Chương IX

POINTER

1

KHÁI NIỆM

Trong ngôn ngữ C, mỗi biến và chuỗi ký tự đều được lưu trữ trong bộ nhớ và có địa chỉ riêng, địa chỉ này xác định vị trí của chúng trong bộ nhớ → C đưa ra kiểu dữ liệu pointer (tạm dịch là con trỏ) để khai báo cho các biến lưu địa chỉ.

Đây là một kiểu dữ liệu đặc biệt và được sử dụng nhiều trong một chương trình C.

Một biến có kiểu pointer có thể lưu được dữ liệu trong nó, là địa chỉ của một đối tượng đang khảo sát.

- Biến
- Chuỗi
- Hàm

2

THAO TÁC TRÊN POINTER

Khai báo biến pointer - pointer hằng

- Trong ngôn ngữ C có một toán tử lấy địa chỉ của một biến đang làm việc, toán tử này là một dấu & (ampersand), tạm gọi là toán tử lấy địa chỉ. Cú pháp như sau:

& biến

với **biến** là một biến thuộc kiểu bất kỳ, nhưng không được là biến thanh ghi.

Ví dụ Nếu có một biến đã được khai báo là

int hệ_số_a;

thì

& hệ_số_a

sẽ là địa chỉ của biến hệ_số_a.

Có nghĩa là khi khai báo, biến hệ_số_a thì được cấp phát một vùng nhớ trong bộ nhớ máy tính. Địa chỉ đầu của vùng nhớ này chính là địa chỉ của biến hệ_số_a: &hệ_số_a

CBGD: ThS.Trần Anh Dũng

3

THAO TÁC TRÊN POINTER

Kết quả của phép toán lấy địa chỉ của một biến là một hằng pointer hằng trỏ đến biến đó, địa chỉ hằng này có thể được xem như một giá trị để gán vào biến pointer.
Hằng pointer cũng có thể là tên mảng hoặc tên hàm.

CBGD: ThS.Trần Anh Dũng

4

THAO TÁC TRÊN POINTER

Ví dụ

Nếu biến con trỏ pint là một biến pointer chỉ để lưu địa chỉ của đối tượng thuộc kiểu int, thì việc gán địa chỉ của một biến thuộc kiểu int, ví dụ biến số_trang, vào cho biến pint là hoàn toàn hợp lệ:

```
pint = &số_trang;
```

Dĩ nhiên ta **không thể** thực hiện việc gán sau:

```
pint = &(số_trang + 2);
```

vì (số_trang + 2) không phải là một biến nên không có địa chỉ trong bộ nhớ.

THAO TÁC TRÊN POINTER

Biến hoặc đối tượng mà con trỏ đang trỏ đến có thể được truy xuất qua tên của biến con trỏ và dấu "*" đi ngay trước biến con trỏ, cú pháp cụ thể như sau:

```
* tên_biến_con_trỏ
```

Ví dụ: *pint;

- Ngược lại, để khai báo một biến pointer để chứa địa chỉ của một đối tượng hoặc lấy một nội dung của đối tượng mà con trỏ đang trỏ đến, C dùng toán tử "*" (dấu hoa thị- asterisk). Cú pháp để khai báo biến pointer:

```
kiểu * tên_biến_pointer
```

Ví dụ: int *pint;

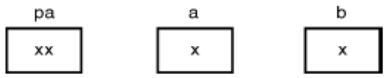
```
pint = &số_trang;
```

THAO TÁC TRÊN POINTER

Ví dụ Xét các khai báo sau:

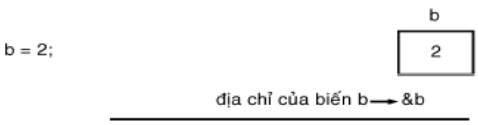
```
int a, b;  /* Khai báo biến a, b thuộc kiểu int */
int *pa;  /* Biến con trỏ pa sẽ trỏ đến một đối tượng int */
```

Sau khi khai báo, ta có ba ô nhớ cho ba biến a, b và pa như sau:



Lúc này, trong cả ba biến đều đang có trị rác. Nếu thực hiện các phép toán sau đây thì sự thay đổi trị trong mỗi ô nhớ của mỗi biến có thể được biểu diễn qua các hình vẽ kèm theo như sau:

Biến a được gán trị là 1

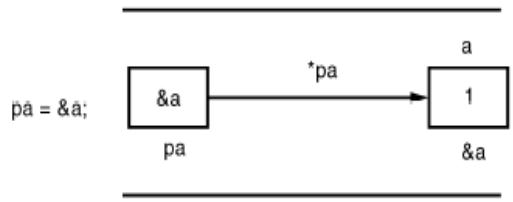


Biến b được gán trị là 2

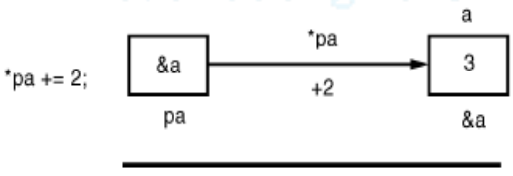
CBGD: ThS.Trần Anh Dũng

7

THAO TÁC TRÊN POINTER



Lấy địa chỉ của biến a, là &a, gán cho pa, nên biến pa trỏ đến biến a. là đối tượng của biến a



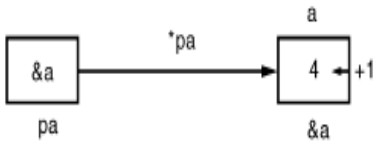
Tăng nội dung của đối tượng *pa, là một biến int, thêm 2

CBGD: ThS.Trần Anh Dũng

8

THAO TÁC TRÊN POINTER

a = *pa +1;



Mọi thao tác thực hiệu trên *pa, cũng chính là thực hiện trên biến gốc a, và ngược lại

pa = &b

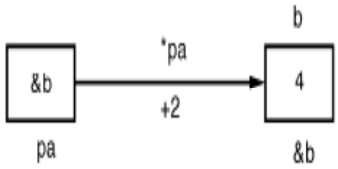
Lấy địa chỉ của biến b, là &b, gán cho pa, nên biến pa trỏ đến biến b. Biến b là đối tượng của biến pa

