ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA CÔNG NGHỆ PHẦN MỀM

C++ NÂNG CAO

THÀNH PHỐ HỒ CHÍ MINH - 2010

PHẦN 1 : TURBO C NÂNG CAO VÀ C++

CHƯƠNG 1 : BIẾN CON TRỎ

§1. KHÁI NIỆM CHUNG

Một con trỏ là một biến chứa địa chỉ của một biến khác. Nếu một biến chứa địa chỉ của một biến khác tthì ta nói biến thứ nhất trỏ đến biến thứ hai .

Cũng như mọi biến khác, biến con trỏ cũng phải được khai báo trước khi dùng. Dạng tổng quát để khai báo một biến con trỏ là :

type *<tên biến>

Trong đó : type là bất kì kiểu dữ liệu cơ bản thích hợp nào được chấp nhận trong C và <tên biến> là tên của một biến con trỏ. Kiểu dữ liệu cơ bản xác định kiểu của những biến mà con trỏ có thể chỉ đến. Ví dụ khai báo biến con trỏ chỉ đến các biến nguyên và biến kiểu kí tự:

```
char *p;
int  *x,*y;
```

Con trỏ có một trị đặc biệt gọi là NULL. Trị này có nghĩa là con trỏ chưa trỏ tới một địa chỉ hợp lệ nào cả. Để dùng được trị này chúng ta phải dùng #include <stdio.h> đầu chương trình

§2. CÁC PHÉP TOÁN VỀ CON TRỎ

C có hai phép toán đặc biệt đối với con trỏ : * và & . Phép toán & là phép toán trả về địa chỉ trong bộ nhớ của biến sau nó. Ví dụ :

```
p = &a;
```

sẽ đặt vào biến p địa chỉ trong bộ nhớ của biến a. Địa chỉ này không có liên quan gì đến trị số của biến a. Nói cách khác địa chỉ của biến a không liên quan gì đến nội dung của biến a.

Phép toán * là phép toán trả về trị của biến đặt tại địa chỉ được mô tả bởi biến đi sau nó. Ví dụ nếu biến a chứa địa chỉ của biến b thì

```
p = *a
```

sẽ đặt trị số của biến b vào biến p

Chương trình 1-1 : Lập chương trình in số 100 lên màn hình

```
main()
  {
    int *p,a,b;
    clrscr();
    a=100;
    p=&a;
    b=*p;
    printf("%d",b);
    getch();
  }
```

§3. TẦM QUAN TRỌNG CỦA DỮ LIỆU KHI KHAI BÁO CON TRỎ

Cần phải bảo đảm là con trỏ luôn luôn trỏ đến một kiểu dữ liệu phù hợp. Ví dụ khi khai báo con trỏ kiểu int , trình biên dịch sẽ hiểu là con trỏ bao giờ cũng chỉ đến một biến có độ dài là 2 byte .
Ta xét một chương trình như sau

Chương trình 1-2

```
main()
  {
    float x=10.1,y;
    int *p;

    clrscr();
    p=&x;
    y=*p;
    printf("%f",y);
    getch();
  }
```

Chương trình này nhằm gán trị của x cho biến y và in ra trị đó. Khi biên dịch chương trình không báo lỗi mà chỉ nhắc nhở :

Suspencious pointer conversion in function main

Tuy nhiên chương trình không gán trị x cho y được. Lí do là ta khai báo một con trỏ int và cho nó trỏ tới biến float x. Như vậy trình biên dịch sẽ chỉ chuyển 2 byte thông tin cho y chứ không phải 4 byte để tạo ra một số dạng float .

§4. CÁC BIỂU THỨC CON TRỎ

1. Các phép gán con trỏ : Cũng giống như bất kì một biến nào khác , ta có thể dùng một con trỏ ở về phải của một phép gán để gán trị của một con trỏ cho một con trỏ khác. Ví dụ ta viết

Chương trình 1-3 :

```
main()
   {
      int x;
      int *p1,*p2;
      clrscr();
      p1 = &x;
      p2 = p1;
      printf(" %p",p2);
      getch();
   }
```

Chương trình này hiện lên địa chỉ của biến x ở dạng hex bằng cách dùng một mã định dạng khác của hàm printf() . %p mô tả rằng sẽ hiện lên một trị chứa trong một biến con trỏ theo dạng reg:xxxx với reg là tên của một trong các thanh ghi segment của CPU còn xxxx là địa chỉ offset tính từ đầu segment .

