CÁC KHÁI NIỆM CƠ BẢN

Chương 1

Mục tiêu

- Cung cấp cách tinh chỉnh từng bước chương trình
- Các bước để giải một bài toán
- Một số khái niệm về thuật toán, kiểu dữ liệu, cấu trúc dữ liệu, …
- Khái niệm con trỏ và các phép toán trên nó

Nội dung

1. Từ bài toán đến chương trình
2. Kiểu dữ liệu trừu tượng, kiểu dữ liệu và cấu trúc dữ liệu
3. Con trỏ

1. Từ Bài Toán Đến Chương Trình

- Thuật toán
- Mã giả và tinh chỉnh từng bước
- Các bước để giải một bài toán

Thuật toán

- Khái niệm thuật toán
- Phân tích thuật toán

Khái niệm thuật toán ([1] p.5)

- Định nghĩa thuật toán
- Các tính chất cơ bản
- Ngôn ngữ biểu diễn thuật toán

Định nghĩa thuật toán

- *Thuật toán* là một thủ tục xác định bao gồm một dãy hữu hạn các bước cần thực hiện để thu được lời giải bài toán
- Một cách không hình thức, thuật toán là một thủ tục tính toán bất kỳ được định nghĩa tốt (*well-defined*) nhận một hoặc một tập giá trị được gọi là đầu vào (*input*) và sinh một hoặc một tập giá trị như được gọi là đầu ra (*output*)
- Như vậy, một thuật toán là một dãy các bước tính toán để chuyển input thành output

Các tính chất cơ bản

- Đầu vào (input): Dữ liệu mà thuật toán nhận vào từ một tập nào đó
- Đầu ra (output): Dữ liệu mà thuật toán đưa ra ứng với lời giải bài toán
- Chính xác (*precision*): Các bước của thuật toán phải được mô tả chính xác

Các tính chất cơ bản

- Hữu hạn (*finiteness*): Thuật toán phải đưa được đầu ra sau một số hữu hạn bước với mọi đầu vào
- Đơn trị (*uniqueness*): Các kết quả trung gian của từng bước phải được xác định một cách đơn trị và chỉ phụ thuộc vào đầu vào và kết quả các bước trước
- Tổng quát (*generality*): Thuật toán có thể áp dụng để giải mọi bài toán có dạng đã cho (một lớp các bài toán)

Ngôn ngữ biểu diễn thuật toán

- Bằng ngôn ngữ tự nhiên (*natural language*)
- Bằng mã giả (pseudocode)
- Bằng ngôn ngữ lập trình cấp cao (*high programming languages*) như Pascal, C, C++, Algol, vv…

Ngôn ngữ biểu diễn thuật toán

- Sự khác nhau giữa mã giả và mã thật là mã giả không quan tâm *cụ thể* đến các vấn đề thiết kế phần mềm
- Các vấn đề về sự trừu tượng dữ liệu, xử lý lỗi thường được bỏ qua trong mã giả để có thể truyền đạt được ý nghĩa thuật toán rõ ràng hơn

Phân tích thuật toán ([1] p.21)

- Phân tích thuật toán là quá trình tìm ra những đánh giá chi phí về tài nguyên của hệ thống cần thiết để thực hiện thuật toán. Nói cách khác phân tích thuật toán là tính toán độ phức tạp của thuật toán
- Tài nguyên của hệ thống như bộ nhớ, CPU, băng thông, … nhưng thường thì chúng ta muốn đo lường về thời gian mà thuật toán thực hiện

Phân tích thuật toán

- Trước khi phân tích thuật toán chúng ta phải có một mô hình kỹ thuật hiện thực (mô hình tính toán mà trên đó thuật toán làm việc) bao gồm một mô hình cho các tài nguyên và chi phí của chúng
- Chúng ta giả sử rằng một mô hình tính toán với một bộ xử lý chung, bộ nhớ truy cập ngẫu nhiên (RAM) như là một công cụ kỹ thuật hiện thực và hiểu rằng thuật toán của chúng ta sẽ được hiện thực như một chương trình máy tính

Phân tích thuật toán

- Trong mô hình RAM các chỉ thị (lệnh) được thực thi tuần tự, không có thao tác đồng thời
- Phân tích thuật toán yêu cầu nhiều công cụ toán học như lý thuyết đếm, giải tích tổ hợp, mô hình hệ thức truy hồi và lý thuyết xác suất vv…
- Tuy nhiên, chúng ta chỉ quan tâm đến việc phân tích thời gian chạy (*running time*) của thuật toán
- Độ phức tạp về thời gian của thuật toán là một hàm $T(n, m, \ldots)$ phụ thuộc vào kích thước n, m, ... của input