CBGD: ThS.Trần Anh Dũng

9

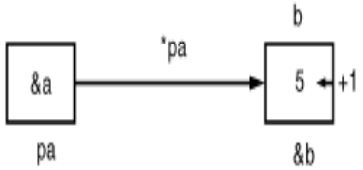
THAO TÁC TRÊN POINTER

*pa += 2;



Tăng nội dung của đối tượng *pa, là một biến int, thêm 2

b = *pa +1;



Mọi tham tác thực hiện trên *pa, cũng chính là thực hiện trên biến gốc b, và ngược lại

CBGD: ThS.Trần Anh Dũng

10

THAO TÁC TRÊN POINTER

Như vậy ta thấy rằng, biến pointer không có kiểu riêng cho nó, nên nó phải được khai báo qua kiểu của đối tượng mà nó sẽ trỏ đến. Do đó, trước khi quyết định khai báo biến pointer, ta cần phải xác định đối tượng của nó thuộc kiểu gì. Nếu chưa xác định được kiểu của đối tượng cần khai báo, C cho phép khai báo một biến pointer trỏ đến đối tượng thuộc kiểu void (không kiểu) như sau:

Ví dụ: `void * pvoid;` ⟶ biến pointer **pvoid** trỏ đến biến kiểu void

Như vậy, bằng các phép lấy địa chỉ của một biến và lấy đối tượng của một pointer, ta có thể truy xuất gián tiếp đến một biến nào đó, mà không cần sử dụng đến tên biến.

11

THAO TÁC TRÊN POINTER

- Có thể cộng, trừ một pointer với một số nguyên (int, long,...). Kết quả là một pointer.
- Pointer có được từ phép cộng hoặc trừ trên, sẽ chỉ đến một đối tượng mới lệch với đối tượng cũ n phần tử, nếu tính theo byte thì đối tượng cũ lệch với đối tượng mới số byte bằng n lần kích thước byte của kiểu đối tượng mà con trỏ đang trỏ đến.

Ví dụ

```
int *pi1, *pi2, n;
pi1 = &n;
pi2 = pi1 + 3;
```

Trong ví dụ trên, biến con trỏ pi1 sau khi được gán trị &n, sẽ chỉ đến biến n, biến pi2 được gán trị từ pi1 cộng thêm 3, điều này có ý nghĩa là biến pi2 trỏ đến một đối tượng mới, lệnh với biến n ba phần tử int, tức 3 × 2 = 6 byte.

12

THAO TÁC TRÊN POINTER

Khi lập trình, thao tác cộng và trừ này có thể được sử dụng để truy xuất mảng vì mảng là một cấu trúc trong đó các phần tử được sắp xếp liên tiếp nhau trong bộ nhớ. Do đó, nếu một biến con trỏ được gán trị là địa chỉ của một biến thành phần nào đó của mảng, thì việc cộng hay trừ biến này với một số nguyên sẽ cho ra địa chỉ của biến thành phần mới, lệch với biến thành phần cũ n phần tử.

Ví dụ Cho khai báo

```
int a[20];      /* Mảng a có 20 phần tử int liên tiếp */
int *p;         /* Biến pointer chỉ đến một đối tượng int */
p = &a[0];   /* p là địa chỉ của phần tử đầu tiên của mảng a */
p += 3;         /* p lưu địa chỉ phần tử a[0 + 3], tức &a[3] */
```


CBGD: ThS.Trần Anh Dũng

13

THAO TÁC TRÊN POINTER

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
main()
    {
        int n, *p1, *p2, i;
        int a[20];
        clrscr();
        printf ("Mang co bao nhieu phan tu:");
        scanf ("%d", &n);
        p1 = p2 = &a[0];
        printf(Nhap cac phan tu cua mang: \n);
        for (i = 0; i < n; i++)
            {
                scanf("%d", p1);
                p1++;
            }
        printf (Cac tri trong mang la: );
        for (i =0; i <n; i++)
            {
                printf("%d", *p2);
                p2++;
            }
        getch();
    }
```

Chương trình cho xuất liệu ví dụ:

```
Mang co bao nhieu phan tu: 5
Nhap cac phan tu cua mang: 5 7 9 0 5
Cac tri trong mang la: 5 7 9 0 5
```

CBGD: ThS.Trần Anh Dũng

14

THAO TÁC TRÊN POINTER

Trong chương trình ví dụ trên, ta dùng hai biến con trỏ để lưu địa chỉ đầu của mảng p1 và p2. Biến p1 được sử dụng trong vòng lặp nhập trị cho các phần tử của mảng, biến p2 lại được sử dụng trong vòng lặp truy xuất mảng, vì sau các thao tác cộng trị, biến con trỏ đã bị thay đổi trị.

- Không thể thực hiện các phép toán nhân, chia, hoặc lấy dư một pointer với một số, vì pointer lưu địa chỉ, nên nếu thực hiện được điều này cũng không có một ý nghĩa nào cả.

Có nghĩa là p1 đã bị thay đổi về trị qua phép p1++ nên không thể gọi mảng thông qua p1 mà phải thông qua p2

CBGD: ThS.Trần Anh Dũng

THAO TÁC TRÊN POINTER

Phép trừ giữa hai pointer vẫn là một phép toán hợp lệ, kết quả là một trị thuộc kiểu int biểu thị khoảng cách (số phần tử) giữa hai pointer đó

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
main()
{
    int *p1, *p2;
    int a[10];
    clrscr();
    p1 = &a[0];
    p2 = &a[5];
    printf ("Dia chi cua bien a[0] la: %p\n", p1);
    printf ("Dia chi cua bien a[5] la: %p\n", p2);
    printf ("Khoang cach giua hai phan tu la %d int\n", p2 - p1);
    getch();
}
```

Chương trình sẽ cho xuất liệu ví dụ:

```
Dia chi cua bien a[0] la: FFE2
Dia chi cua bien a[5] la: FFEC
Khoang cach giua hai phan tu la 5 int
```

CBGD: ThS.Trần Anh Dũng

THAO TÁC TRÊN POINTER

- Không thể thực hiện việc cộng hai pointer lại với nhau.

- Ngoài ra, biến pointer có thể được xử lý như một biến bình thường, có nghĩa là ta có thể thực hiện các phép toán so sánh, gán, tăng, giảm .... Các phép toán này đòi hỏi các pointer là toán hạng phải có cùng kiểu. Nếu việc thực hiện phép toán trên con trỏ là không rõ ràng, bộ dịch sẽ nhắc nhở ngay.

