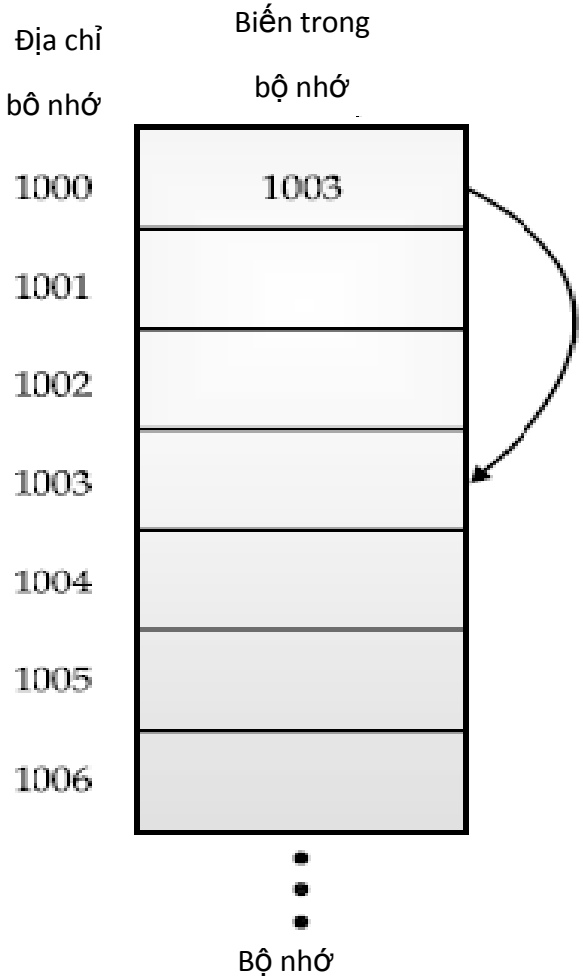
CHƯƠNG 5
CON TRỎ
(Pointers)

1. Giới thiệu

- Một con trỏ là **1 biến chứa một địa chỉ bộ nhớ**. Địa chỉ này là vị trí của một đối tượng khác trong bộ nhớ.
- Nếu một biến chứa địa chỉ của một biến khác, biến thứ nhất được gọi là trỏ đến biến thứ hai.

1. Giới thiệu



Một biến được cấp phát ô nhớ tại địa chỉ 1000 có giá trị là địa chỉ (1003) của 1 biến khác. Biến thứ nhất được gọi là con trỏ.

2. Khai báo biến con trỏ

- **Cú pháp**:

type *pointerVariable;

type: xác định kiểu dữ liệu của biến mà con trỏ trỏ đến.

Ví dụ:

int *a;

a

3. Toán tử con trỏ (*pointer operators*)

- ***Toán tử &*** là toán tử 1 ngôi, trả về địa chỉ bộ nhớ của toán hạng của nó.
  - Toán tử & dùng để gán địa chỉ của biến cho biến con trỏ

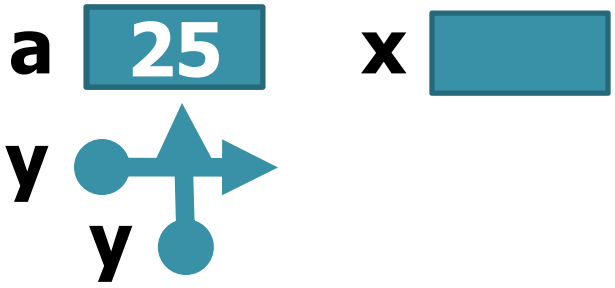
    Cú pháp:

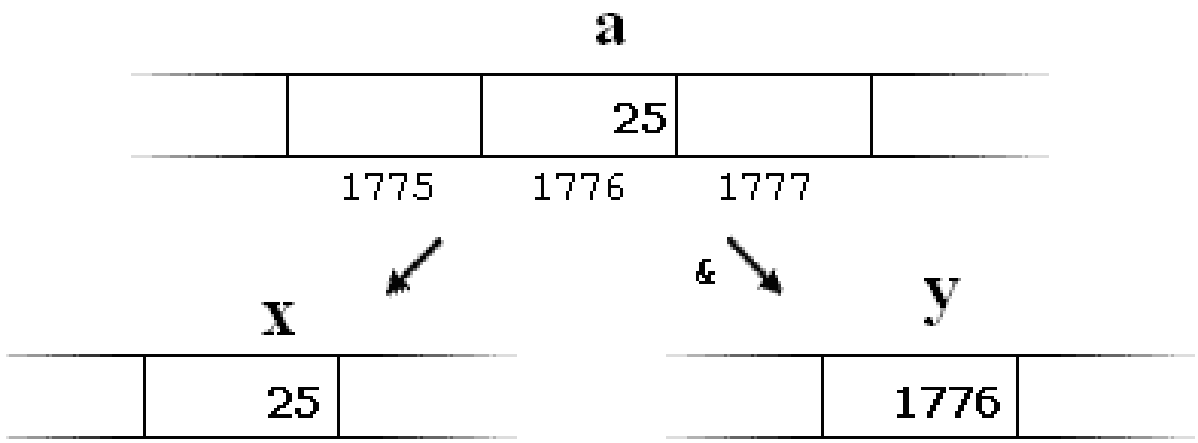
    <Tên biến con trỏ>=&<Tên biến>

3. Toán tử con trỏ (*pointer operators*)

- ***Ví dụ:***

```
int a=25, x;
int *y;
x=a;
y=&a;
```





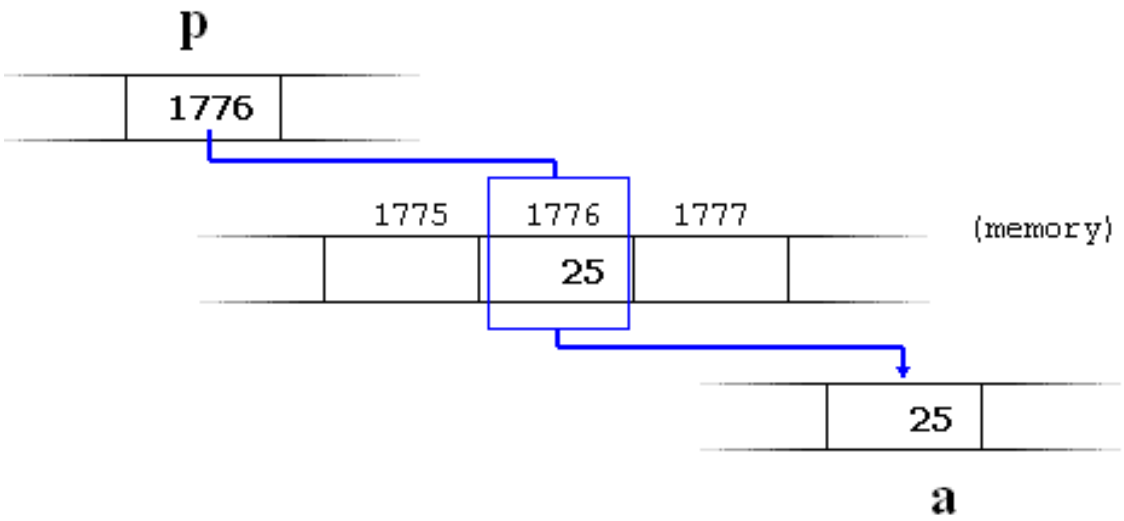
3. Toán tử con trỏ (*pointer operators*)

- ***Toán tử **** : là toán tử một ngôi trả về giá trị tại địa chỉ con trỏ trỏ đến.

Cú pháp:

***<Tên biến con trỏ>**

Ví dụ: **a=*p ;**



4. Các thao tác trên con trỏ

- ***Lệnh gán con trỏ***

  Có thể dùng phép gán để gán giá trị của một con trỏ cho một con trỏ khác có cùng kiểu

  Ví dụ:

```
int x;
int *p1, *p2;
p1 = &x;
p2 = p1;
```

  Sau khi đoạn lệnh trên được thực hiện, cả hai p1 và p2 cùng trỏ đến biến x.

4. Các thao tác trên con trỏ

- ***Phép toán số học trên con trỏ***
  - Chỉ có 2 phép toán sử dụng trên con trỏ là phép cộng và trừ
  - Khi cộng (+) hoặc trừ (-) 1 con trỏ với 1 số nguyên N; kết quả trả về là 1 con trỏ. Con trỏ này chỉ đến vùng nhớ cách vùng nhớ của con trỏ hiện tại một số nguyên lần kích thước của kiểu dữ liệu của nó.