Phân tích thuật toán

- Các máy cụ thể có tốc độ khác nhau, thì thời gian thực hiện trên đó sẽ chỉ khác nhau một hằng so với $T(n, m, \ldots)$
- Thời gian chạy của thuật toán trên một đầu vào cụ thể là số các thao tác (phép toán) nguyên thuỷ hoặc các bước được thực hiện
- Như vậy có thể coi thời gian chạy là độc lập với máy cụ thể

Mã giả và tinh chỉnh từng bước ([3] p. 23)

- Tinh chỉnh từng bước là phương pháp thiết kế giải thuật gắn liền với lập trình
- Đầu tiên, bài toán được thể hiện tổng quan bằng ngôn ngữ tự nhiên được gọi là *mã giả*
- Sau đó, nó được chi tiết hóa dần và hướng về ngôn ngữ lập trình đã chọn
- Cuối cùng, bài toán được thể hiện bằng các lệnh của ngôn ngữ lập trình đã chọn

Mã giả và tinh chỉnh từng bước

- Ví dụ: Viết chương trình sắp xếp mảng số nguyên a có n phần tử theo thứ tự tăng dần
- Tinh chỉnh lần 1:

```
for i = 0 to n - 1
{
    - Tìm số nhỏ nhất a[j] (i <= j < n)(1)
    - Đổi chỗ a[i] với a[j]              (2)
}
```

Mã giả và tinh chỉnh từng bước

- Tinh chỉnh lần 2:
  - (1) có thể được viết:

```
int j = i;
for k = j + 1 to n - 1
    if(a[k] < a[j])
        j = k;
```

  - (2) có thể được viết:

```
int temp = a[i];
a[i] = a[j];
a[j] = temp;
```

Mã giả và tinh chỉnh từng bước

- **Tinh chỉnh lần 3 (kết quả):**

```
void sort(int a[], int n)
{
   for(int i = 0; i < n - 1; i++)
   {
      int j = i;
      for(int k = j + 1; k < n; k++)
         if(a[k] < a[j])
            j = k;
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

Các bước để giải một bài toán

- Các bước chung để giải bài toán đó là:
  - Thiết kế: Vấn đề được phân tích và cách giải quyết vấn đề đó được thiết kế, kết quả là một *giải thuật* để giải quyết vấn đề
  - Viết mã: Giải thuật được viết theo cú pháp của một ngôn ngữ cấp cao (chẳng hạn C++), kết quả là một *chương trình*
  - Kiểm tra, thực thi và phát hiện lỗi: Chương trình được kiểm tra cẩn thận và nếu có bất kỳ lỗi nào xảy ra thì chương trình đó phải được sửa lại
  - Bảo trì: Chương trình được cập nhật, sửa đổi theo yêu cầu của những người sử dụng nó

2. Kiểu Dữ Liệu Trừu Tượng, Kiểu Dữ Liệu, Cấu Trúc Dữ Liệu

- **Kiểu dữ liệu trừu tượng ([2] p. 31):**
  - **Sự mô tả một cấu trúc dữ liệu theo thao tác thực hiện (thay vì theo chi tiết cài đặt) được gọi là kiểu dữ liệu trừu tượng (*abstract data type* – ADT)**
  - **Động cơ chính của việc phát triển các kiểu dữ liệu trừu tượng nhằm làm giảm độ phức tạp của những chương trình lớn**

Kiểu dữ liệu trừu tượng, kiểu dữ liệu, cấu trúc dữ liệu

- **Kiểu dữ liệu:**
  - **Kiểu dữ liệu là một sự qui định về cấu trúc, kích thước, và giá trị của dữ liệu, cũng như cách xử lý, biểu diễn chúng**
  - **Một dữ liệu bao giờ cũng thuộc về một kiểu dữ liệu nào đó**
  - **Mỗi ngôn ngữ lập trình đều có định nghĩa sẵn một số kiểu dữ liệu cơ bản cho nó như `int`, `char`, `double`, ...**
  - **Trong nhiều trường hợp, nếu chỉ với các kiểu dữ liệu cơ bản thì không đủ để phản ánh các bài toán trong thế giới thực**

Kiểu dữ liệu trừu tượng, kiểu dữ liệu, cấu trúc dữ liệu

- Dẫn đến là cần phải xây dựng các kiểu dữ liệu mới, dựa trên việc tổ chức, liên kết các thành phần dữ liệu có kiểu dữ liệu đã định nghĩa sẵn (kiểu dữ liệu cơ bản)
- Những kiểu dữ liệu được xây dựng như thế gọi là những kiểu dữ liệu có cấu trúc
- Đa số các ngôn ngữ lập trình cấp cao cho phép người sử dụng định nghĩa một số kiểu dữ liệu có cấu trúc mới như mảng, chuỗi, `struct`, . . .

Kiểu dữ liệu trừu tượng, kiểu dữ liệu, cấu trúc dữ liệu

- Cấu trúc dữ liệu ([3] p. 9) :
  - Các dữ liệu thuộc kiểu dữ liệu có cấu trúc được gọi là các cấu trúc dữ liệu (*data structure*). Ví dụ mảng, `struct`, danh sách liên kết, … là các cấu trúc dữ liệu (CTDL)
  - Với một cấu trúc dữ liệu đã chọn, phải có giải thuật (phép toán) xử lý tương ứng
  - Khi cấu trúc dữ liệu thay đổi, giải thuật cũng phải thay đổi theo để tránh xử lý gượng ép trên CTDL không phù hợp
  - Việc chọn một cấu trúc dữ liệu sẽ dẫn đến giải thuật hiệu quả (hay kém hiệu quả) hơn so với cái khác

Kiểu dữ liệu trừu tượng, kiểu dữ liệu, cấu trúc dữ liệu

- Trong môn học này, chúng ta sẽ xét một số cấu trúc dữ liệu như danh sách liên kết (đơn, kép, vòng) , ngăn xếp (stack), hàng đợi (queue), cây (tree), … và các phép toán trên chúng

3. Con Trỏ

- Khái niệm
- Biến con trỏ
- Một số phép toán con trỏ
- Con trỏ và mảng một chiều
- Tham số con trỏ
- Phép toán `new`
- Phép toán `delete`
- Con trỏ và đối số dòng lệnh
- Tham số của hàm `main()`

Khái niệm

- *Con trỏ* (*pointer*) hay *biến con trỏ* được sử dụng để trỏ (chỉ) đến vùng nhớ của những mảng mà dung lượng của nó được chỉ định trong thời gian thực thi hoặc những cấu trúc dữ liệu khác mà kích thước của nó có thể "co giãn" (*scalable/grow and shrink*) lúc chương trình thực thi

Biến con trỏ

- Chương trình ví dụ, Figure 1.1
- Lưu ý cách khai báo