Ví dụ Cho các khai báo sau:

```
int * a1;
char * a2;
a1 = 0;              /* Chương trình dịch sẽ nhắc nhở lệnh này */
a2 = (char *)0;
if( a1 != a2) /* Chương trình dịch sẽ nhắc nhở kiểu của đối tượng */
        {
            a1 = (int *) a2; /* Hợp lệ vì đã ép kiểu */
        }
```

CBGD: ThS.Trần Anh Dũng

17

THAO TÁC TRÊN POINTER

- Có thể áp dụng phép toán chuyển kiểu bắt buộc để chuyển kiểu một pointer, chuyển một pointer đang chỉ đến một đối tượng thuộc kiểu nào đó thành pointer chỉ đến một đối tượng thuộc kiểu khác, cả hai đối tượng đều cùng địa chỉ. Phép chuyển kiểu này thường được dùng để đổi kiểu của một pointer thành một kiểu pointer khác để có thể lấy đối tượng của nó với kiểu khác mà ta mong muốn.

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
main()
        {
            int *pint, a = 0 6141;
            char *pchar;
            clrscr();
            pint = &a;
            pchar = (char *) &a;
            printf ("Tri cua bien pint la: %p\n", pint);
            printf ("Tri cua bien pchar la: %p\n", pchar);
            printf ("Doi tuong pint dang quan ly la %X \n", *pint);
            printf ("Doi tuong pchar dang quan ly la %c \n", *pchar);
            pchar++;
            printf ("Doi tuong pchar dang quan ly la %c \n", *pchar);
            getch();
        }
```

Chương trình cho xuất liệu ví dụ:

```
Tri cua bien pint la: FFF4          Doi tuong pchar dang quan ly la A
Tri cua bien pchar la: FFF4         Doi tuong pchar dang quan ly la a
Doi tuong pint dang quan ly la 6141
```

CBGD: ThS.Trần Anh Dũng

18

THAO TÁC TRÊN POINTER

C cho phép khai báo một biến pointer là hằng hoặc đối tượng của một pointer là hằng. Lúc đó, việc gán, hoặc tăng giảm trị lưu trong pointer hằng là không hợp lệ, hoặc thay đổi đối tượng của pointer khi pointer được khai báo là chỉ đến một đối tượng hằng đều bị C báo lỗi

Ví dụ

Các khai báo sau đây là biến pointer hằng:

1. `int a, b;`

   `int * const pint = &a;`

   biến pointer pint chỉ lưu địa chỉ của biến a, và được khởi động trị lúc mới khai báo mà thôi

   `pint = &b;` ← *C báo lỗi*

2. `char * const ps = "ABCD";`

   ps là một pointer hằng chỉ đến chuỗi ký tự

ps là một pointer hằng chỉ đến một chuỗi ký tự

THAO TÁC TRÊN POINTER

Các khai báo sau đây cho thấy đối tượng của một pointer là hằng:

1. `int i;`

   `const int * pint;`

   *khai báo này cho thấy *pint sau này sẽ là một đối tượng hằng*

   `pint = &i;`

   `i = 1;`

   `(*pint) ++;` → *lệnh này sẽ báo lỗi, vì không thể thay đổi trị của một đối tượng hằng của con trỏ*

   `i++;` → *hoàn toàn hợp lệ*

2. `const char * s = "ABCD"` hoặc

   `char const * s = "ABCD";`

   s là một biến pointer chỉ đến một chuỗi ký tự hằng

THAO TÁC TRÊN POINTER

Như vậy, việc đặt từ const trước hay sau dấu * trong khai báo pointer có ý nghĩa khác nhau:

const trước * : pointer chỉ đến đối tượng hằng

* trước const : pointer hằng

21

POINTER VÀ MẢNG

Có một sự tương quan chặt chẽ giữa mảng và pointer, vì trong C, tên mảng là một hằng pointer, nó chính là địa chỉ của phần tử đầu tiên của mảng. Do đó, ta hoàn toàn có thể truy xuất mảng bằng một pointer.

Ví dụ 6.14:

Cho khai báo sau:

```
int a[10];
int *pa;
```


thì a là một hằng pointer, do đó ta có thể dùng tên mảng là pointer để truy xuất mảng, đối tượng

*(a + 0) chính là a[0],

*(a + 1) chính là a[1],

....

*(a + i) chính là a[i].

22

POINTER VÀ MẢNG

Mặt khác, a là một hằng pointer nên ta hoàn toàn có khả năng gán trị này vào cho biến pa, hoặc địa chỉ của phần tử đầu tiên &a[0], để pa trỏ đến mảng

```
pa = a;
```

hoặc

```
pa =&a[0];
```

Khi đó, (pa + 0) sẽ chỉ đến phần tử a[0],

(pa + 1) sẽ chỉ đến phần tử a[1],

...

(pa + i) sẽ chỉ đến phần tử a[i].

23

POINTER VÀ MẢNG

Ví dụ 6.15: Xét chương trình sau:

```
#include <stdio.h>
#include <conio.h>
main()
        {
            int *pa, a[10];
            int i, s = 0;
            clrscr();
            pa = a;
            printf ("Moi nhap 10 phan tu can tinh tong \n");
            for (i = 0; i < 10; i++)
                scanf ("%d", pa + i);
            for (i = 0; i < 10; i++)
                s += pa[i];
            printf ("Tong cac phan tu cua mang la: %d \n", s);
            getch();
        }
```

Chương trình cho xuất liệu ví dụ:

```
Moi nhap 10 phan tu can tinh tong
2 3 5 9 0 5 7 3 7 5
Tong cac phan tu cua mang la: 46
```


24

POINTER VÀ MẢNG

- Ngoài ra, biến pointer có một địa chỉ trong bộ nhớ, còn không thể lấy địa chỉ của tên mảng.

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
main()
{
    int *pint, a;
    clrscr();
    pint = &a;
    printf ("Dia chi cua bien pint la: %p\n", &pint);
    printf ("Dia cua bien a, pint dang tro toi la: %p\n", pint);
    getch();
}
```

Chương trình sẽ cho xuất liệu ví dụ:

Dia chi cua bien pint la: FFF4

Dia cua bien a, pint dang tro toi la: FFF2

CBGD: ThS.Trần Anh Dũng

25

POINTER VÀ MẢNG

Khai báo đối số của hàm, có thể khai báo đối số giả dưới dạng mảng:

int a[]

hoặc có thể khai báo dưới dạng pointer:

int *a

Ví dụ

```
void nhap_tri_mang (int a[ ], int n)
{
    int i;
    for (i =0; i <n; i++)
        scanf ("%d", &a[i]);
}
```

đối số giả dưới dạng mảng

```
void nhap_tri_mang (int *a, int n)
{
    int i;
    for (i =0; i <n; i++)
        scanf ("%d", &a[i]);
}
```

đối số giả dưới dạng pointer

CBGD: ThS.Trần Anh Dũng

26

ĐỐI SỐ CỦA HÀM LÀ POINTER – TRUYỀN ĐỐI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

Bình thường, đối số của hàm được khai báo ở dạng tham số trị, khi đó hàm sẽ lấy giá trị của các đối số thật gởi cho hàm khi gọi hàm để tính toán trong hàm, mọi sự thay đổi trị của đối số thật trong hàm đều không thể thực hiện được, mà điều này trong nhiều trường hợp lại cần thiết.

Ví dụ

```
#include <stdio.h>
#include <conio.h>
void hoan_doi (int a, int b);
main()
        {
            int so1 = 1, so2 = 2;
            clrscr();
            printf ("Tri cua hai bien so1 va so2 la: %d %d\n", so1, so2);
            hoan_doi(so1, so2),
            printf("Sau khi goi ham, tri cua so1 va so2 la: %d %d\n",so1,so2);
            getch();
        }
```

CBGD: ThS.Trần Anh Dũng

27

ĐỐI SỐ CỦA HÀM LÀ POINTER – TRUYỀN ĐỐI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

```
void hoan_doi (int a, int b)
    {
        int temp;
        temp = a;
        a = b;
        b = temp;
    }
```

Chương trình sẽ cho xuất liệu ví dụ:

```
Tri cua hai bien so1 va so2 la: 1 2
Sau khi goi ham, tri cua so1 va so2 la: 1 2
```

Chương trình này không cho kết quả mong muốn, vì ta muốn sau khi gọi hàm, hai trị trong hai biến so1 và so2 sẽ được thay đổi cho nhau.

CBGD: ThS.Trần Anh Dũng

28

ĐỐI SỐ CỦA HÀM LÀ POINTER – TRUYỀN ĐỐI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

Để thực hiện được điều này, ta cần phải truyền địa chỉ của đối số thật vào cho hàm, khi đó việc khai báo danh sách đối số trong hàm ứng với đối số được truyền theo địa chỉ sẽ là pointer. Khi đó mọi sự thay đổi trên đối tượng mà con trỏ đang trỏ đến trong hàm chính là thay đổi trên đối số thật.

Cách cũ

```
void hoan_doi (int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

Cách mới

```
void hoan_doi (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

CBGD: ThS.Trần Anh Dũng

29

ĐỐI SỐ CỦA HÀM LÀ POINTER – TRUYỀN ĐỐI SỐ THEO SỐ DẠNG THAM SỐ BIẾN

```
#include <stdio.h>
#include <conio.h>
void hoan_doi (int *a, int *b);
main ()
    {
        int so1 = 1, so2 = 2;
        clrscr();
        printf ("Tri cua hai bien so1 va so2 la: %d %d\n", so1, so2);
        hoan_doi (&so1, &so2);
        printf ("Sau khi goi ham, tri cua so1 va so2 la: %d %d\n", so1, so2);
        getch();
    }
void hoan_doi (int *a, int *b)
    {
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
    }
```

Chương trình sẽ cho xuất liệu ví dụ:

Tri cua hai bien so1 và so2 la: 1 2

Sau khi goi ham, tri của so1 va so2 la: 2 1

CBGD: ThS.Trần Anh Dũng

30

HÀM TRẢ VỀ POINTER VÀ MẢNG

Cũng bình thường như các kiểu khác, một pointer có thể được trả về từ hàm, nếu pointer trỏ đến đối tượng là mảng, thì hàm trả về mảng; nếu pointer chỉ trỏ đến biến bình thường, thì hàm chỉ trả về con trỏ trỏ đến biến thường, cú pháp khai báo hàm như sau:

kiểu * tên_hàm (danh_sách_khai_báo_đối_số);

với **kiểu** là kiểu của đối tượng mà pointer được hàm trả về trỏ đến. Kiểu này là bất kỳ, nếu chưa xác định được kiểu của đối tượng trả về, có thể void. Khi muốn sử dụng pointer chỉ đến kiểu void, ta cần ép kiểu về kiểu ta cần làm việc.

HÀM TRẢ VỀ POINTER VÀ MẢNG

Ví dụ int * lon_nhat (int a, int b, int c);

Hàm lon_nhat() trả về một địa chỉ:

- Là địa chỉ của một int
- Hoặc địa chỉ của một mảng các int

Việc sử dụng địa chỉ theo đối tượng nào là do nơi gọi

HÀM TRẢ VỀ POINTER VÀ MẢNG

```
#include <stdio.h>
#include <conio.h>
int *nhap_tri(int *num);
main()
    {
        int *pint, so_phan_tu, i;
        clrscr();
        pint = nhap_tri (&so_phan_tu);
        printf ("Cac phan tu cua mang la:");
        for (i =0; i <so_phan_tu; i++) /* in trị của mảng */
            printf ("%d", pint[i]);
        getch();
    }
int *nhap_tri (int *num)
    {
        static int a[10];
        int i, n;
        printf ("Nhap kich thuoc mang: ");
        scanf (%d, &n);
        *num = n;
        printf ("Nhap tri %d phan tu cua mang: ", n);
        for (i = 0; i <n; i++)
            scanf ("%d", &a[i]);
        return a;
    }
```

Chương trình sẽ cho xuất liệu ví dụ:

```
Nhap kich thuoc mang: 8
Nhap tri cho 8 phan tu cua mang: 1 2 3 4 5 6 7 8
Cac phan tu cua mang la: 1 2 3 4 5 6 7 8
```

Pointer được trả qua lệnh return chính là tên hàm, là địa chỉ đầu mảng.
Đối số thật đưa vào cho hàm là một địa chỉ của một biến int để nhận trị này

CBGD: ThS.Trần Anh Dũng

33

CHUỖI KÝ TỰ

Như đã đề cập trong phần hằng chuỗi, một hằng chuỗi, khi được sử dụng, đều có một địa chỉ riêng trong bộ nhớ, địa chỉ này là pointer trỏ đến đầu chuỗi.