2. Các phép toán số học của con trỏ : Trong C , ta chỉ có thể dùng hai phép toán số học tác động lên con trỏ là phép + và - . Để hiểu được cái gì sẽ xảy ra khi thực hiện một phép toán số học lên con trỏ ta giả sử p1 là một con trỏ chỉ đến một số nguyên có địa chỉ là 2000 . Sau khi thực hiện biểu thức

p1++ ;

con trỏ sẽ chỉ đến số nguyên nằm ở địa chỉ 2002 vì mỗi khi tăng con trỏ lên 1 nó sẽ chỉ đến số nguyên kế tiếp mà mỗi số nguyên lại có độ dài 2 byte . Điều này cũng đúng khi giảm . Ví dụ :

p1-- ;

sẽ trỏ tới số nguyên ở địa chỉ 1998 . Như vậy mỗi khi con trỏ tăng lên 1 , nó sẽ chỉ đến dữ liệu kế tiếp tại địa chỉ nào đó tuỳ theo độ dài của kiểu dữ liệu. C còn cho phép cộng hay trừ một số nguyên với một con trỏ . Biểu thức :

p1 = p1 + 9;

sẽ làm cho con trỏ chỉ tới phần tử thứ 9 có kiểu là kiểu mà p1 trỏ tới và nằm sau phân tử hiện thời nó đang trỏ đến . Ngoài các phép toán trên , con trỏ không chấp nhận một phép toán nào khác .

3. So sánh các con trỏ : Chúng ta có thể so sánh 2 con trỏ trong một biểu thức quan hệ . Ví dụ cho hai p và q , phát biểu sau đây là hợp lệ :

```
if (p<q)
   printf("p tro den mot vi tri bo nho thap hon q\n");
```

Tuy nhiên cần nhớ rằng phép toán trên là so sánh hai địa chỉ chứa trong p và q chứ không phải nội dung của hai biến mà p và q trỏ tới .

4. Các ví dụ về việc dùng con trỏ :

Chương trình 1-4 : Phân tích chương trình sau :

```
main()
 {
  int i,j,*p;

  i=5;
  p=&i;
  j=*p;
  *p=j+2;
 }
```

Trong chương trình trên ta khai báo hai biến nguyên là i và j và một biến con trỏ p trỏ tới một số nguyên . Chương trình sẽ phân phối bộ nhớ cho 3 biến này ví dụ tại các địa chỉ 100 , 102 và 104 vì mỗi số nguyên dài 2 byte và con trỏ mặc nhiên cũng được mã hoá bằng 2 byte .

100		i
102		j
104		p

lệnh i=5 cho trị số của biến i là 5

100	5	i
102		j
104		p

lệnh p= &i làm cho con trỏ chỉ tới biến i nghĩa là con trỏ p chứa địa chỉ của biến i . Bây giờ p chỉ đến biến i .

100	5	i
102		j
104	100	p

lệnh j=*p đặt nội dung của biến do p chỉ tới (biến i) vào biến j nghĩa là gán 5 cho j

100	5	i
102	5	j
104	100	p

Một trong những vấn đề lí thú khi dùng con trỏ là xem nội dung bộ nhớ của máy tính . Chương trình sau đây cho phép ta vào địa chỉ bắt đầu của RAM mà ta muốn khảo sát và sau đó hiện lên nội dung mỗi byte ở dạng số hex . Trong chương trình có từ khoá far dùng để tham khảo đến các vị trí không nằm trong cùng một segment .

Chương trình 1-5 :

```
main()
  {
    unsigned long int start;
    char *p;
    int t;

    clrscr();
    printf("Nhap vao dia chi bat dau ma ban muon xem : ");
    scanf("%lu",&start);
    p = (char far *) start;
    for(t=0;;t++,p++)
      if(!(t%16))
          {
            printf("%2x\n",*p);
            getch();
          }
  }
```

Trong chương trình ta dùng định dạng %x trong hàm printf() để in ra số dạng hex . Dòng p = (char far *) start; dùng biến đổi số nhập vào thành một con trỏ .

§5. CON TRỎ VÀ MẢNG

Trong chương trước chúng ta đã thấy các ví dụ về mảng . Con trỏ thường được dùng khi xử lí mảng . Chúng ta xét chương trình sau :

Chương trình 1-6 :

```
main()
  {
    int a[10],*pa,x;
    a[0]=11;
    a[1]=22;
    a[2]=33;
    a[3]=44;
    clrscr();
    pa=&a[0];
    x=*pa;
    pa++;
    x=*pa;
```

```
  x=*pa+1;
  x=*(pa+1);
  x=*++pa;
  x=++*pa;
  x=*pa++;
 }
```