```
int* iPtr = &i;
```

Có thể viết:

```
int* iPtr;
iPtr = &i;
```

  - Dấu hoa thị * cho biết iPtr có thể chứa địa chỉ của một biến
  - Dấu & là toán tử tiền tố một ngôi
    - Nó trả về địa chỉ của biến i

Biến con trỏ

- Chú ý là dấu hoa thị phải được đặt trước mỗi biến con trỏ trong khai báo:

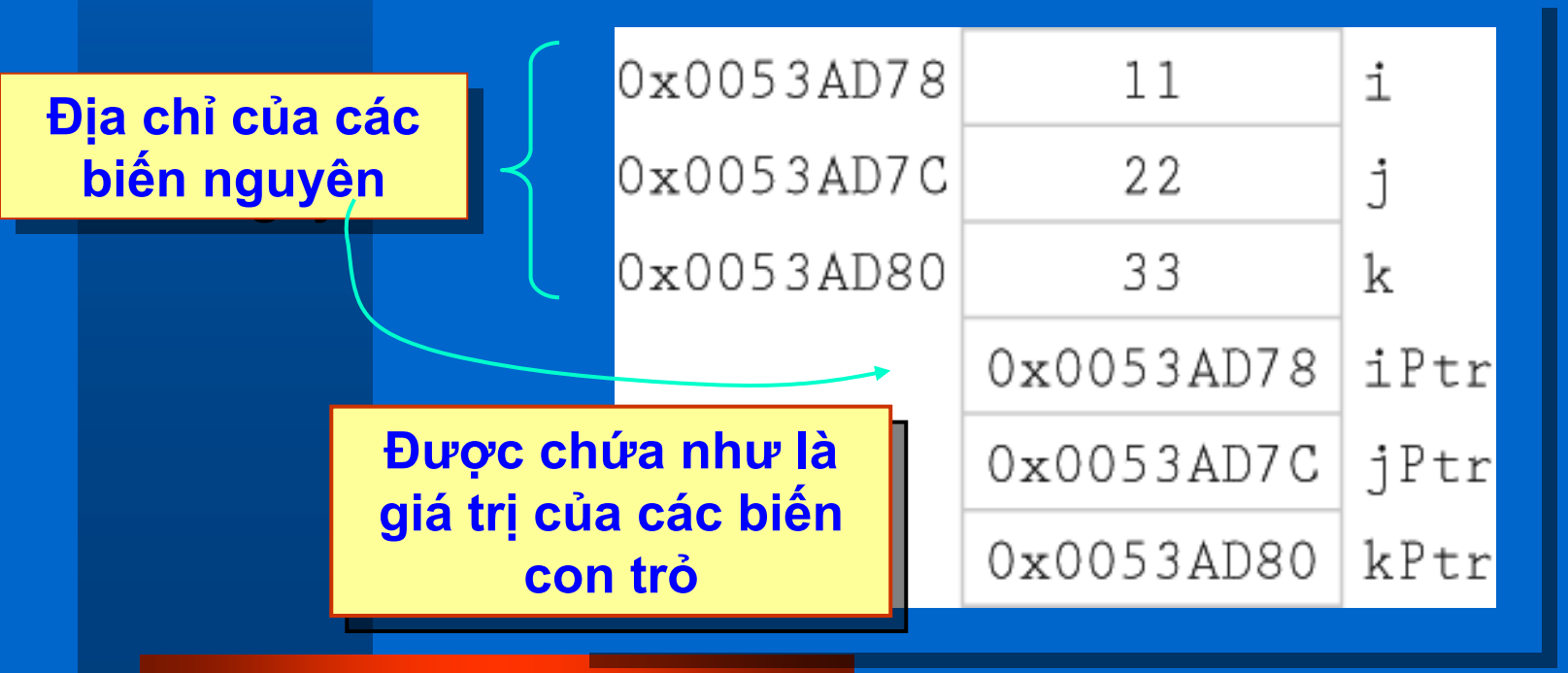
```
int *intPtr1, *intPtr2;
```

- Có thể sử dụng `typedef` để làm cho chương trình dễ đọc hơn
  - Điều này sẽ không cần sử dụng dấu hoa thị trong khai báo biến con trỏ