Ví dụ 6.26: Khi khai báo

```
Hello, world!
```

thì chuỗi này sẽ được C ghi vào một nơi nào đó trong bộ nhớ và có địa chỉ xác định. Địa chỉ này có thể được gán vào cho một biến con trỏ trỏ đến ký tự để quản lý chuỗi.

Ví dụ 6.27: Cho khai báo

```
char *s;
```

ta có thể gán trị cho biến pointer s, là một biến pointer trỏ đến ký tự, địa chỉ của chuỗi "Hello, world!";

CBGD: ThS.Trần Anh Dũng

34

CHUỖI KÝ TỰ

Ví dụ 6.28: Cho khai báo

```
char s[20];
s = Hello, world!;
```
→ *không hợp lệ*

Chính vì thao tác trên không hợp lệ, mà C cung cấp một số hàm để tiện cho việc truy xuất chuỗi.

Như vậy, các thao tác về chuỗi đều dựa trên pointer trỏ đến chuỗi, do đó việc gán trị là chuỗi vào cho mảng các ký tự để tạo ra một biến chuỗi là không hợp lệ

CBGD: ThS.Trần Anh Dũng

35

CHUỖI KÝ TỰ

1- Nhập trị chuỗi

Việc nhập trị cho chuỗi bao gồm hai bước: đầu tiên cần khai báo một nơi trống để chứa chuỗi, sau đó dùng một hàm nhập trị để lấy chuỗi.

Để có một nơi lưu chuỗi, cách đơn giản nhất là ta hãy khai báo một mảng ký tự, mảng này có thể là tĩnh hay tự động.

Ví dụ 6.29: Khai báo mảng

```
char s [40];
```

Việc nhập chuỗi vào mảng s có thể dùng một trong hai hàm: **gets**() hoặc **scanf**(). Cả hai hàm đều có prototype trong file stdio.h.

CBGD: ThS.Trần Anh Dũng

36

CHUỖI KÝ TỰ

1- Nhập trị chuỗi

Hàm gets()

```
char * gets (char * s);
```

Khi nhập trị, hàm này đọc các ký tự đến khi nào gặp ký tự quy định hàng mới (tức ký tự '\n', tức khi ta ấn phím ENTER) thì kết thúc việc nhập

Ví dụ

```
#include <stdio.h>
#include <conio.h>
main()
    {
        char ten[41];
        char *pten;
        clrscr();
        printf ("Ban ten gi?\n");
        pten = gets (ten);
        printf("%s? A! Chao ban %s\n", ten, pten);
        getch();
    }
```

Chương trình cho xuất liệu ví dụ:

```
Ban ten gi?
Dang Thanh Tin
Dang Thanh Tin? A! Chao ban Dang Thanh Tin
```

Hàm này trả về một pointer trỏ đến chuỗi, pointer này chính là tên mảng, là đối số sau khi gán chuỗi

CHUỖI KÝ TỰ

- ***Hàm scanf()*** cũng cho phép nhập chuỗi qua định dạng nhập %s. Việc nhập chuỗi sẽ kết thúc khi hàm scanf() gặp một trong các ký tự khoảng trắng, ký tự tab hay ký tự xuống hàng đầu tiên mà nó gặp. Đây chính là điểm khác nhau giữa hai hàm nhập chuỗi gets() và scanf().

Ví dụ

```
#include <stdio.h>
#include <conio.h>
main()
    {
        char ten1[41], ten2[41];
        clrscr();
        printf(Moi ban nhap hai ten: );
        scanf ("%s %s", ten1, ten2);
        printf("A! Chao hai ban %s va %s \n", ten1, ten2);
        getch();
    }
```

Chương trình cho xuất liệu ví dụ như sau:

```
Moi ban nhap hai ten: Dang Thanh
A! Chao hai ban Dang va Thanh
```

khoảng trắng phân biệt hai chuỗi

Lệnh gets hiểu Dang Thanh Tin là 1 chuỗi (không có dấu xuống hàng)
Lệnh scanf hiểu Dang Thanh Tin là 3 chuỗi (do dấu khoảng trắng)

CHUỖI KÝ TỰ

Để xuất chuỗi, hai hàm thường hay được dùng:

- puts()
- printf()

Cả hai hàm có prototype trong file stdio.h.

39

CHUỖI KÝ TỰ

2- Xuất chuỗi

- Hàm puts() là một hàm xuất chuỗi rất dễ sử dụng. Ta chỉ cần cung cấp cho hàm đối số là địa chỉ của chuỗi cần in. Hàm này sẽ đọc từng ký tự của chuỗi và in ra màn hình cho đến khi gặp ký tự NUL thì in ra màn hình thêm một ký tự xuống hàng nữa.

```
int puts (char * s);
```

Ví dụ

```
#include <stdio.h>
#include <conio.h>
main()
    {
        char ten[41];
        clrscr();
        printf("Moi ban nhap ten: ");
        gets(ten);
        printf("A! Chao ban: ");
        puts (ten);
        getch();
    }
```

Chương trình cho xuất liệu ví dụ:

```
Moi ban nhap ten: Dang Thanh Tin
A! Chao ban: Dang Thanh Tin
-
```

dấu nhảy (cursor) sau khi xuất chuỗi

40

CHUỖI KÝ TỰ

- ***Hàm printf*** () cũng cho phép xuất chuỗi ra màn hình nếu ta dùng định dạng xuất "%s" cho nó. Hàm này sẽ không tự động in thêm ký tự xuống hàng mới như hàm puts().