4. Các thao tác trên con trỏ

Ví dụ :

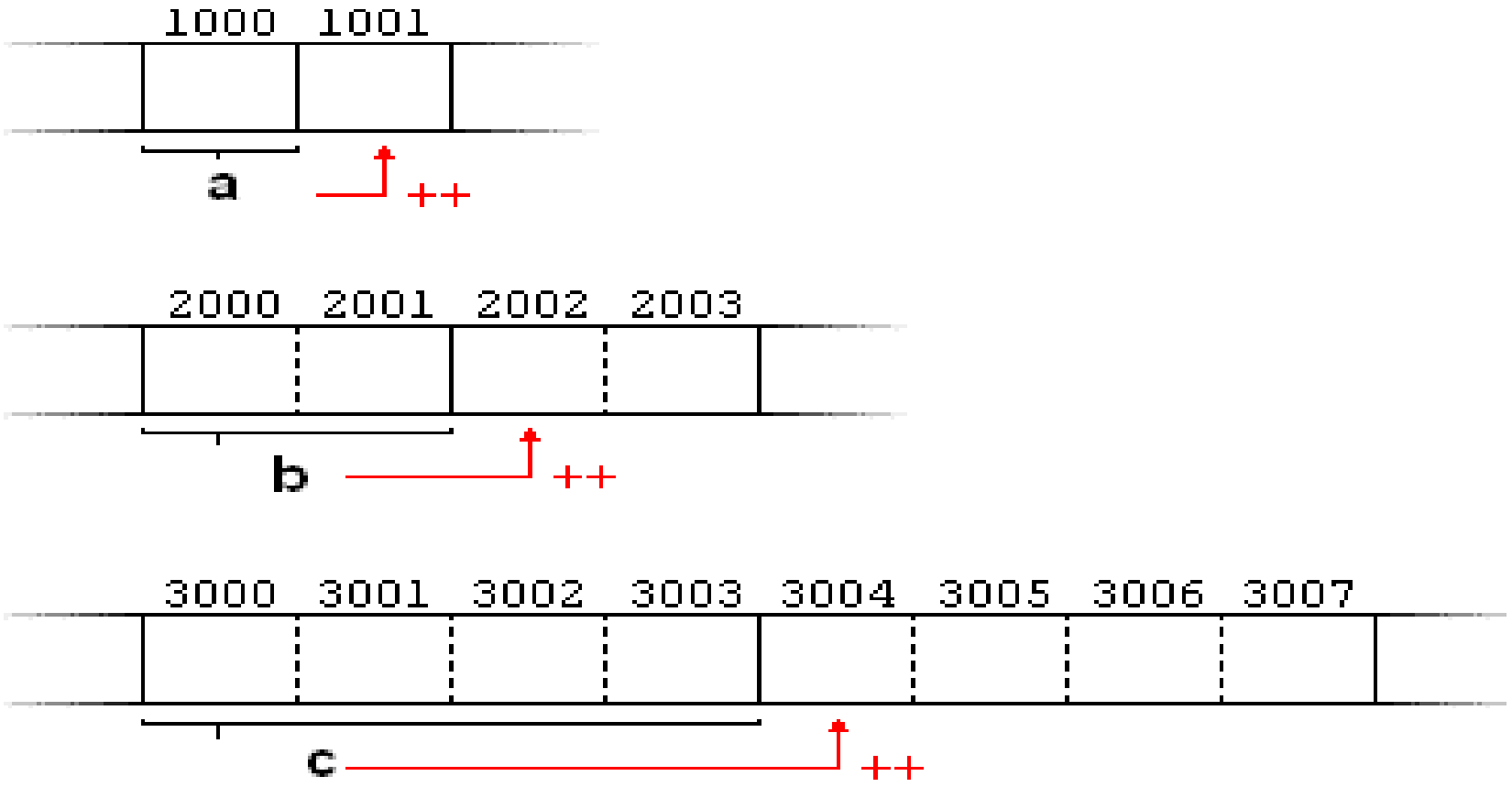
```
char *a;
short *b;
long *c;
```

Các con trỏ a, b, c lần lượt trỏ tới ô nhớ **1000**, **2000** và **3000**.