```
typedef int* intPointer;
intPointer intPtr1, intPtr2;
```

Biến con trỏ

- Con trỏ và địa chỉ:
  - Kết quả của sự khai báo và khởi tạo ba biến con trỏ trong chương trình



Một số phép toán con trỏ

- Khởi tạo:
  - Biến con trỏ có thể được khởi tạo trong khai báo, đối tượng phải cùng kiểu với con trỏ

```
int i = 11;
int* iPtr = &i;
```

  - Cách viết sau là lỗi cú pháp:

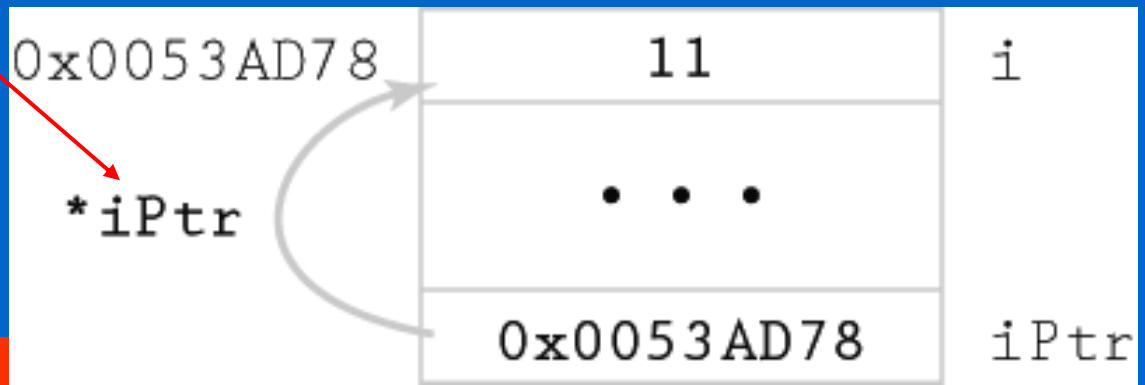
```
double doubleVar;
int* iPtr = &doubleVar;     // LỖI
```

Một số phép toán con trỏ

- Giá trị 0 (hay NULL) có thể được gán đến bất kỳ biến con trỏ nào và được gọi là *địa chỉ rỗng* (*null address*)

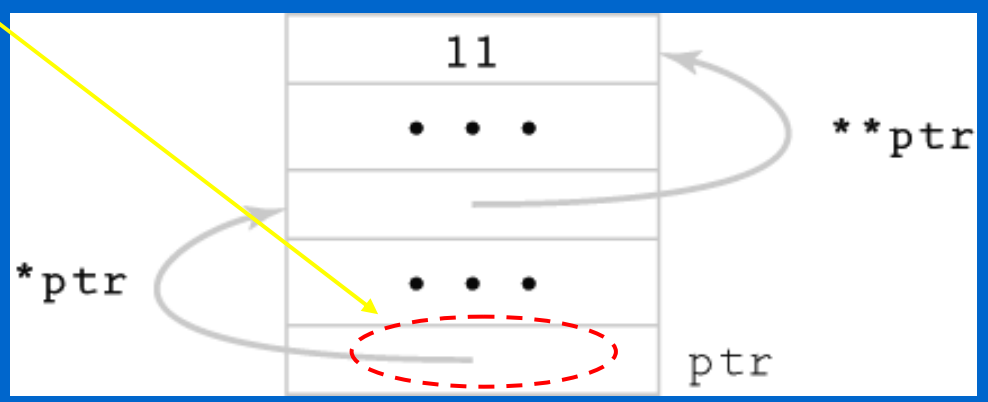
```
char* cPtr = 0;   // địa chỉ rỗng
int *iPtr = 0;
```

- Toán tử gián tiếp `*` :
