Chương 01.8: Con trỏ

I. Địa chỉ và con trỏ

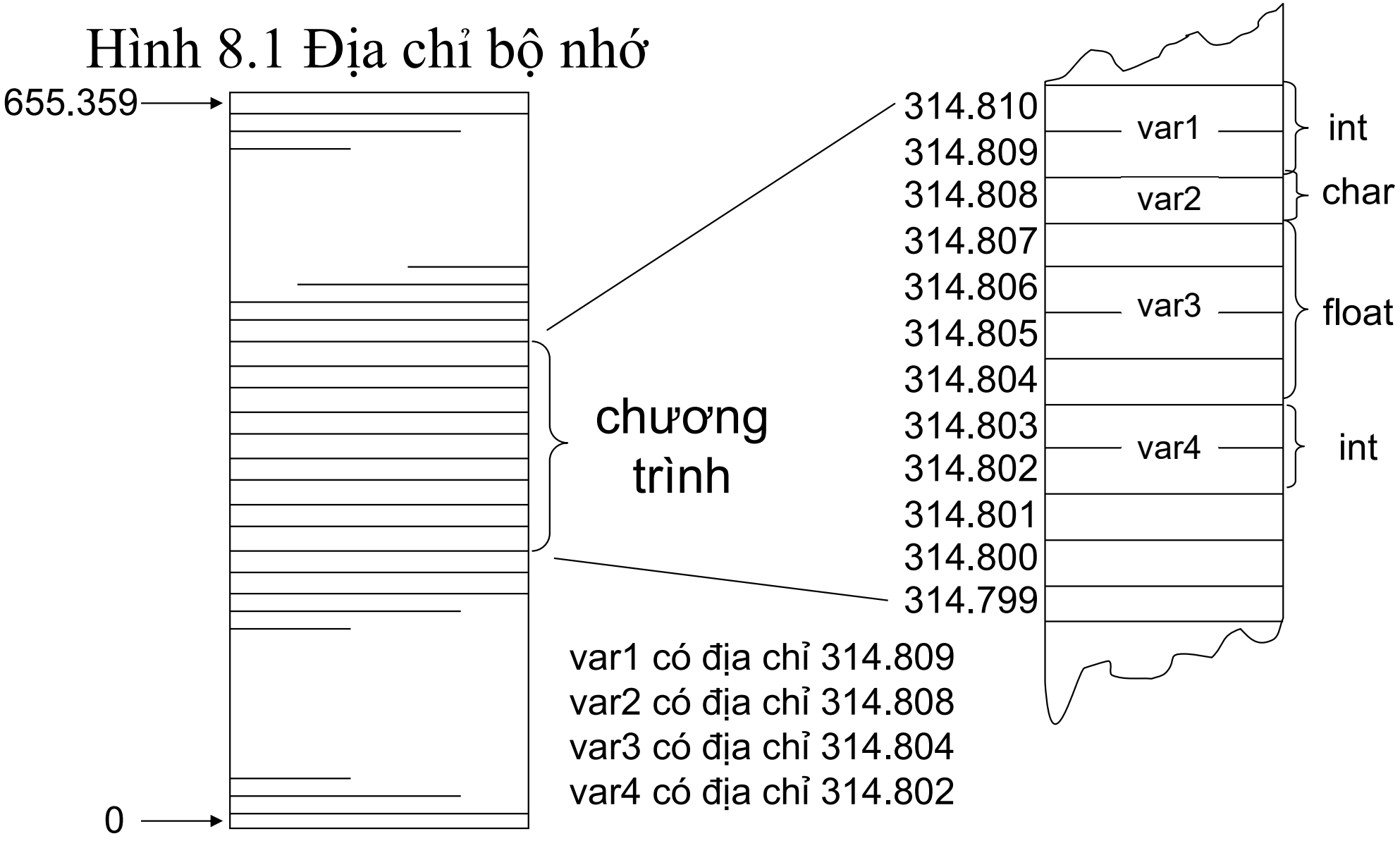
1. Địa chỉ (hằng con trỏ)
2. Toán tử địa chỉ &
3. Khai báo biến con trỏ
4. Truy nhập biến qua con trỏ
5. Con trỏ **void** và con trỏ NULL
6. Các phép toán trên con trỏ
7. Con trỏ trỏ tới con trỏ

1. Địa chỉ (hằng con trỏ)

- Mỗi byte trong bộ nhớ máy tính có một địa chỉ. Các địa chỉ này là các số bắt đầu từ 0 trở đi. Ví dụ có 1 MB bộ nhớ thì địa chỉ thấp nhất là 0 và địa chỉ cao nhất là 1.048.575.
- Bất kỳ chương trình nào khi được nạp vào bộ nhớ đều chiếm một khoảng địa chỉ. Điều đó có nghĩa là mọi biến và mọi hàm trong chương trình đều bắt đầu tại một địa chỉ cụ thể. Hình 8.1 cho thấy các địa chỉ bộ nhớ.

