con trỏ

- Toán tử **&** là toán tử một ngôi và nó trả về địa chỉ ô nhớ của một biến.
- Ví dụ:

```
iPt = &x; //iPt trỏ vào x
```





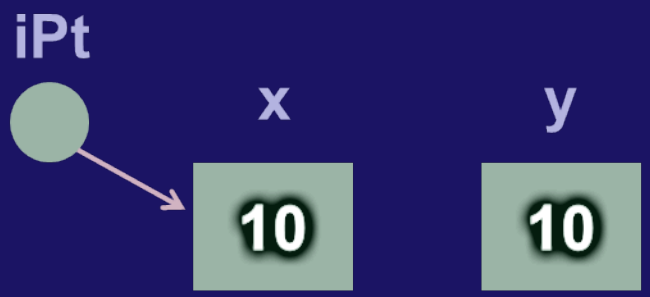

```
&x = 104; //nghĩa là địa chỉ biến x là 104.
iPt = &x; //khi đó iPt = 104.
```

Các toán tử con trỏ (tt)

- Toán tử * là toán tử một ngôi, trả về giá trị chứa trong vùng nhớ được trỏ đến bởi biến con trỏ.

```
iPt = &x;
*iPt = 10;
```

```
y = *iPt;
```



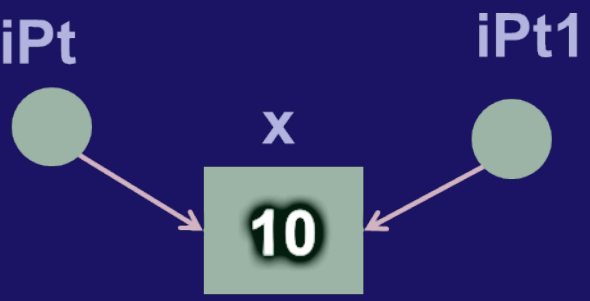
Các toán tử con trỏ (tt)

- Toán tử gán được sử dụng để gán giá trị cho con trỏ thông qua toán tử &.

```
iPt = &x;
```

- Địa chỉ của biến **x** được lưu vào biến con trỏ **iPt**.
- Cũng có thể gán giá trị cho con trỏ thông qua một biến con trỏ khác trỏ có cùng kiểu.

```
iPt1 = iPt;
```

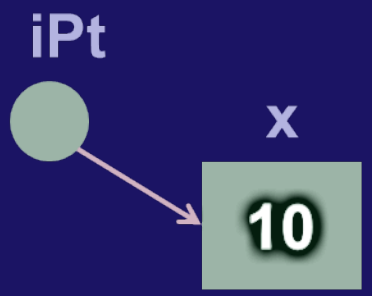


Các toán tử con trỏ (tt)

- Có thể gán giá trị cho các biến thông qua con trỏ

```
*iPt = 10;
```

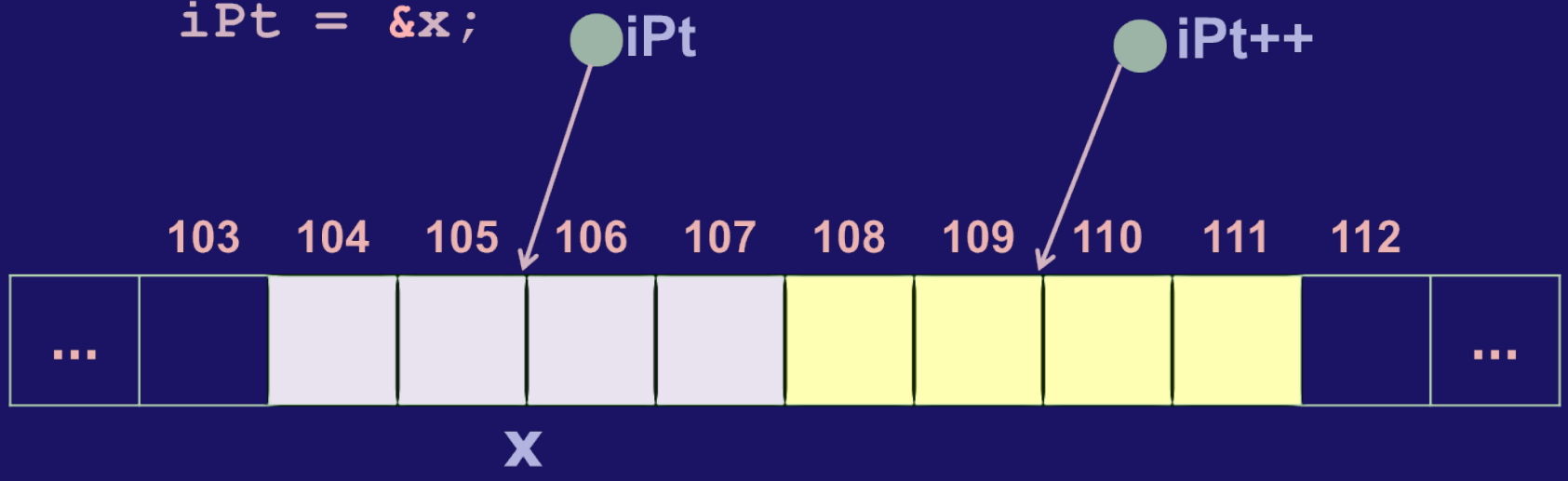
- Câu lệnh trên gán giá trị 10 cho biến x nếu con trỏ iPt đang trỏ đến chỗ nhớ của biến x.