  `*iPtr` là giá trị của đối tượng mà `iPtr` trỏ đến



Một số phép toán con trỏ

- Chú ý là toán tử gián tiếp có thể được áp dụng cho nhiều mức
  - Con trỏ trỏ đến con trỏ khác



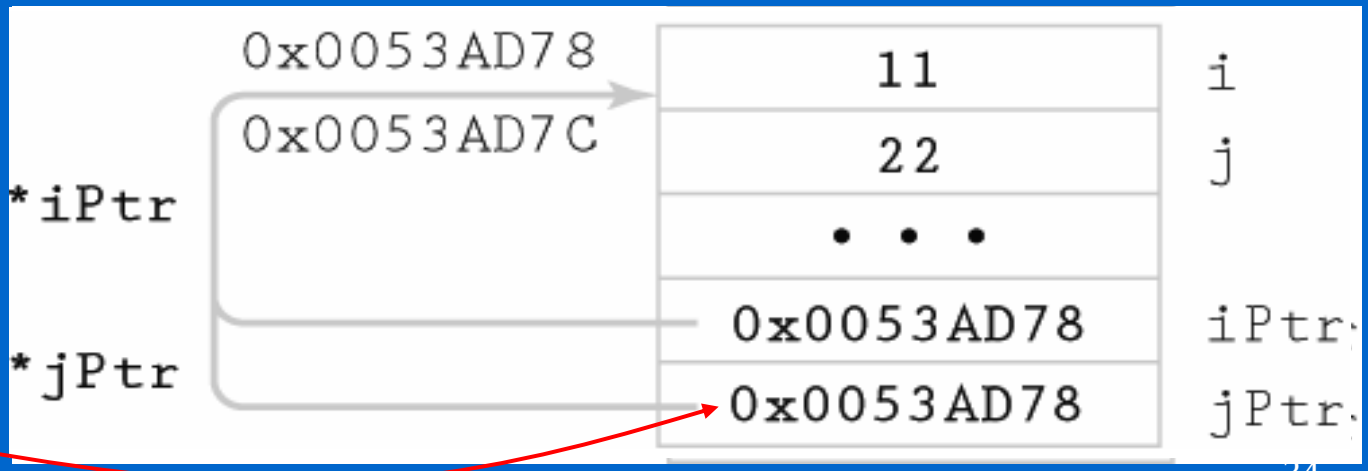
- Nhập/xuất: Toán tử * có thể được sử dụng trong các câu lệnh nhập và xuất

```
cout << *iPtr;
cin >> *iPtr;
```

Một số phép toán con trỏ

- **Phép gán:**
  - Biến con trỏ có thể gán đến biến con trỏ khác
  - Các con trỏ phải cùng kiểu

```
jPtr = iPtr;
```



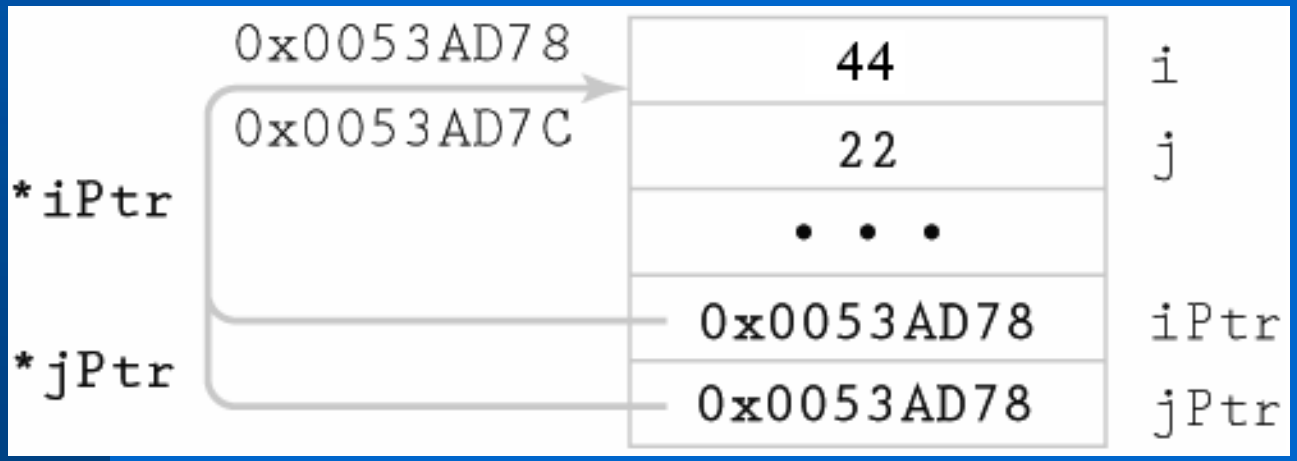
Bây giờ cả hai con trỏ cùng trỏ đến biến `i`

Một số phép toán con trỏ

- Khi cả hai `iPtr` và `jPtr` cùng trỏ đến một đối tượng, việc thực hiện gán một giá trị đến `*iPtr` hay `*jPtr` sẽ cho cùng kết quả