1. Địa chỉ (hằng con trỏ) *tiếp*

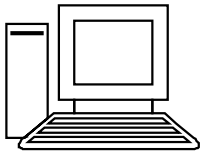
Hình 8.1 Địa chỉ bộ nhớ



var1 có địa chỉ 314.809
var2 có địa chỉ 314.808
var3 có địa chỉ 314.804
var4 có địa chỉ 314.802

2. Toán tử địa chỉ &

- Toán tử địa chỉ ký hiệu là &, được dùng để lấy địa chỉ của một biến. Toán tử & phải đặt trước tên biến muốn lấy địa chỉ. Ví dụ: Chương trình sau sẽ đưa ra địa chỉ của 3 biến nguyên a, b, c.



```
#include<iostream.h>
void main()
{
  int a,b,c;
  cout<<"Dia chi cua a: "<<&a<<'\n';
  cout<<"Dia chi cua b: "<<&b<<'\n';
  cout<<"Dia chi cua c: "<<&c<<'\n';
}
```

3. Khai báo biến con trỏ

- Vì địa chỉ bộ nhớ là số nên nó cũng có thể lưu trữ trong một biến giống như giá trị của các kiểu int, char và float. Một biến mà chứa giá trị địa chỉ gọi là biến con trỏ hay gọi tắt là con trỏ. Nếu một con trỏ chứa địa chỉ của một biến thì ta nói rằng con trỏ trỏ tới biến đó.
- Để khai báo các biến con trỏ ta dùng cú pháp sau:

*Kiểu *Tên_biến_con_trỏ;*

trong đó *Kiểu* là kiểu dữ liệu của đối tượng mà biến con trỏ sẽ trỏ tới. Dấu * có nghĩa là trỏ tới. Nên để dấu * bên cạnh tên kiểu để nhấn mạnh rằng nó là một phần của kiểu chứ không phải của tên biến con trỏ.

3. Khai báo biến con trỏ (tiếp)

- Ví dụ:

  int *ptr;

  Lệnh này khai báo một biến con trỏ có tên là ptr trỏ tới các số nguyên int. Nói cách khác con trỏ ptr có thể chứa địa chỉ của các biến nguyên.

- Để khai báo nhiều biến con trỏ cùng trỏ tới một kiểu dữ liệu ta viết:

  *Kiểu *Biến1, *Biến2, *Biến3,...;*

  Mặc dù dấu * để cạnh tên biến con trỏ nhưng vẫn nên hiểu nó là một phần của kiểu.

  Ví dụ: int *p, *q;

3. Khai báo biến con trỏ (tiếp)

- Khi khai báo một biến con trỏ thì biến con trỏ này sẽ chứa một giá trị vô nghĩa (trừ khi được khởi tạo). Giá trị vô nghĩa này có thể là địa chỉ của một ô nhớ nào đó nằm trong phần chương trình của ta hoặc hệ điều hành. Điều này sẽ rất nguy hiểm nếu ta đưa giá trị vào ô nhớ do con trỏ này trỏ tới. Bởi vậy, trước khi sử dụng một con trỏ ta phải đưa địa chỉ vào nó.
- Con trỏ trỏ tới kiểu nào thì chỉ chứa được địa chỉ của các biến kiểu đó. Không thể gán địa chỉ của biến float tới một con trỏ trỏ tới int.

4. Truy nhập biến qua con trỏ

- Một câu hỏi đặt ra là nếu không biết tên một biến mà chỉ biết địa chỉ của nó thì có truy nhập được vào biến đó không? Câu trả lời là có. Con trỏ chứa địa chỉ của một biến nên ta có thể truy nhập biến qua con trỏ.
- Để truy nhập tới biến do con trỏ ptr trỏ tới ta dùng toán tử truy nhập gián tiếp * đặt trước tên biến con trỏ: *ptr. *ptr tương đương với tên của biến, chỗ nào dùng được tên biến thì chỗ đó dùng được *ptr.

4. Truy nhập biến qua con trỏ

- Toán tử truy nhập gián tiếp cũng ký hiệu là * nhưng có nghĩa là *giá trị của biến được trỏ tới bởi biến con trỏ nằm bên phải nó*, khác với dấu * khi khai báo biến con trỏ có nghĩa là *trỏ tới*.
- Ví dụ:

```
int v;      //Khai báo biến có kiểu int
int* p;     //Khai báo biến con trỏ p trỏ tới int
p = &v;     //Gán địa chỉ của biến v cho con trỏ p
v = 3;      //Gán 3 vào v
*p = 3;     //Gán 3 vào v gián tiếp qua con trỏ p
```

5. Con trỏ trỏ tới **void** và con trỏ NULL

- Ta biết rằng con trỏ trỏ tới kiểu nào thì chỉ chứa được địa chỉ của các biến kiểu đó. Tuy nhiên trong C++ còn có một loại con trỏ đa năng có thể trỏ tới bất kỳ kiểu dữ liệu nào. Con trỏ đó gọi là con trỏ trỏ tới void. Khai báo con trỏ trỏ tới void như sau:

```
void *ptr;
```