Cộng các con trỏ với một số nguyên:

```
a = a + 1;//con trỏ a dời đi 1 byte
b = b + 1;//con trỏ b dời đi 2 byte
c = c + 1; //con trỏ c dời đi 4 byte
```

4. Các thao tác trên con trỏ



4. Các thao tác trên con trỏ

- ***Lưu ý:*** cả hai toán tử tăng (++) và giảm (--) đều có quyền ưu tiên lớn hơn toán tử *****

  Ví dụ: *p++;

  Lệnh *p++ tương đương với *(p++) : thực hiện là tăng **p** (địa chỉ ô nhớ mà nó trỏ tới chứ không phải là giá trị trỏ tới).

4. Các thao tác trên con trỏ

Ví dụ:

```
*p++ = *q++;
```

- Cả hai toán tử tăng (++) đều được thực hiện sau khi giá trị của *q được gán cho *p và sau đó cả q và p đều tăng lên 1. Lệnh này tương đương với:

```
*p = *q;
p++;
q++;
```

4. Các thao tác trên con trỏ

```
#include <iostream.h>
#include<conio.h>
void main ()
{
   int a = 20, b = 15, *pa, *pb, temp;
   pa = &a; // con trỏ pa chứa địa chỉ của a
   pb = &b; // con trỏ pb chứa địa chỉ của b
   temp = *pa;
   *pa = *pb;
   *pb = temp;
   cout << "a = " << a << endl;
   cout << "b = " << b;
}
```

```
// kết quả xuất ra
màn hình
a = 15
b = 20
```

5. Cấp phát bộ nhớ động

- Con trỏ cung cấp sự hổ trợ cho cấp phát bộ nhớ động trong C/C++.
- Cấp phát động là phương tiện nhờ đó một chương trình có thể dành được thêm bộ nhớ trong khi đang thực thi, giải phóng bộ nhớ khi không cần thiết
- C/C++ hổ trợ hai hệ thống cấp phát động: một hệ thống được định nghĩa bởi C và một được định nghĩa bởi C++.

5. Cấp phát bộ nhớ động

- **Cấp phát động được định nghĩa bởi C**
  - ***Vùng nhớ Heap*** được sử dụng cho việc cấp phát động các khối bộ nhớ trong thời gian thực thi chương trình. Gọi là bộ nhớ động.
  - ***Hàm malloc() và free()*** dùng để cấp phát và thu hồi bộ nhớ, trong thư viện **stdlib.h**

5. Cấp phát bộ nhớ động

- **Hàm malloc():** cấp phát bộ nhớ động.
  - Prototype của hàm có dạng

    void *malloc(length)
  - ***length***: là số byte muốn cấp phát.
  - Hàm malloc() trả về một con trỏ có kiểu void, do đó có thể gán nó cho con trỏ có kiểu bất kỳ.
  - Sau khi cấp phát thành công, hàm malloc() trả về địa chỉ của byte đầu tiên của vùng nhớ được cấp phát từ heap. Nếu không thành công (không có đủ vùng nhớ rỗi yêu cầu), hàm malloc() trả về null.

5. Cấp phát bộ nhớ động

- Ví dụ:

```
char *p;
p = (char *) malloc(1000); //cấp phát 1000
  bytes
```

Vì hàm malloc() trả về con trỏ kiểu **void**, nên phải ép kiểu (casting) nó thành con trỏ **char** cho phù hợp với biến con trỏ p.

5. Cấp phát bộ nhớ động

- Ví dụ:

  int *p;

  p = (int *) malloc(50*sizeof(int));

  Toán tử sizeof để xác định kích thước kiểu dữ liệu int.

5. Cấp phát bộ nhớ động

- Kích thước của heap không xác định nên khi cấp phát bộ nhớ phải kiểm tra giá trị trả về của hàm malloc() để biết là bộ nhớ có được cấp phát thành công hay không.

  Ví dụ:

```
p = (int *)malloc(100);
if(p == NULL)
{
  cout << "Khong du bo nho";
  exit(1);
}
```

5. Cấp phát bộ nhớ động

- **Hàm free():** Trả về vùng nhớ được cấp phát bởi hàm malloc().
- Cú pháp:

void free(void *p);

p là con trỏ đến vùng nhớ đã được cấp phát trước đó bởi hàm malloc().

5. Cấp phát bộ nhớ động

- **Cấp phát động được định nghĩa bởi C++**

  C++ cung cấp hai toán tử cấp phát bộ nhớ động: **new** và **delete**.

  - **Toán tử new** cấp phát bộ nhớ và trả về một con trỏ đến byte đầu tiên của vùng nhớ được cấp phát.
  - **Toán tử delete** thu hồi vùng nhớ được cấp phát trước đó bởi toán tử new.