```
*jPtr = 44;
cout << *iPtr;
```

Giá trị gì được in?



- Lưu ý là toán tử `*` có thể xuất hiện cả hai phía trong câu lệnh gán

Một số phép toán con trỏ

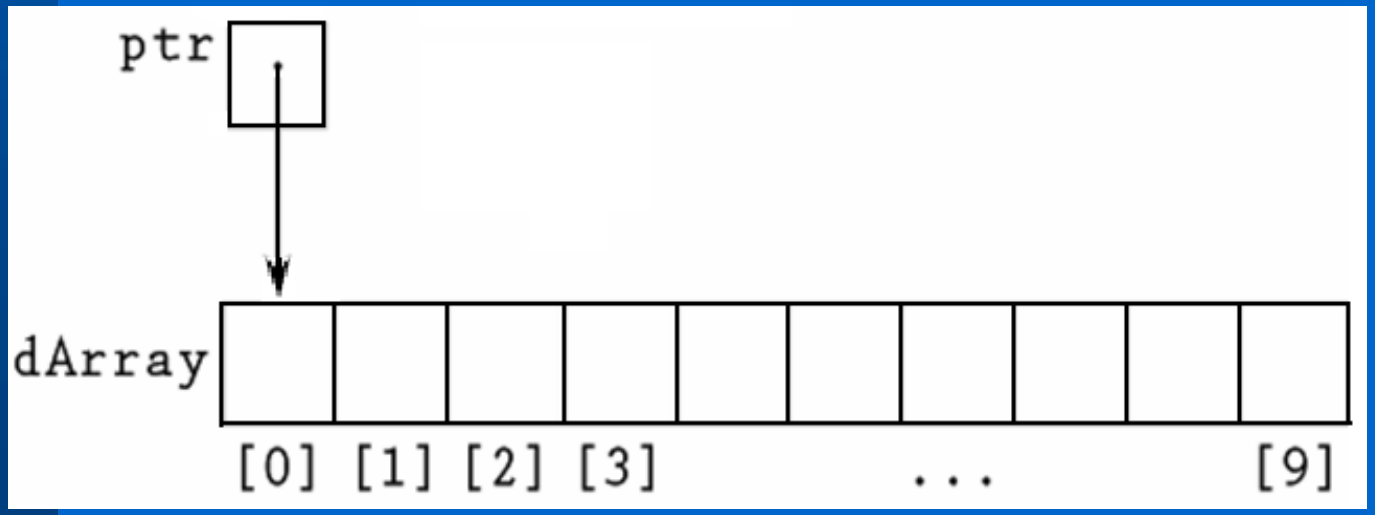
- So sánh:
  - Các toán tử quan hệ có thể được sử dụng để so sánh hai con trỏ cùng kiểu
  - Toán tử thường dùng để so sánh là `==` và `!=` để kiểm tra xem hai con trỏ có trỏ đến cùng vị trí nhớ không
- Số học con trỏ:
  - Biểu thức `iPtr++` sẽ tăng giá trị địa chỉ được chứa trong `iPtr`
  - Đại lượng được tăng là số byte của đối tượng mà con trỏ trỏ đến

Con trỏ và mảng một chiều

- **Xét các khai báo:**