- Con trỏ NULL là con trỏ không trỏ tới bất cứ cái gì, nó chứa giá trị rỗng (bằng 0). Để có con trỏ rỗng ta gán giá trị 0 vào biến con trỏ. Trong C++ có một tên hằng rỗng là NULL được khai báo trong iostream.h, ta có thể sử dụng tên hằng này để tạo con trỏ rỗng.

```
int* ptr=NULL;
```

5. Con trỏ trỏ tới **void** và con trỏ NULL (tiếp)

✧ **Ví dụ:**

```
int ivar;
float fvar;
int* iptr;
float* fptr;
void* vptr;
iptr = &ivar;
//iptr = &fvar;  //lỗi vì gán float*  tới  int*
fptr = &fvar;
//fptr = &ivar;  //lỗi vì gán int*  tới  float*
vptr = &ivar;   //được vì gán int*  tới  void*
vptr = &fvar;   //được vì gán float*  tới  void*
```

6. Các phép toán trên con trỏ

✧ Các phép toán số học:

- Chỉ có 4 phép toán dùng được với con trỏ là +, -, ++, --.
- Khi cộng hoặc trừ biến con trỏ với một số thì số đó phải nguyên.
- Các phép toán số học tác động trên con trỏ khác với bình thường. Cụ thể là khi tăng biến con trỏ lên 1 đơn vị thì địa chỉ chứa trong biến con trỏ không tăng lên một mà tăng lên một lượng bằng kích thước kiểu dữ liệu con trỏ trỏ tới (thường là 2 với kiểu int, 4 với kiểu float và 8 với kiểu double).

6. Các phép toán trên con trỏ (tiếp)

- Ví dụ: giả sử p là con trỏ int chứa địa chỉ 200, sau khi lệnh

  ++p;

  được thực hiện thì p sẽ có giá trị là 202. Nếu p là con trỏ float thì sau lệnh trên p sẽ có giá trị là 204.

✧ **Các phép toán so sánh:** có thể so sánh hai biến con trỏ bằng các phép toán so sánh. Tuy nhiên việc so sánh này chỉ có ý nghĩa trong hai trường hợp sau:

- So sánh hai con trỏ để xem chúng có bằng con trỏ NULL không.

6. Các phép toán trên con trỏ (tiếp)

- So sánh hai con trỏ khi chúng cùng liên quan tới một đối tượng, chẳng hạn là cùng trỏ tới một biến.

✧ **Phép gán:** Có thể gán một biến con trỏ cho một biến con trỏ có cùng kiểu trỏ tới.

✧ **Lưu ý**: Khi dùng toán tử tăng hoặc giảm với biến do con trỏ trỏ tới thì phải chú ý về thứ tự thực hiện các phép toán. Ví dụ: nếu ta viết

```
*p++;
```

thì con trỏ sẽ tăng lên 1 chứ không phải biến do con trỏ trỏ tới tăng lên 1, bởi vì phép toán * và ++ cùng mức ưu tiên, được kết hợp từ phải qua trái. Muốn tăng biến do con trỏ trỏ tới ta phải viết:

```
(*p)++;
```

7. Con trỏ trỏ tới con trỏ

- Trong C++, một con trỏ có thể trỏ tới một con trỏ khác, tức là một con trỏ có thể chứa địa chỉ của một biến con trỏ khác.

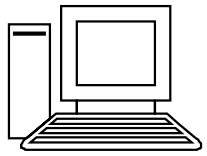
7. Con trỏ trỏ tới con trỏ (tiếp)

- Để khai báo một biến con trỏ trỏ tới một con trỏ ta dùng thêm dấu * nữa. Ví dụ:

  int **p; //p là con trỏ trỏ tới một con trỏ int

- Để truy nhập tới biến qua con trỏ trỏ tới con trỏ ta phải dùng hai lần toán tử truy nhập gián tiếp. Kiểu truy nhập này gọi là truy nhập gián tiếp bội (Multiple Indirection). Ví dụ:

```
char ch;
char* p;
char** mp;
ch='A';
p=&ch;
mp=&p;
cout<<"Ky tu nam trong bien ch la: "<<**mp;
```



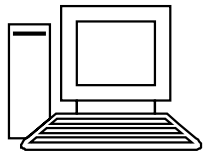
II. Con trỏ, mảng và xâu ký tự

1. Con trỏ và mảng
2. Con trỏ và xâu ký tự

1. Con trỏ và mảng

- Con trỏ được sử dụng để truy nhập vào các phần tử của mảng và làm đối số truyền vào hàm. Và khi mảng làm đối số truyền vào hàm thì con trỏ cũng rất hữu ích.
- Các phần tử của mảng có thể được truy nhập qua ký hiệu của mảng ([]) hoặc ký hiệu của con trỏ (*). Ví dụ:

```
int a[5]={31,54,77,52,93};
int i;
//Dua ra bang ky hieu cua mang
for(i=0;i<5;i++) cout<<a[i]<<'\n';
//Dua ra bang ky hieu cua con tro
for(i=0;i<5;i++) cout<<*(a+i)<<'\n';
```