*int a[10] , *pa , x;* khai báo một bảng gồm 10 phần tử kiểu int , được liệt kê là a[0],a[1],..,a[9] , một con trỏ để chỉ đến một biến kiểu int và một biến kiểu int là x.
a[0] = 11. . .; từ a[4] đến a[9] chưa được khởi gán . Như vậy chúng sẽ chứa trị ngẫu nhiên đã có tại những vị trí bộ nhớ đã phân phối cho chúng .
pa=&a[0]; đặt vào pa địa chỉ của phần tử đầu tiên của mảng . Biểu thức này có thể viết đơn giản là pa = a ; vì tên của một mảng luôn luôn được trình biên dịch coi là địa chỉ của phần tử đầu tiên của mảng . Tên của mảng không có chỉ số kèm theo có thể được dùng trong chương trình như một hằng địa chỉ .
*x=*pa;* đặt nội dung của biến nguyên mà pa trỏ đến vào (tức là a[0]) vào x . Như vậy x = 11
pa++; pa được tăng lên 1 và bây giờ trỏ vào phần tử thứ 2 của mảng tức là chứa địa chỉ của phần tử a[1]
*x=*pa* ; pa trỏ đến phần tử a[1] nên x = 22
*x = *pa +1* ; x =23
*x = *(pa+1)* ; trước hết pa+1 được thực hiện , nghĩa là pa trỏ vào a[2] , sau đó nội dung của a[2] được gán cho x nên x= 33 .Tuy pa tham gia vào phép toán nhưng trị số của nó không thay đổi .
*x = *++pa*; ++ được thực hiện trước nên pa trỏ tới a[2] . Sau đó trị của a[2] được gán cho x nên x =33
*x= ++*pa*; *pa được thực hiện trước . Do pa chỉ đến a[2] nên *pa=33 và ++*pa=34 . Như vậy x = 34 và a[2]=34
*x=*pa++*; nội dung của pa (tức 34) được đặt vào x . Sau đó nó được tăng lên 1 nên chỉ vào a[3].

Chương trình 1-7:

```
main()
 {
  static int num[]={92,81,70,69,58};
  int dex;
  clrscr();
  for(dex=0;dex<5;dex++)
    printf("%d\n",num[dex]);
  getch();
 }
```

Chương trình 1-8 :

```
main()
 {
  static int num[]={92,81,70,69,58};
  int dex;
  clrscr();
  for(dex=0;dex<5;dex++)
    printf("%d\n",*(num+dex));
```

```
  getch();
 }
```

Hai chương trình chỉ khác nhau ở biểu thức : *(num+dex) . Cách viết này tương đương với num[dex] .Nói cách khác truy cập đến phần tử có chỉ số dex trong mảng num . Chúng ta hiểu *(num+dex) như sau : đầu tiên num là địa chỉ của phần tử đầu tiên của mảng num và ta muốn biết trị số của phần tử có chỉ số dex . Vì vậy num+dex sẽ là địa chỉ của phần tử thứ dex . *(num+dex) xác định nội dung của phần tử (num+dex) . Tóm lại :

(array+index) tương tự array(index)

Có hai cách truy cập mảng là :

theo kí hiệu mảng &array[index]
theo kí hiệu con trỏ array+index

Chương trình 1-9 : Tính nhiệt độ trung bình bằng cách dùng con trỏ

```
main()
 {
  float temp[40];
  float sum=0.0;
  int num,day=0;
  clrscr();
  do
    {
       printf("Cho nhiet do ngay thu %d: ",day+1);
       scanf("%f",temp+day);
    }
  while(*(temp+day++)>0);
  num = day-1;
  for(day=0;day<num;day++)
    sum+=*(temp+day);
  printf("Nhiet do trung binh la : %.3f",sum/num);
  getch();
 }
```

Trong ví dụ trên chúng ta đã dùng biểu thức (temp+day) để truy cập mảng . Tuy nhiên viết while((*temp++)>0) vì temp là hằng con trỏ chứ không phải biến con trỏ . Như vậy chỉ được phép thay đổi trị của biến con trỏ chứ không được thay đổi trị của hằng con trỏ . Chúng ta viết lại chương trình như sau :

Chương trình 1-10 :

```
main()
 {
  float temp[40];
  float sum=0.0;
  int num,day=0;
  float *p;

  clrscr();
  p=temp;
  do
    {
       printf("Cho nhiet do ngay thu %d: ",day+1);
```

```
        scanf("%f",p);
        day++;
     }
   while(*(p++)>0);
   p=temp;
   num=day-1;
   for(day=0;day<num;day++)
     sum+=*(p++);
   printf("Nhiet do trung binh la : %.3f",sum/num);
   getch();
  }
```

Trong chương trình này địa chỉ của temp được đưa vào biến con trỏ p . Sau đó ta tham khảo tới p giống như temp . Ta dùng p trỏ tới mảng và *p là nội dung của địa chỉ đó . Hơn nã do p là biến con trỏ nên ta có thể tăng nó bằng phát biểu p++.