5. Cấp phát bộ nhớ động

- **Cú pháp**:

  p = new type;

  delete p;

- p là một biến con trỏ nhận địa chỉ của vùng nhớ được cấp phát đủ lớn để chứa 1 đối tượng có kiểu là type

5. Cấp phát bộ nhớ động

- Ví dụ:

```
#include <iostream.h>
#include <new>
int main()
{
  int *p;
  p = new int; // allocate space for an int
  *p = 100;
  cout << "At " << p << " ";
  cout << "is the value " << *p << "\n";
  delete p;
  return 0;
}
```

6. Con trỏ void (*void pointers*)

- **Con trỏ void** là một loại con trỏ đặc biệt mà có thể trỏ đến bất kỳ kiểu dữ liệu nào.
- Cú pháp:

  void *pointerVariable;

- Ví dụ:

```
void *p;
p = &a; // p trỏ đến biến nguyên a
p = &f; //p trỏ đến biến thực f
```

6. Con trỏ void (*void pointers*)

- Kiểu dữ liệu khi khai báo biến con trỏ chính là kiểu dữ liệu của ô nhớ mà con trỏ có thể trỏ đến.
- Địa chỉ đặt vào biến con trỏ phải cùng kiểu với kiểu của con trỏ

Ví dụ:

```
int a; float f;
int *pa; float *pf;
pa = &a; pf = &f;// hợp lệ
pa = &f; pf = &a; //không hợp lệ
```

6. Con trỏ void (*void pointers*)

- Tuy nhiên, ta cũng có thể ép kiểu con trỏ về đúng kiểu tương ứng khi dùng trong các biểu thức.

Ví dụ:

- Nếu p đang trỏ đến biến nguyên a, để tăng giá trị của biến a lên 10 ta phải dùng lệnh sau:

```
(int*)*p + 10;
```

- Nếu p đang trỏ đến biến thực f, để tăng giá trị của biến f lên 10 ta phải dủng lệnh sau:

```
(float*)*p + 10;
```

7. Con trỏ null (*Null pointers*)

- Một con trỏ hiện hành không trỏ đến một địa chỉ bộ nhớ hợp lệ thì được gán giá trị NULL
- NULL được định nghĩa trong <cstdlib>

Ví dụ:

```
#include <iostream.h>
void main()
{ int *p;
  cout <<"Gia tri con tro p tro den la: "<< *p; }
```

- Kết quả của chương trình trên là:

NULL POINTER ASSIGNMENT

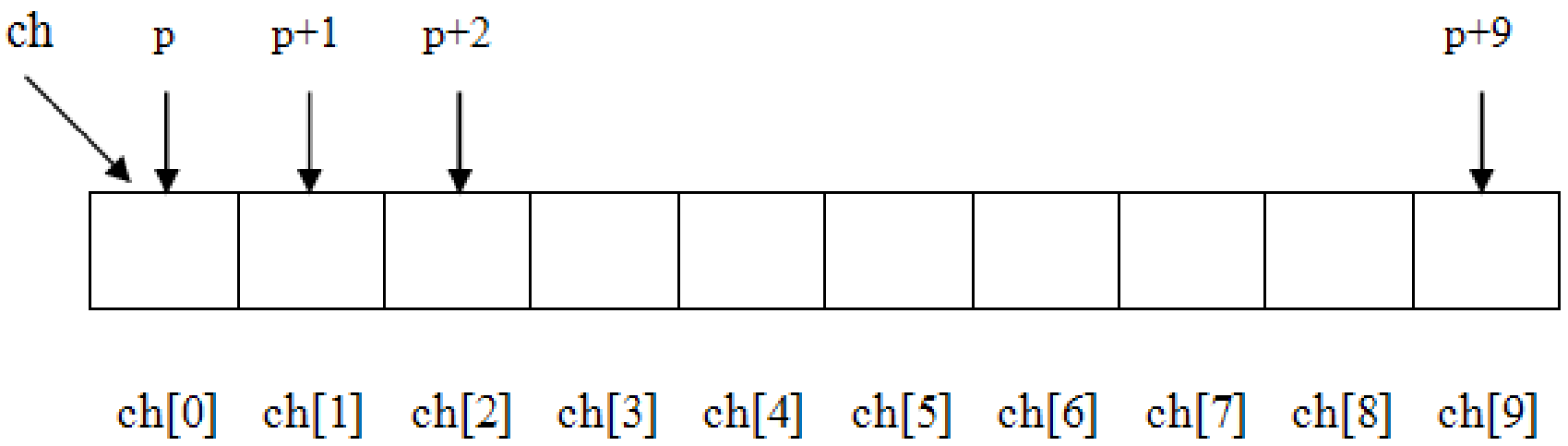
8. Con trỏ và mảng

- Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ:
  - Những phần tử của mảng được xác định bằng chỉ số trong mảng và cũng có thể được xác định qua biến con trỏ.
  - Tên của một mảng tương đương với địa chỉ phần tử đầu tiên của nó, tương tự một con trỏ tương đương với địa chỉ của phần tử đầu tiên mà nó trỏ tới.

8. Con trỏ và mảng

Ví dụ:

```
char ch[10], *p;
p = ch;
```



8. Con trỏ và mảng

- p được gán địa chỉ của phần tử đầu tiên của mảng ch.

```
p = ch;
```

- Để tham chiếu phần tử thứ 3 trong mảng ch, ta dùng một trong 2 cách sau:
  - ch[2]
  - *(p+2).

8. Con trỏ và mảng

- **Truy cập các phần tử mảng bằng con trỏ**

Kiểu mảng	Kiểu con trỏ
&<Tên mảng>[0]	<Tên con trỏ >
&<Tên mảng> [<Vị trí>]	<Tên con trỏ> + <Vị trí>
<Tên mảng>[<Vị trí>]	*(< Tên con trỏ > + <Vị trí>)

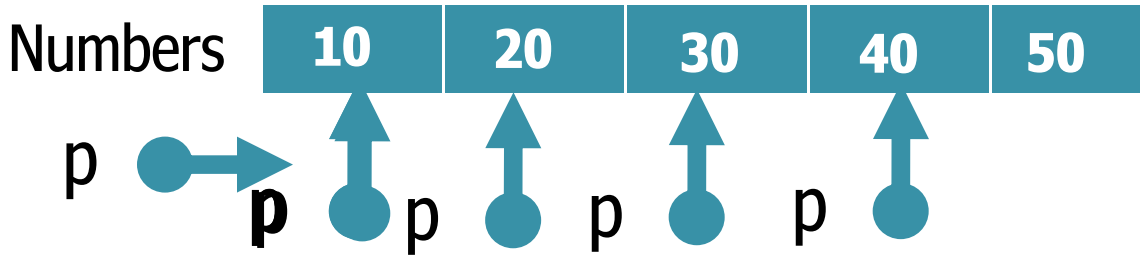
8. Con trỏ và mảng

Ví dụ:

```
#include <iostream.h>
#include <conio.h>
void main ()
{
   int numbers[5], * p;
   p = numbers;  *p = 10;
   p++;  *p = 20;
   p = &numbers[2];  *p = 30;
   p = numbers + 3;  *p = 40;
   p = numbers;  *(p+4) = 50;
   for (int n=0; n<5; n++)
   cout << numbers[n] << ", ";
}
```

8. Con trỏ và mảng

```
int  Numbers[5];
int  *p;
p= Numbers;
*p = 10;
p++;    *p = 20;
p = &numbers[2];  *p = 30;
p = numbers + 3;  *p = 40;
p = numbers;      *(p+4) = 50;
```



8. Con trỏ và mảng

Ví dụ: Xuất mảng sử dụng con trỏ

```
#include <iostream>
void main()
{
   int a[] = {0,1,2,3,4,5,6,7,8,9};
   int *p;
   p = a;
   for(int i=0 ; i<10 ; i++)
   {
        *(p+i) *= 10; //tuong duong a[i] = a[i]*10
        cout << "a[" << i << "] = " << *(p+i) << "\n";
   }
}
```

9. Mảng con trỏ

- Mỗi biến con trỏ là một biến đơn. Ta có thể tạo mảng của các con trỏ với mỗi phần tử của mảng là một con trỏ.
- **Cú pháp**:

  type *pointerArray[elements];

  - ***type***: kiểu dữ liệu mà các con trỏ phần tử trỏ đến.
  - ***pointerArray***: tên mảng con trỏ.
  - ***elements***: số phần tử của mảng con trỏ.

9. Mảng con trỏ

Ví dụ:

```
int *p[5];
int a=6;
p[0] = &a;
p[2] = p[0];
int b;
b = *p[0];
```