Các toán tử con trỏ (tt)

- Chỉ có phép toán cộng và trừ trên con trỏ

```
int x, *iPt;
iPt = &x;
```




```
&x = 104; //nghĩa là địa chỉ biến x là 104
iPt = &x; //khi đó iPt = 104
iPt++; //khi đó iPt = 108
```

Các toán tử con trỏ (tt)

- Mỗi lần con trỏ được tăng trị, nó trỏ đến chỗ nhớ của phần tử kế tiếp.
- Mỗi lần con trỏ được giảm trị, nó trỏ đến chỗ nhớ của phần tử đứng trước nó.
- Tất cả con trỏ sẽ tăng hoặc giảm trị theo kích thước của kiểu dữ liệu mà chúng đang trỏ đến.

6.5. Con trỏ và mảng một chiều

- Khi khai báo một mảng ta được một con trỏ;
- Ví dụ:

```
int a[10];
```

a là một con trỏ trỏ vào phần tử mảng đầu tiên a[0] hay a = &a[0].

- Địa chỉ của một phần tử mảng có thể được biểu diễn theo hai cách:
  - Sử dụng ký hiệu & trước một phần tử mảng.
  - Ví dụ: `&a[i];`
  - Sử dụng một biểu thức trong đó chỉ số của phần tử được cộng vào tên của mảng.
  - Ví dụ: `(a+i)`

Con trỏ và mảng một chiều (tt)

```
int main() {
      int a[10] = {1,2,3,4,5,6,7,8,9,10};
      for (int i = 0; i < 10; i++)
      {
            cout<<"\na["<<i<<"]="<<*(a+i);
            cout<<"\n&a["<<i<<"]="<<(a+i);
      }
      return 0;
}
```

Cấp pháp bộ nhớ động (tt)

- Hàm malloc()
  - Cú pháp:

```
<ten_con_tro> = (KDL*) malloc(n*size);
```

  - Hàm malloc() cấp phát cho con trỏ vùng nhớ n*size byte.
  - Ví dụ:

```
int *iPt = (int*)malloc(5*sizeof(int));
```

  - Cấp phát cho con trỏ iPt 5 chỗ nhớ chứa 5 số nguyên kiểu int.

Cấp pháp bộ nhớ động (tt)

- Hàm calloc()
  - ✓ Hàm calloc() tương tự như malloc(), nhưng điểm khác biệt chính là mặc nhiên giá trị 0 được lưu vào không gian bộ nhớ vừa cấp phát.
  - ✓ Cú pháp:

```
<tên_con_trỏ> = (KDL*) calloc(n, size);
```

  - ✓ Ví dụ:

```
intPointer iPt = (int*)calloc(5,sizeof(int));
```

Cấp pháp bộ nhớ động (tt)

- Hàm realloc()
  - ✓ Có thể cấp phát lại cho một vùng đã được cấp (thêm/bớt số bytes) bằng cách sử dụng hàm realloc(), mà không làm mất dữ liệu.
  - ✓ Hàm realloc() nhận hai tham số
  - ✓ Tham số thứ nhất là con trỏ tham chiếu đến bộ nhớ
  - ✓ Tham số thứ hai là tổng số byte muốn cấp phát
  - ✓ Cú pháp:

    <ten_con_tro> = (KDL*) realloc(ten_con_tro, n*size);
  - ✓ Ví dụ:

```
iPt = (int*)realloc(iPt, 7*sizeof(int));
```

6.7. Giải phóng bộ nhớ

- Hàm free()
  - ✓ Được sử dụng để giải phóng bộ nhớ khi nó không cần dùng nữa.
  - ✓ Cú pháp:

```
free(iPt);
```

  - ✓ Hàm này giải phóng không gian nhớ được trỏ bởi iPt.
  - ✓ iPt phải được dùng trước đó với lời gọi hàm malloc(), calloc(), hoặc realloc().