```
double  dArray[10];
double *ptr = &(dArray[0]);
// hay double *ptr = dArray
```

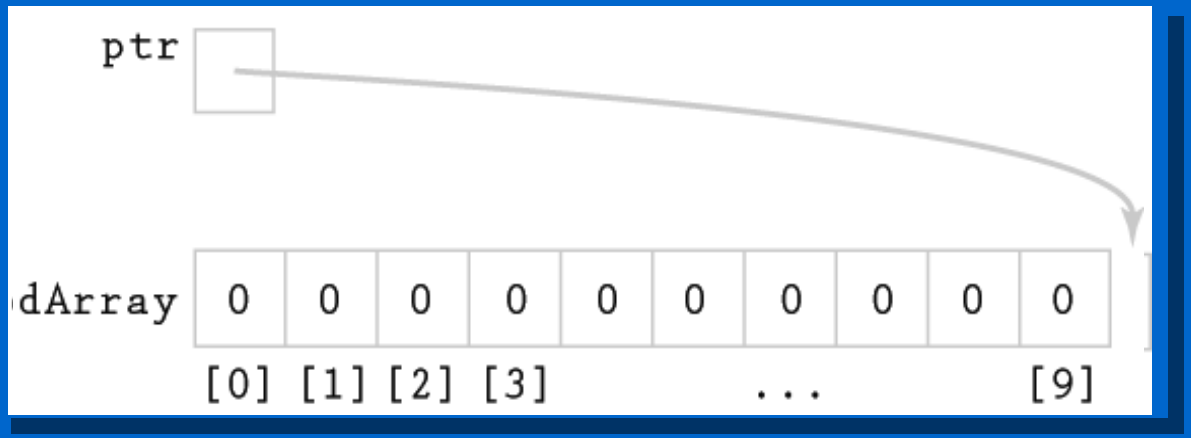
- **Kết quả:**



Con trỏ và mảng một chiều

- Có thể sử dụng số học con trỏ để thiết lập giá trị cho các phần tử

```
for(int i = 0; i < 10; i++)
  { *ptr = 0;     ptr++; }
```



- Nếu chúng ta tiếp tục tăng `ptr` trong vòng lặp thì tất cả phần tử trong mảng sẽ được khởi tạo

Tham số con trỏ

- Tham số con trỏ được xem như là tham số tham chiếu
- Khi tham số là con trỏ, đối số tương ứng phải là địa chỉ của đối tượng cần truyền
- Đối tượng này phải cùng kiểu với con trỏ
- Kiểu trả về của một hàm có thể là con trỏ
- Figure 1.2, Sample

Phép toán new

- Khi khai báo một mảng, ví dụ:

```
const int CAPACITY = 10;
double arrayName [CAPACITY];
```

  - Dành vùng nhớ cho 10 giá trị double
  - Vùng nhớ không đổi trong thời gian thực thi
- Nhược điểm:
  - Nếu kích thước mảng nhiều hơn số giá trị cần chứa, dư thừa bộ nhớ
  - Nếu kích thước mảng quá nhỏ, hiện tượng tràn mảng sẽ xảy ra

Phép toán new

- Để khắc phục các trở ngại trên, chúng ta cần:
  - Yêu cầu cấp phát vùng nhớ vừa đủ khi cần
  - Giải phóng vùng nhớ khi không sử dụng
- Phép toán new, ví dụ:

```
int* intPtr;
intPtr = new int;
```

  - Yêu cần cấp vùng nhớ trong thời gian thực thi
  - Vùng nhớ được cấp phát đủ lớn để chứa giá trị có kiểu được chỉ định
  - Toán tử new trả về địa chỉ của vùng nhớ vừa được cấp

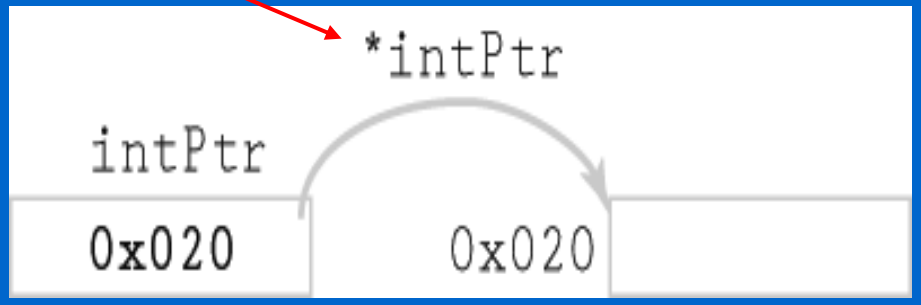
Phép toán new

- Các khai báo