Ví dụ 6.33:

```
#include <stdio.h>
#include <conio.h>
main()
    {
        char ten[41];
        clrscr();
        printf ("Moi ban nhap ten: ");
        gets(ten);
        printf ("A! Chao ban: %s", ten);
        getch();
    }
```

Chương trình cho xuất liệu ví dụ:

```
Moi ban nhap ten: Dang Thanh Tin
A! Chao ban: Dang Thanh Tin
```


CBGD: ThS.Trần Anh Dũng

41

CHUỖI KÝ TỰ

3- Gán trị cho chuỗi

Việc gán trị cho biến chuỗi thực tế là việc chép từng ký tự từ hằng chuỗi hoặc biến chuỗi đã biết sang một biến chuỗi khác. Trong C, thao tác này được thực hiện nhờ hàm **strcpy()**, hàm này có prototype trong file string.h như sau:

```
char *strcpy(char *dest, const char *src);
```

Ví dụ

```
char *strcpy(char *dest, const char *src)
    {
        int i;
        for (i = 0; (dest[i] = src[i]) != '\0'; i++)
                    ;
        return dest;
    }
```


CBGD: ThS.Trần Anh Dũng

42

CHUỖI KÝ TỰ

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
    {
        char ten1[41], ten2[41];
        clrscr();
        printf("Moi ban nhap ten: ");
        gets(ten1);
        strcpy (ten2, ten1);
        printf("A! Chao ban: %s", ten2);
        getch();
    }
```

Chương trình cho xuất liệu ví dụ:

Moi ban nhap ten: Dang Thanh Tin

A! Chao ban: Dang Thanh Tin

Chú ý, nếu chiều dài chuỗi đích không đủ để nhận hết dữ liệu từ chuỗi nguồn thì C vẫn thực hiện việc gán trị cho chuỗi đích mà không báo lỗi nào, tuy nhiên chương trình sẽ chạy sai cho các lệnh dưới hoặc không kết thúc được

CHUỖI KÝ TỰ

4- Lấy chiều dài chuỗi

string.h: ← **size_t strlen(const char *s);**

Ví dụ Xét chương trình nhập một chuỗi, đổi các ký tự thường của chuỗi đó thành ký tự hoa, in chuỗi đó ra lại màn hình.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
    {
        char ten[41];
        int i;
        clrscr();
        printf("Moi ban nhap ten: ");
        gets(ten);
        for ( i = 0; i < strlen (ten); i++)
            if (ten[i] >= 'a' && ten[i] <= 'z')
                ten[i] -= 32;
        printf ("A! Chao ban: %s", ten);
        getch();
    }
```

Chương trình cho xuất liệu ví dụ như sau:

Moi ban nhap ten: dang thanh tin

A! Chao ban: DANG THANH TIN

CHUỖI KÝ TỰ

5- Nối chuỗi

string.h:

char *strcat(char *dest, const char *src):

Ví dụ 6.37:

Thiết kế chương trình nối hai chuỗi nhập từ bàn phím.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 41
main()
    {
        char hoten[41], ten[10];
        int i;
        clrscr();
        printf("Moi ban nhap Ho: ");
        gets(hoten);
        printf("Moi ban nhap Ten: ");
        gets(ten);
        if ( strlen (hoten) + strlen (ten) <MAX )
            {
                strcat ( hoten, ten );
                printf("A! Chao ban: %s", hoten);
            }
        else
        printf ("Khong du chieu dai chuoi! \n");
        getch();
    }
```

Chương trình cho xuất liệu ví dụ:

```
Moi ban nhap Ho: Dang Thanh
Moi ban nhap Ten: Tin
A! Chao ban: Dang Thanh Tin
```

Khi nhập có thêm khoảng ký tự trắng tại đây

45

CHUỖI KÝ TỰ

6- So sánh chuỗi

string.h:

int strcmp(const char *s1, const char*s2);

Hàm này so sánh hai chuỗi s1 và s2 và trả về một trị là:

- số dương nếu s1 > s2
- số âm nếu s1 < s2
- số 0 nếu s1 == s2

46

CHUỖI KÝ TỰ

Ví dụ 6.38:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main()
    {
        clrscr();
        printf("%d \n", strcmp("QUAN", quan));
        printf("%d \n', strcmp("QUAN", "QUAN"));
        printf("%d \n", strcmp("quan", "QUAN"));
        printf("%d \n", strcmp("quang, quanG"));
        printf("%d \n", strcmp("quang, quan"));
        getch();
    }
```

Chương trình cho xuất liệu:

```
-32
 0
32
32
103
```


POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

C cho phép khai báo các biến động, các biến này khi cần thì xin chỗ, không cần thì giải phóng vùng nhớ cho chương trình sử dụng vào mục đích khác.

Các biến động này được cấp phát trong vùng nhớ heap, là vùng đáy bộ nhớ, và được quản lý bởi các biến pointer

có prototype nằm trong file **alloc.h** hoặc **stdlib.h**:

```
void *malloc(size_t size);
void *calloc(size_t nitems, size_t size);
```

với - **size_t** là kiểu để khai báo kích thước khối bộ nhớ động cần xin, size_t có thể là int hoặc unsigned.

- **nitems** là số phần tử cần xin.

Nếu hàm này xin được khối bộ nhớ cần thiết thì chúng sẽ trả về một pointer trỏ đến đầu khối này

Nếu không xin được khối bộ nhớ cần thiết, hàm sẽ về trị là một con trỏ NULL

POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

Ví dụ 6.39:

Cần xin một khối bộ nhớ có 10 phần tử int, ta viết như sau:

```
int *pint;
pint = (int *) malloc (10 * siezof (int));
```

hoặc

```
pint = (int *) calloc (10, siezof (int));
```


POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

Tuy nhiên, biến động là biến được xin trong vùng nhớ heap, biến này sẽ không bị mất đi bình thường khi kết thúc chương trình, nên C đưa ra hàm free() để giải phóng khối bộ nhớ được xin bằng hàm malloc() hoặc calloc().

```
void free (void * block);
```

với - block là pointer trỏ đến đầu khối bộ nhớ trong vùng nhớ heap cần giải phóng.