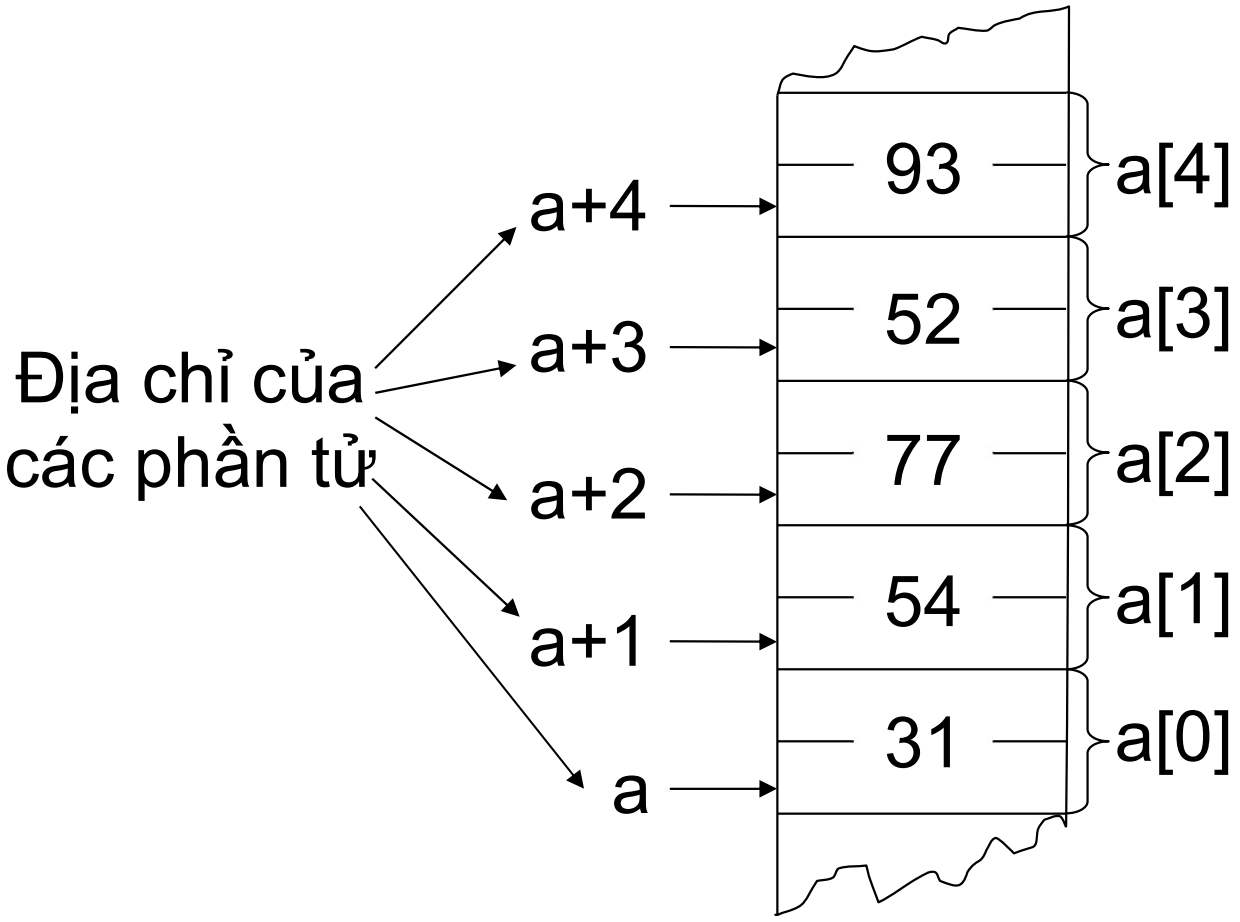


1. Con trỏ và mảng (tiếp)

- Biểu thức *(a+i) tương đương với a[i]. Ví dụ, với i=2 thì *(a+2) là phần tử thứ 3 (có giá trị là 77).
- Tại sao *(a+2) lại là phần tử thứ 3? Như ta đã biết, tên biến mảng chính là địa chỉ của phần tử đầu tiên của biến mảng. Khi ta viết (a+2) thì trình biên dịch sẽ thực hiện cộng địa chỉ với 2. Khi cộng địa chỉ với 2 trình biên dịch lấy kích thước kiểu dữ liệu của mảng nhân với 2 rồi mới cộng vào địa chỉ. Kết quả (a+2) cho ta địa chỉ của phần tử thứ 3. Để truy nhập tới phần tử thứ 3 khi biết địa chỉ phải sử dụng toán tử truy nhập gián tiếp *(a+2).