```
int* intPtr;
intPtr = new int;
```

được cấp vùng nhớ là biến nặc danh:

- Nó không có tên, chỉ có địa chỉ
- Biến này chỉ có thể được truy xuất gián tiếp thông qua con trỏ



Phép toán new

- Với khai báo `int anArray[10];`
  thì `anArray` là địa chỉ cơ sở của mảng
- Có thể dùng toán tử `new` để thay thế khai báo trên:

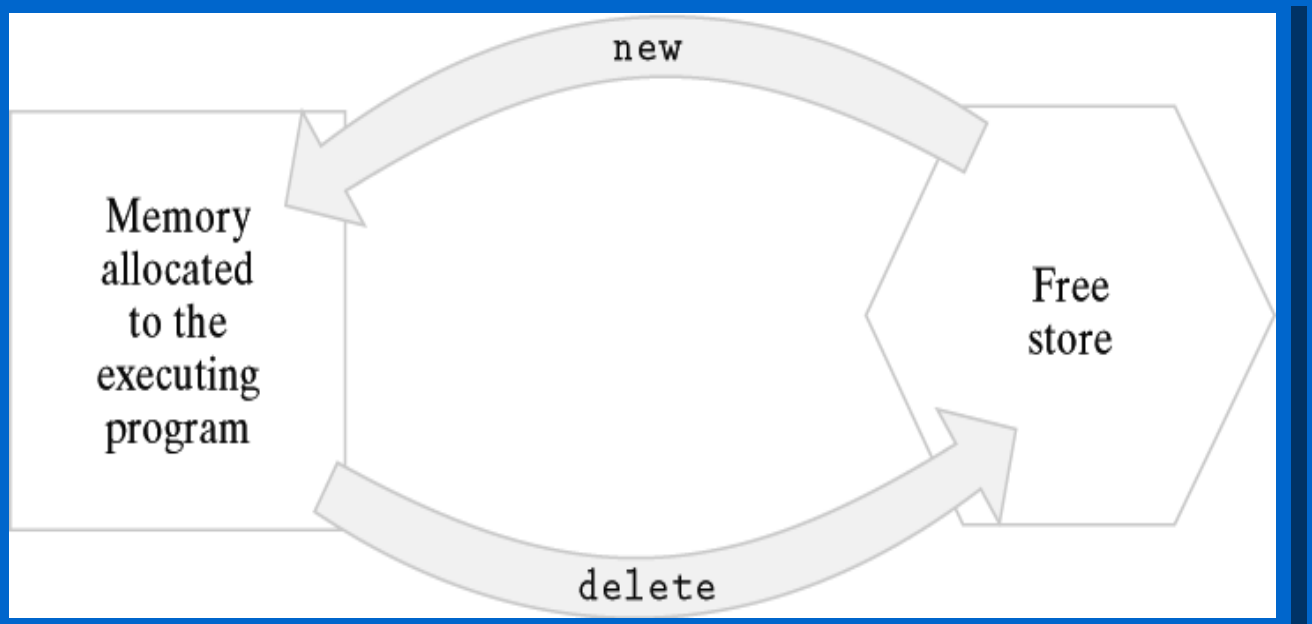
```
int *arrayPtr;
arrayPtr = new int[10];
```

- Tương tự, `arrayPtr` bây giờ là địa chỉ cơ sở của mảng – được cấp phát lúc thực thi

Vùng nhớ được cấp phát vừa đủ để chứa các phần tử trong mảng.

Phép toán `delete`

- Phép toán `delete` dùng để:
  - Giải phóng vùng nhớ được cấp phát bằng phép toán new
  - Cấp phát lại khi cần đến sau này



Phép toán `delete`

- Giải phóng vùng nhớ cho một biến

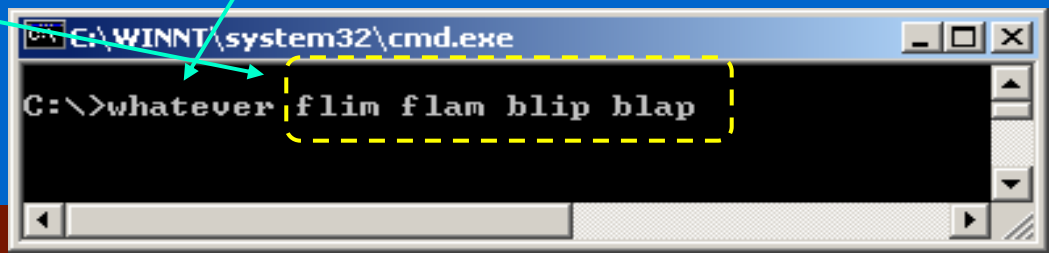
```
int *intPtr = new int;
delete intPtr;
```

- Giải phóng vùng nhớ cho một mảng

```
cin >> numVal;
double *dPtr = new double[numVal];
delete [] dPtr;
```

Con trỏ và đối số dòng lệnh

- Làm thế nào để truyền các đối số đến hàm `main()` trong C++
  - Có thể thực hiện với một mảng con trỏ
- Làm thế nào để hàm `main()` được gọi khi chương trình được thực thi
  - Tại dấu nhắc lệnh, nhập tên chương trình muốn thực thi
  - Bất kỳ chuỗi nhập nào sau tên chương trình sẽ là đối số của hàm `main()`



Tham số của hàm `main()`

- Cách tiêu biểu để khai báo các tham số của hàm `main()` là:

```
int main (int argc, char* argv[])
{   ...  }
```

- `argc` là số đối số nhận được
- `argv` là mảng con trỏ kiểu `char`
- Figure 1.3, Sample