Ví dụ 6.40:

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
main()
    {
        int *pint, s = 0, i;
        pint = (int *) calloc (10, sizeof (int));          /* xin biến động */
        /* hoặc pint = (int *) malloc (10 * sizeof (int)); */
```

POINTER VÀ VIỆC ĐỊNH VỊ BỘ NHỚ ĐỘNG

```
if (pint == NULL)                                   /* kiểm tra sự tồn tại */
    {                                                   /* của biến động */
        printf ("Khong du bo nho \n");
        exit (1);
    }
clrscr();
printf ("Moi nhap 10 tri vao mang: ");
for (i = 0; i <10; i++)                                 /* Thao tác trên biến */
    scanf ("%d", &pint[i]);                         /* động                 */
for (i = 0; i <10; i++)
    s += pint[i];
printf ("Tong cac phan tu cua mang la: %d \n", s);
getch();
free (pint);                                        /* giải phóng biến động */
}
```

Chương trình cho xuất liệu ví dụ:

Moi nhap 10 tri vao mang: 1 2 3 4 5 6 7 8 9 0

Tong cac phan tu cua mang la: 45

MẢNG CÁC POINTER

C cho phép khai báo mảng các pointer, việc khai báo và sử dụng mảng các pointer làm cho một chương trình C rất linh động trong việc quản lý dữ liệu, nhất

Cú pháp khai báo mảng các pointer:

kiểu * tên_mảng [Kích_thước];

Ví dụ 6.41: Khi khai báo

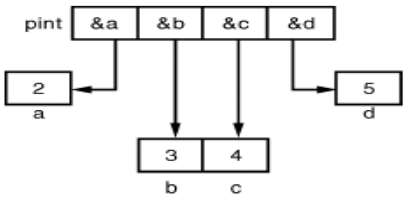
```
int * pint[4];
```

thì pint là tên một mảng, mảng này được cấp chỗ trong bộ nhớ để chứa bốn pointer,

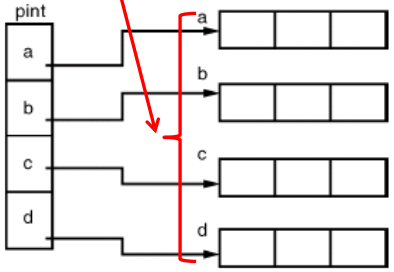
MẢNG CÁC POINTER

Ví dụ 6.42: Khi khai báo

```
int * pint[4];
int a = 2, b = 3, c = 4, d = 5;
pint[0] = &a;
pint[1] = &b;
pint[2] = &c;
pint[3] = &d;
```




```
int * pint[4];
int a[3], b[3], c[3], d[3];
pint[0] = a;
pint[1] = b;
pint[2] = c;
pint[3] = d;
```



CBGD: ThS.Trần Anh Dũng

53

MẢNG CÁC POINTER

Ví dụ 6.43:

```
#include <stdio.h>
#include <conio.h>
void sap_xep(int *pi[], int num);
main()
        {
            int *pint[4], i;
            int a = 2, b = 3, c = 4, d = 5;
            clrscr();
            pint[0] = &a;
            pint[1] = &b;
            pint[2] = &c;
            pint[3] = &d;
            sap_xep(pint, 4);
            printf ("Cac phan tu cua mang la: ");
            for (i = 0; i < 4; i++)
                printf ("%d", *pint[i]);
            getch();
        }
```


CBGD: ThS.Trần Anh Dũng

54

MẢNG CÁC POINTER

```
void sap_xep(int *pi[], int num)
    {
        int i, j, max, vtmax, *ptam;
        for (i = 0; i < num-1; i++)
            {
                max = *pi[i];
                vtmax = i;
                for (j = i+1; j < num; j++)
                    if (max < *pi[j])
                        {
                            max = *pi[j];
                            vtmax = j;
                        }
                ptam = pi[i];
                pi[i] = pi[vtmax];
                pi[vtmax] = ptam;
            }
    }
```

Chương trình cho xuất liệu:

Cac phan tu cua mang la: 5 4 3 2

CBGD: ThS.Trần Anh Dũng

55

MẢNG CÁC POINTER

Ví dụ

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>
#define MAX 81
void sap_xep(char *ps[], int num);
main()
    {
        char *s[4];
        int i;
        clrscr();
        for (i = 0; i < 4; i++)
            {
                s[i] = (char * ) malloc (MAX * sizeof(char));
                if ( s[i] == NULL )
                    {
                        printf ("Khong du bo nho dong cho thao tac \n");
                        exit (0);
                    }
                printf ("Moi nhap chuoi thu %d: ", i+1);
                gets (s[i]);
            }
        sap_xep(s, 4);
        printf ("Cac chuoi duoc sap xep la: \n");
        for (i = 0; i < 4; i++)
            puts (s[i]);
        getch();
        for (i = 0; i < 4; i++)
            free(s[i]);
    }
```

CBGD: ThS.Trần Anh Dũng

56

MẢNG CÁC POINTER

```
void sap_xep(char *ps[], int num)
    {
        int i, j, vtmax;
        char *max, *ptam;
        for (i = 0; i < num-1; i++)
            {
                max = ps[i];
                vtmax = i;
                for (j = i+1; j < num; j++)
                    if ( strcmp (max, ps[i]) < 0 )
                        {
                            max = ps[j];
                            vtmax = j;
                        }
                ptam = ps[i];
                ps[i] = ps[vtmax];
                ps[vtmax] = ptam;
            }
    }
```

hoán đổi hai pointer với nhau

Chương trình trên cho xuất liệu ví dụ:

Moi nhap chuoi thu 1: Dang Thanh Tin	Cac chuoi duoc sap xep la:Vo Van Danh
Moi nhap chuoi thu 2: Vo Van Danh	Ngo Hung Phuong
Moi nhap chuoi thu 3: Doan Sy Tri	Doan Sy Tri
Moi nhap chuoi thu 4: Ngo Hung Phuong	Dang Thanh Tin

CBGD: ThS.Trần Anh Dũng

57

MẢNG CÁC POINTER

Một mảng các pointer cũng có thể được khởi động trị nếu mảng là mảng toàn cục hay mảng tĩnh.

Ví dụ 6.46:

```
static char *thu[7] = {
                        "Thu 2", "Thu 3", "Thu 4", "Thu 5",
                        "Thu 6", "Thu 7", "Chua nhat"
                      };
```


CBGD: ThS.Trần Anh Dũng

58

POINTER CỦA POINTER

bản thân biến pointer cũng có địa chỉ, địa chỉ này là địa chỉ của một biến pointer. C cho phép khai báo một pointer trỏ đến một đối tượng là một pointer

kiểu ** tên_pointer

Chú ý rằng, trước tên_pointer là hai dấu * biểu thị tên_pointer là pointer của pointer chỉ đến một đối tượng có kiểu là kiểu.

Ví dụ 6.47:

```
int **pint;
```

thì biến pint là một biến pointer chỉ đến đối tượng là một pointer, pointer này lại chỉ đến đối tượng là một int, hoặc mảng các int.

POINTER CỦA POINTER

Do đó, ta có thể dùng biến pointer của pointer để quản lý mảng hai chiều, khi đó, tên của mảng hai chiều là một hằng pointer, hằng này có thể được sử dụng để gán vào cho biến pointer của pointer.