§6. CON TRỎ VÀ CHUỖI

Rất nhiều hàm thư viện trong C làm việc với chuỗi theo con trỏ . Ví dụ hàm strchr() trả về con trỏ trỏ đến lần xuất hiện đầu tiên của một kí tự nào đó trong chuỗi Ví dụ : ptr = strchr(str,'x')
thì biến con trỏ ptr sẽ được gán địa chỉ của lần xuất hiện kí tự 'x' đầu tiên trong chuỗi str .
Sau đây là chương trình cho phép ta gõ vào một câu và một kí tự cần định vị trong câu . Chương trình sẽ cho ta :

- địa chỉ bắt đầu của chuỗi
- địa chỉ của kí tự cần định vị
- độ lệch so với điểm đầu chuỗi

Chương trình 1-11 :

```
#include<string.h>
main()
  {
   char ch,line[81],*ptr;
   clrscr();
   printf("Cho mot cau : ");
   gets(line);
   printf("Cho ki tu can tim : ");
   ch=getche();
   ptr=strchr(line,ch);
   printf("\nChuoi bat dau tai dia chi %u.\n",line);
   printf("Ki tu xuat hien lan dau tai %u.\n",ptr);
   printf("Do la vi tri %d",(ptr-line+1));
   getch();
  }
```

Chuỗi cũng có thể được khởi tạo bằng con trỏ . Ta xét ví dụ sau

Chương trình 1-11 :

```
main()
  {
   char *chao="Xin chao !";
```

```
   char ten[30];

   clrscr();
   printf("Cho ten cua ban : ");
   gets(ten);
   printf(chao);
   puts(ten);
   getch();
  }
```

Trong chương trình trên ta đã khởi tạo chuỗi bằng phát biểu

char *chao = " Xin chao !"

thay cho

static char chao[]=" Xin chao !"

Cả hai cách đều cho cùng một kết quả . Trong phương án dùng con trỏ , chao là biến con trỏ nên có thể thay đổi được . Ví dụ phát biểu :

puts(++chao)

sẽ cho kết quả : in chao !

Nếu ta có một mảng chuỗi ta cũng có thể dùng mảng con trỏ trỏ tới mảng chuỗi này . Ta khởi tạo chúng giống như khởi tạo biến con trỏ đơn .

Chương trình 1-12 :

```
#define         max    5
main()
  {
   int dex;
   int enter=0;
   char name[40];
   static char *list[max]=
         {
           "Hung",
           "Ngan",
           "Van",
           "Hoa",
           "Tien"
         };
   clrscr();
   printf("Cho ten cua ban : ");
   gets(name);
   for(dex=0;dex<max;dex++)
     if (strcmp(list[dex],name)==0)
         enter=1;
   if (enter==1)
     printf("Ban da dang ki hoc lop C");
   else
     printf("Ban chua dang ki vao lop");
   getch();
  }
```

Phát biểu char *list[max] nói rằng list là một mảng con trỏ gồm max phần tử chỉ tới các kí tự . Chúng ta xét tiếp một ví dụ như sau :

Chương trình 1-13 : Nhập vào một dãy tên và sắp xếp lại đúng thứ tự a,b,c

```
#define   maxnum   38
#define   maxlen   81
main()
  {
    static char name[maxnum][maxlen];
    char *ptr[maxnum];
    char *temp;
    int count = 0;
    int in,out;

    clrscr();
    while (count<maxnum)
      {
          printf("Ban cho ten : ");
          gets(name[count]);
          if (strlen(name[count])==0)
            break;
          ptr[count++]=name[count];
      }
    for (out=0;out<count-1;out++)
      for (in=out+1;in<count;in++)
          if (strcmp(ptr[out],ptr[in])>0)
            {
              temp=ptr[in];
              ptr[in]=ptr[out];
              ptr[out]=temp;
            }
    printf("Danh sach da sap xep :\n");
    for(out=0;out<count;out++)
      printf("Ten thu %d : %s\n",out+1,ptr[out]);
    getch();
  }
```

Chương trình này dùng cả mảng chuỗi và mảng con trỏ chuỗi . Con trỏ nằm trong mảng được khai báo như sau :

char *ptr[maxnum]

chuỗi nằm trong mảng hai chiều

static char name[maxnum][maxlen]

Do ta không biết một chuỗi dài bao nhiêu nên phải dùng mảng chuỗi name có tối đa maxnum phần tử , mỗi phần tử có maxlen kí tự . Khi nhập chuỗi phát biểu

ptr[count++] = name[count

sẽ gán địa chỉ của mỗi chuỗi được cất giữ trong mảng name[][] vào phần tử con trỏ ptr . Sau đó mảng con trỏ này được sắp xếp dựa trên mảng name[][] nhương mảng name[][] không thay đổi gì cả .

Ngôn ngữ C có thể xử lí các thành phần của mảng như một mảng . Cụ thể C có thể xem một dòng của mảng hai chiều như là một mảng một chiều. điều này rất tiện lợi như ta đã thấy trong chương trình trên . Câu lệnh ptr[count++] = name[count hoàn toàn hợp lí vì vế