1. Con trỏ và mảng (tiếp)

✧ Địa chỉ của các phần tử mảng



1. Con trỏ và mảng (tiếp)

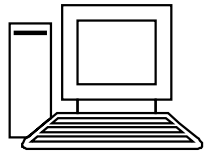
- **Hằng con trỏ và biến con trỏ:** Tên biến mảng là một địa chỉ cụ thể mà hệ thống đã chọn để đặt mảng. Địa chỉ này không thể thay đổi và nó được duy trì khi biến mảng còn tồn tại. Người ta gọi các địa chỉ không thay đổi được là các hằng con trỏ. Vì tên biến mảng a ở ví dụ trên là hằng nên ta không thể viết a++ hay a+=2.

  Một địa chỉ thì không thể thay đổi nhưng biến con trỏ chứa địa chỉ thì có thể thay đổi.

1. Con trỏ và mảng (tiếp)

- **Hằng con trỏ và biến con trỏ: (tiếp)**

  Ví dụ sau dùng biến con trỏ để đưa ra các phần tử của mảng:



```
 int a[5]={31,54,77,52,93};
int i;
int *p=a;  //p tro toi phan tu dau tien cua mang a
//Dua ra bang bien con tro
cout<<"Dua ra bang bien con tro: "<<'\n';
for(i=0;i<5;i++) cout<<*p++<<' ';
```

2. Con trỏ và xâu ký tự

- Như ta đã biết, xâu ký tự thực chất là mảng ký tự. Bởi vậy ta có thể dùng ký hiệu con trỏ để truy nhập vào các ký tự của xâu giống như truy nhập vào các phần tử của mảng. Ví dụ:

  char s[6]=”DHNNI”;

  cout<<*(s+1);//Dua ra ky tu thu 2 la H

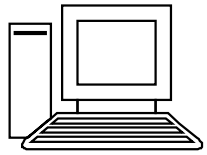
- **Con trỏ trỏ tới hằng xâu ký tự:** Khi khai báo và khởi tạo biến xâu ký tự ta có thể khai báo như một mảng ký tự hoặc khai báo như một con trỏ trỏ tới kiểu ký tự. Ví dụ:

  char s1[] = ”Khai bao nhu mot mang”;

  //char* s1 = ”Khai bao nhu con con tro”;

2. Con trỏ và xâu ký tự (tiếp)

Sau khai báo trên ta sẽ được hai biến xâu ký tự s1 và s2. Tuy nhiên hai biến xâu này có một sự khác nhau: s1 là một địa chỉ, một hằng con trỏ, s2 là một biến con trỏ; s2 có thể thay đổi còn s1 không thể thay đổi. Ví dụ:



```
char s1[]="Khai bao nhu mot mang";
char* s2 ="Khai bao nhu mot con tro";
cout<<s1<<'\n';
cout<<s2<<'\n';
//s1++;                 //Bao loi, s1 la hang con tro
s2++;                   //Duoc
cout<<s2;               //Chi hien: hai bao nhu mot con tro
```

Chú ý: Khi thay đổi s2 thì ký tự đầu tiên của xâu sẽ thay đổi. Ở ví dụ trên, sau khi tăng s2 lên 1 thì ký tự đầu tiên của xâu là h.

2. Con trỏ và xâu ký tự (tiếp)

✧ Mảng con trỏ trỏ tới các hằng xâu ký tự:

- Giống như mảng các biến kiểu int hoặc float, ta cũng có mảng con trỏ. Mảng con trỏ hay dùng nhất là mảng con trỏ trỏ tới các hằng xâu ký tự.
- Ta xét hai cách khai báo sau đây:

```
//Dùng mảng hai chiều
char days[7][10]={"Sunday","Monday","Tuesday","Wednesday",
                  "Thursday","Friday","Saturday"};
//Dùng con trỏ
char* days[7]={"Sunday","Monday","Tuesday","Wednesday",
               "Thursday","Friday","Saturday"};
```