Ví dụ 6.48:

```
int **pint;
int a[4][4]
```

thì `pint = a;`

là một lệnh hợp lệ

POINTER CỦA POINTER

Ví dụ sau đây, pointer của pointer chỉ đến mảng các pointer, mảng này chứa các địa chỉ của các biến int.

Ví dụ:

```
int *m[4];
int a = 1, b = 2, c = 3, d = 4;
int **pint;
pint = m;
m[0] = &a;
m[1] = &b;
m[2] = &c;
m[3] = &d;
```


ĐỐI SỐ CỦA HÀM MAIN

C hoàn toàn cho phép việc nhận đối số vào hàm main(), có hai đối số C đã quy định theo thứ tự:

int agrc: đối số cho biết số tham số đã nhập, kể cả tên chương trình.

char *argv[]: mảng các pointer trỏ đến các chuỗi là tham số đi theo sau tên chương trình khi chạy chương trình từ DOS. Chuỗi là tên chương trình luôn được chỉ bởi agrv[0].

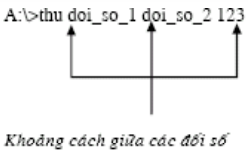
Ví dụ 6.52:

Một chương trình có tên là thu.C, hàm main() được khai báo:

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

Khi chạy chương trình từ DOS, ví dụ, nhập từ dấu nhắc

ĐỐI SỐ CỦA HÀM MAIN



thì trong chương trình ta có:

- argc = 4 vì có 4 đối số được truyền vào cho chương trình.
- argv[0] chỉ chuỗi "thu"

  argv[1] chỉ chuỗi "doi_so_1"

  argv[2] chỉ chuỗi "doi_so_2"

  argv[3] chỉ chuỗi "123"

  argv[4] là pointer NULL: hết đối số

Như vậy khi chạy chương trình, ta luôn gọi tên chương trình, nên argc tối thiểu là 1.

ĐỐI SỐ CỦA HÀM MAIN

Ví dụ 6.53: Xét chương trình ví dụ sau:

```
#include <stdio.h>
#include <conio.h>
main (int argc, char *argv[])
        {
            int i;
            clrscr();
            printf ("Cac doi so cua chuong trinh la: \n");
            printf ("Ten chuong trinh la: %s \n", argv[0]);
            if ( argc >1 )
                for (i = 1; i < argc; i++)
                    printf ("Doi so thu %d: %s \n", i, argv[i]);
            getch();
        }
```

ĐỐI SỐ CỦA HÀM MAIN

Nếu nhập từ bàn phím như sau

C:\>thu_main tin thu 123 ↵

thì chương trình cho xuất liệu là:

```
Cac doi so cua chuong trinh la:
Ten chuong trinh la: C:\thu_main.exe
Doi so thu 1: tinDoi so thu 2: thu
Doi so thu 3: 123
```


ĐỐI SỐ CỦA HÀM MAIN

Ví dụ 6.54:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main (int argc, char *argv[])
    {
        int i, tong = 0;
        if ( argc > 1 )
            {
                for (i = 1; i < argc; i++)
                    tong += atoi(argv[i]);
                printf ("Tong la %d \n",tong);
            }
        else
            printf ("Chuong trinh can doi so di theo \n");
        getch();
    }
```

Khi chạy chương trình từ DOS, ta có xuất liệu:

```
C:\>main 1 2 3 4 5
Tong la 15
```

Trong chương trình trên, ta có sử dụng hàm atoi(), đây là hàm đổi từ chuỗi sang số int, hàm có prototype trong file stdlib.h như sau:

```
int atoi (const char *s);
```

POINTER TRỎ ĐẾN HÀM

Cú pháp khai báo một pointer chỉ tới hàm:

kiểu (* tên_pointer) (kiểu_các_đối_số);

với: - **kiểu** là kiểu bất kỳ, đây là kiểu dữ liệu mà hàm trả về cho nơi gọi. Kiểu này bắt buộc phải có, dù hàm trả về dữ liệu int.

- **tên_pointer** là tên biến pointer chỉ tới hàm, đây là một khai báo biến bình thường, nên sau khai báo, C cấp vùng nhớ cho biến tên_pointer.

- **kiểu_các_đối_số** xác định kiểu của các đối số cần đưa vào cho hàm, mà hàm này đang được pointer chỉ tới. Kiểu_các_đối_số này có thể có hay không cũng được, nếu có, C sẽ kiểm tra đối số đưa vào có phù hợp không, nếu không, C không cần quan tâm đến các đối số khi gọi hàm.

POINTER TRỎ ĐẾN HÀM

Ví dụ:

Nếu khai báo

```
int (* p_function) (int, int);
```

và đã có hàm

```
int cong (int a, int b)
  {
    ...
  }
```

thì ta có thể thực hiện phép gán

```
p_function = cong;
```

và sau đó có thể dùng pointer này gọi hàm

```
tong = (*p_function) (m, n);
```

ỨNG DỤNG

- Danh sách liên kết là stack
  - Đưa một phần tử vào stack – thao tác push
  - Lấy một phần tử từ stack – thao tác pop
  - Xem stack
  - Khởi động stack
- Danh sách liên kết là queue
  - Thêm một thông tin vào queue – thao tác add
  - Lấy một thông tin khỏi queue – thao tác delete
  - Xem trị hiện hành của phần tử đầu của queue
  - Khởi động queue

69

BÀI TẬP

1. Viết chương trình với một hàm cho phép truy xuất chuỗi trong stack (danh sách liên kết và mảng) và in ra màn hình thông tin theo thứ tự alphabet.

2. Dùng cấu trúc dữ liệu queue dạng danh sách liên kết, tính biểu thức dạng đa thức bất kỳ sau:

$$f(x) = a_o x^n + a_1 x^m + ... + a_{n-1} x^3 + a_n$$

trong phần thông tin có hai vùng biến

- hệ số
- số mũ

3. Viết chương trình với một hàm duyệt toàn bộ các phần tử trong queue, trả về số phần tử trong queue.

4. Viết chương trình tạo một danh sách liên kết lưu các thông tin là các số nguyên theo thứ tự từ lớn tới nhỏ. Thiết kế hàm insert() cho phép chèn một phần tử lưu thông tin số vào vị trí có thứ tự phù hợp trong chuỗi.

70