6.8. Toán tử new và delete

- Toán tử **new** trong C++ dùng cấp phát bộ nhớ cho một biến con trỏ.
- Cú pháp:

```
<ten_con_tro> = new  <ten_KDL>[kich_thuoc];
```

- Ví dụ:

```
int *iPt = new int[5];
```

- Toán tử **delete**: Giải phóng bộ nhớ của con trỏ.
- Cú pháp:

```
delete <ten_con_tro>;
```

- Ví dụ:

```
delete iPt;
```

6.8. Toán tử new và delete

- Toán tử delete không thực sự xóa bất cứ điều gì. Nó giải phóng bộ nhớ và trao lại quyền sử dụng vùng nhớ được cấp phát cho hệ điều hành. Sau đó, hệ điều hành được tự do gán lại vùng nhớ đó cho một ứng dụng khác.
- Mặc dù câu lệnh "delete ptr" giống như việc xóa một biến, nhưng thực tế không phải! Biến con trỏ ptr vẫn có thể sử dụng như trước và có thể được gán một giá trị mới giống như bất kỳ biến nào khác.

* So sánh malloc và new

	malloc	new
Ngôn ngữ	C	C++
Kích thước cần cấp phát	Phải tính toán trước khi gọi	Tự động tính toán kích thước vùng nhớ
Cấp phát thất bại	Trả về NULL	
Cấp phát thành công	Trả về con trỏ void*, cần ép kiểu về dữ liệu cần dùng	Kiểu con trỏ là kiểu của đối tượng được cấp phát
Loại đối tượng	Hàm	Toán tử
Override	Không thể	Có thể
Thay đổi kích thước vùng nhớ cấp phát	Có thể thông qua hàm realloc()	Không thể thay đổi
Giải phóng bộ nhớ	Hàm free()	Toán tử delete

* Rò rỉ bộ nhớ trong C++

- Ví dụ 1:

```
void doSomething() {
        int *ptr = new int;
}
```

Giải phóng vùng nhớ khi ra khỏi phạm vi con trỏ

- Ví dụ 2

```
int value = 10;
int *ptr = new int; // cấp phát vùng nhớ
ptr = &value; // địa chỉ vùng nhớ cấp phát trước đó bị mất
```

Giải phóng vùng nhớ trước khi ghi đè con trỏ

- Ví dụ 3

```
int *ptr = new int;
ptr = new int; // địa chỉ vùng nhớ cấp phát trước đó bị mất, rò rỉ bộ nhớ
```

Giải phóng vùng nhớ trước khi cấp phát lại vùng nhớ

6.9. Con trỏ đa cấp và mảng 2 chiều

- Sử dụng con trỏ cấp 2 để cấp phát mảng động hai chiều.
- Ví dụ 1: cấp phát mảng 2 chiều số nguyên gồm m hàng n cột.

```
int **a, m = 4, n = 5;
```

Bước 1: Cấp phát số hàng (m hàng).

```
a = new int*[m];
```

Bước 2: Cấp phát số phần tử cho mỗi hàng (n phần tử).

```
for (int i = 0; i < m; i++)
    a[i] = new int[n];
```

- Giải phóng bộ nhớ

```
delete []a;
```

Con trỏ đa cấp và mảng 2 chiều (tt)

- Ví dụ 2: cấp phát mảng chuỗi để lưu trữ danh sách gồm họ và tên của n người.

```
char **list, m = 4, n = 5;
```

Bước 1: Cấp phát số phần tử theo danh sách (n phần tử).

```
list = new char*[n];
```

Bước 2: Cấp phát chuỗi lưu họ và tên người (n chuỗi).

```
for (int i = 0; i < n; i++)
    list[i] = new char[30];
```

- Giải phóng bộ nhớ

```
delete []list;
```

6.9. Kỹ thuật sử dụng con trỏ

1. Cài đặt chương trình (sử dụng con trỏ cấp phát bộ nhớ động) thực hiện các yêu cầu sau:
   - ✓ Nhập số nguyên dương n thỏa mãn 5≤n≤30.
   - ✓ Nhập một dãy d có n số nguyên.
   - ✓ Hiển thị dãy số d vừa nhập.
   - ✓ Nhập số nguyên x và số nguyên k thỏa mãn 1 ≤ k ≤ n, chèn x vào vị trí k trong dãy d.
   - ✓ Hiển thị dãy mới chèn.
   - ✓ Sắp xếp dãy d theo chiều tăng dần.
   - ✓ Hiển thị dãy mới sắp.

Thank you!