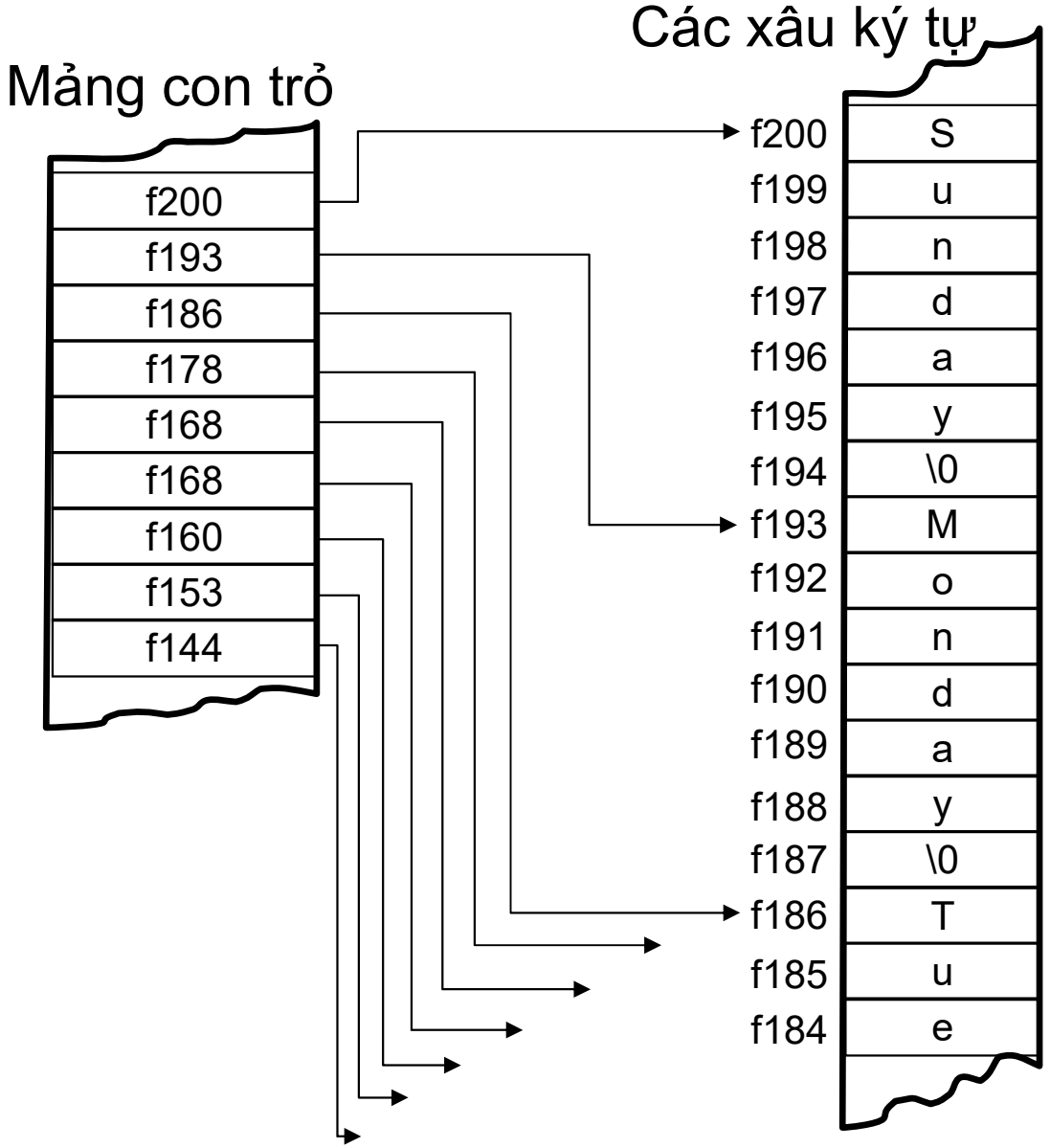
2. Con trỏ và xâu ký tự (tiếp)

✧ Mảng con trỏ trỏ tới các hằng xâu ký tự: (tiếp)

- Nếu khai báo theo mảng hai chiều thì các mảng con chứa các xâu ký tự phải có kích thước bằng nhau (10). Do đó, với những xâu có số ký tự nhỏ hơn 10 sẽ gây lãng phí bộ nhớ.
- Nếu khai báo theo con trỏ thì trình biên dịch C++ sẽ để các xâu ký tự liên tiếp nhau trong bộ nhớ và dùng một mảng con trỏ để trỏ tới các xâu này (Hình trang sau cho thấy các xâu ký tự trong bộ nhớ). Một xâu ký tự là một mảng kiểu char, do đó một mảng con trỏ trỏ tới xâu ký tự thực chất là một mảng con trỏ trỏ tới char. Đây chính là lý do tại sao ta khai báo là char*

2. Con trỏ và xâu ký tự (tiếp)

Địa chỉ của ký tự đầu tiên chính là địa chỉ của xâu. Các địa chỉ này được lưu trữ trong mảng con trỏ.

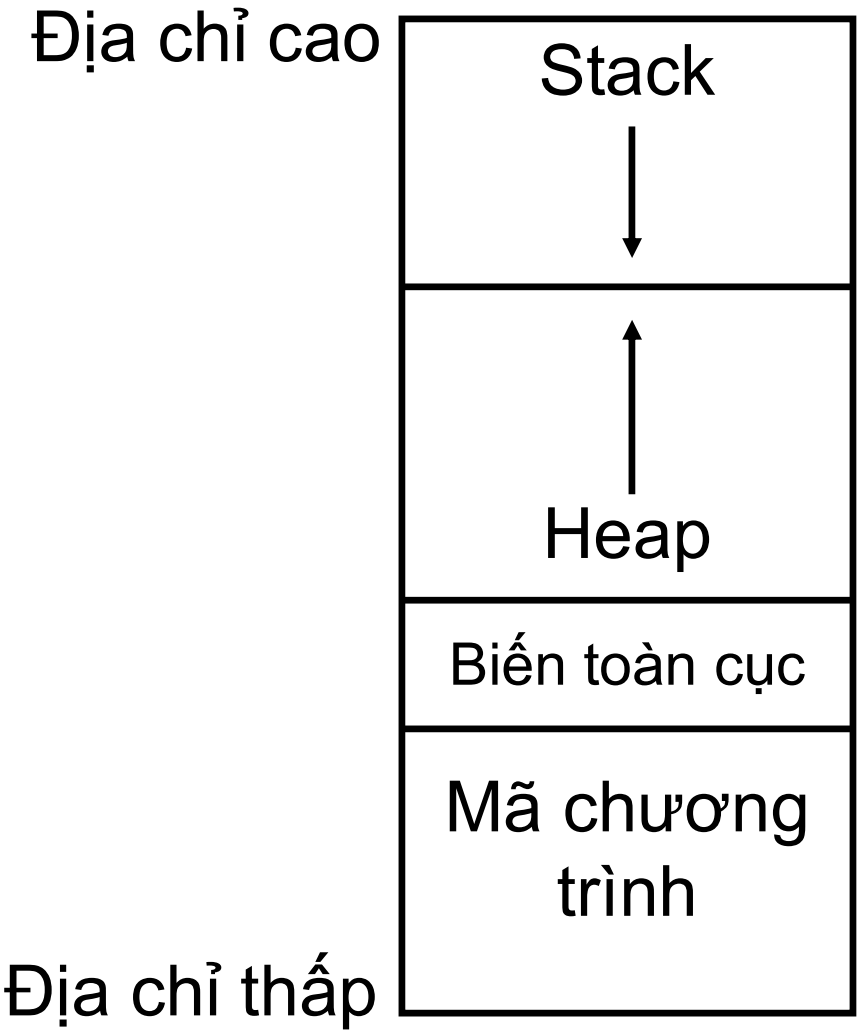


III. Quản lý bộ nhớ với new và delete

1. Cách sử dụng bộ nhớ của một chương trình C++
2. Hạn chế của mảng
3. Toán tử new và delete
4. Khởi tạo ô nhớ được cấp phát động
5. Mảng động

1. Cách sử dụng bộ nhớ của một chương trình C++

- Một chương trình C++ khi chạy sẽ chiếm một vùng nhớ trong bộ nhớ. Vùng nhớ này được chia thành 3 phần: phần chứa mã chương trình, phần chứa các biến tĩnh và biến ngoài (gọi là Heap), phần chứa các biến tự động (gọi là Stack). Stack mở rộng từ địa chỉ cao xuống địa chỉ thấp, Heap mở rộng từ địa chỉ thấp lên địa chỉ cao.



2. Hạn chế của việc lưu trữ bằng mảng

- Mảng rất hay được sử dụng khi cần lưu trữ một số lượng lớn các biến hay đối tượng. Tuy nhiên tại thời điểm viết chương trình ta phải xác định kích thước của mảng chứ không đợi được đến khi chương trình thực hiện. Đoạn chương trình sau sẽ sinh ra lỗi:

```
cin>>size;          //Lấy kích thước mảng
int  a[size];       //Lỗi, kích thước mảng phải là hằng
```

- Trong nhiều trường hợp, tại thời điểm viết chương trình ta không biết được là cần bao nhiêu bộ nhớ. Nếu dự trù nhiều mà không dùng hết thì lãng phí bộ nhớ, nếu dự trù ít mà cần lưu trữ nhiều thì không có chỗ chứa. Vấn đề này được khắc phục bằng cơ chế cấp phát động bộ nhớ nhờ toán tử new và delete.

3. Toán tử new và delete

- C++ có 2 toán tử thực hiện chức năng cấp phát và giải phóng bộ nhớ. Cú pháp như sau:

  Con_trỏ = new Kiểu_dl_của_biến; //Cấp phát

  delete Con_trỏ; //Giải phóng bộ nhớ

  Ví dụ: int *p = new int; delete p;

  trong đó Biến con trỏ phải được khai báo trỏ đến kiểu dữ liệu của biến.

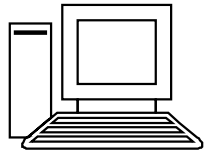
- Toán tử new sẽ cấp phát một ô nhớ trong phần nhớ Heap, trong khi chương trình đang chạy, đủ để chứa một giá trị có kiểu Kiểu_dl_của_biến và trả về một con trỏ trỏ tới nó.

3. Toán tử new và delete (tiếp)

- Toán tử delete sẽ giải phóng vùng nhớ được trỏ tới bởi biến con trỏ. Chỉ nên dùng delete để giải phóng vùng nhớ được cấp phát bởi new. Nếu dùng delete để giải phóng các vùng nhớ không được cấp phát bởi new sẽ gây ra nhiều nguy hiểm.
- Vì kích thước phần Heap có giới hạn nên có thể sẽ hết. Nếu phần nhớ Heap đã hết mà ta vẫn cấp phát thì new sẽ trả về con trỏ rỗng. Bởi vậy, luôn luôn phải kiểm tra con trỏ được trả về bởi new trước khi dùng nó.

3. Toán tử new và delete (tiếp)

✧Ví dụ về sử dụng new và delete:

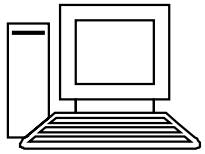


```
//Khai bao su dung thu vien chuong trinh
#include<iostream.h>
void main()
{
  int* p;
  p=new int;          //cap phat bo nho chua kieu int
  if(!p)
    {
      cout<<"Cap phat bo nho bi loi";
      return 1;
    }
  *p=100;             //Gan 100 vao o nho vua duoc cap
  cout<<*p;           //Hien thi noi dung cua o nho vua duoc cap
  delete p;           //Giai phong o nho vua duoc cap
}
```

4. Khởi tạo ô nhớ được cấp phát động

- Ta có thể khởi tạo giá trị cho các ô nhớ được cấp phát động bởi new. Giá trị khởi tạo phải đặt trong ngoặc đơn sau tên kiểu dữ liệu. Ví dụ:

```
int* p;
p = new int(1200);
cout<<*p;
```



5. Mảng động

- Với cơ chế cấp phát động bộ nhớ ta có thể cấp phát bộ nhớ cho cả một biến mảng. Điều này cho phép xác định số phần tử của mảng trong khi chạy chương trình. Cú pháp cấp phát động cho mảng một chiều như sau:

  Con_trỏ = new Kiểu_của_mảng[size];

  trong đó size là số phần tử của mảng, size có thể là hằng, biến hoặc biểu thức.
- Để giải phóng vùng nhớ cấp phát cho mảng ta dùng toán tử delete:

  delete [] Con_trỏ;

5. Mảng động (tiếp)

✧ Ví dụ về mảng động:

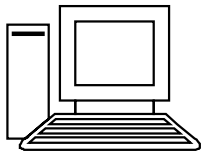
```
//Khai bao su dung thu vien chuong trinh
#include<iostream.h>
int main()
{
  int* p;
  int n,i;

  cout<<"Nhap vao so phan tu cua mang: ";cin>>n;
  p=new int[n];                    //Cap phat bo nho cho mang n phan tu nguyen
  if(!p)
    {
      cout<<"Cap phat bo nho bi loi";
      return 1;
    }
```

//Tiếp trang sau

5. Mảng động (tiếp)

✧ Ví dụ về mảng động: (tiếp)



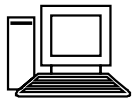
```
 //Nhap cac gia tri vao mang
  cout<<"Nhap vao mang so nguyen:\n";
  for(i=0;i<n;++i)
    {
     cout<<"Nhap vao so thu "<<i+1<<": ";cin>>p[i];
    }
  //Dua cac so nhap vao ra man hinh
  cout<<"Cac so da nhap la:\n";
for(i=0;i<n;++i) cout<<p[i]<<'  ';
  delete [] p;          //Giai phong vung nho cap phat cho mang
}
```

Ví dụ

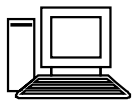
1. Cho tệp văn bản ‘daysonguyen.txt’chứa dãy số nguyên có n phần tử. Đọc dãy số nguyên vào mảng động. Sắp xếp dãy số tăng dần theo giải thuật sắp xếp chọn.
2. Nhập vào 1 số nguyên dương. Cho biết số nguyên đó có phải là số nguyên tố không. Y/c sử dụng tất cả là biến động.
3. Cho dãy số nguyên có n phần tử. Tìm vị trí phần tử lớn nhất. Y/c sử dụng mảng động.

Bài tập chương 8

- Bài 1. Viết chương trình nhập vào một dãy n số nguyên, lưu dãy số này trong một danh sách liên kết đơn P. Hãy tạo một danh sách liên kết đơn Q là đảo ngược của P.

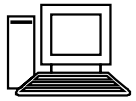


- Bài 2. Viết chương trình nhập vào một dãy n số nguyên, lưu dãy số này trong một danh sách liên kết đơn P. Hãy sắp xếp dãy số theo chiều không giảm sử dụng phương pháp sắp xếp chọn.



Bài tập chương 8

- Bài 3. Viết chương trình nhập vào một dãy n số nguyên, lưu dãy số này trong một mảng động. Sắp xếp dãy số tăng dần theo phương pháp chọn. Đưa dãy số đã sắp xếp ra màn hình.



- Bài 4. Cho dãy số nguyên a_1, a_2, a_3, …, a_n. Tạo hai dãy số, một dãy chứa các số chẵn và một dãy chứa các số lẻ. Yêu cầu trong chương trình chỉ sử dụng mảng động.