Bài 6a - MẢNG VÀ CON TRỎ

Nội dung bài học

I. Mảng

1. Mảng trong C

Mảng là một tập hợp các phần tử cố định có cùng một kiểu, gọi là kiểu phần tử. Kiểu phần tử có thể là: ký tự, số, chuỗi ký tự;

Ta có thể chia mảng làm 2 loại: mảng 1 chiều và mảng nhiều chiều.

2. Mảng một chiều

Mảng 1 chiều là một dãy các phần tử có cùng tên gọi, có 1 chỉ số để chỉ thứ tự của phần tử đó trong dãy. Mảng một chiều còn có thể hiểu như một Vector.

Khai báo mảng với số phần tử xác định (khai báo tường minh)

Cú pháp:

<Kiểu> <Tên mảng >[n]

Trong đó:

- Tên mảng: đây là một cái tên đặt đúng theo quy tắc đặt tên của danh biểu;
- n: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi kích thước của mảng là gì);
- Kiểu: mỗi phần tử của mảng có dữ liệu thuộc kiểu gì;
- Ở đây, ta khai báo một biến mảng gồm có n phần tử, phần tử thứ nhất là tên mảng [0], phần tử cuối cùng là tên mảng[n -1];

Ví dụ:

int a[10];

/* Khai báo biến mảng tên a, phần tử thứ nhất là a[0], phần tử cuối cùng là a[9].*/

Ta có thể coi mảng a là một dãy liên tiếp các phần tử trong bộ nhớ như sau:

Vị trí	0	1	2	3	4	5	6	7	8	9
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

Hình 1: Hình ảnh mảng a trong bộ nhớ

Khai báo mảng với số phần tử không xác định (khai báo không tường minh)

Cú pháp:

<Kiểu> <Tên mảng> <[]>

Khi khai báo, không cho biết rõ số phần tử của mảng, kiểu khai báo này thường được áp dụng trong các trường hợp: vừa khai báo vừa gán giá trị, khai báo mảng là tham số hình thức của hàm.

Vừa khai báo vừa gán giá trị

Cú pháp:

<Kiểu> <Tên mảng> []= {Các giá trị cách nhau bởi dấu phẩy}

Nếu vừa khai báo vừa gán giá trị thì mặc nhiên C sẽ hiểu số phần tử của mảng là số giá trị mà chúng ta gán cho mảng trong cặp dấu {}. Chúng ta có thể sử dụng hàm sizeof() để lấy số phần tử của mảng như sau:

Số phần tử=sizeof(tên mảng)/ sizeof(kiểu)

Truy xuất từng phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua Tên biến mảng theo sau là chỉ số nằm trong cặp dấu ngoặc vuông [ ]. Chẳng hạn a[0] là phần tử đầu tiên của mảng a được khai báo ở trên. Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.

Ví dụ 1:

int a[10];

Trong khai báo này, việc truy xuất các phần tử được chỉ ra trong hình 1. Chẳng hạn phần tử thứ 2 (có vị trí 1) là a[1]…

Ví dụ 2: Vừa khai báo vừa gán trị cho 1 mảng 1 chiều các số nguyên. In mảng số nguyên này lên màn hình.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int n,i,j,tam;
   int dayso[]={66,65,69,68,67,70};
   n=sizeof(dayso)/sizeof(int); /*Lay so phan tu*/
   printf("\n Noi dung cua mang ");
   for (i=0;i<n;i++)
      printf("%d ",dayso[i]);
   getch();
   return 0;
}
```

Ví dụ 3: Đổi một số nguyên dương thập phân thành số nhị phân. Việc chuyển đổi này được thực hiện bằng cách lấy số đó chia liên tiếp cho 2 cho tới khi bằng 0 và lấy các số dư theo chiều ngược lại để tạo thành số nhị phân. Ta sẽ dùng mảng một chiều để lưu lại các số dư đó.

```
#include<conio.h>
#include<stdio.h>
int main()
{
    unsigned int N;
    unsigned int Du;
    unsigned int NhiPhan[20],K=0;
    int i;
    printf("Nhap vao so nguyen N= ");scanf("%d",&N);
    do
    {
        Du=N % 2;
        NhiPhan[K]=Du;
        K++;
        N = N/2;
    } while(N>0);
    printf("Dang nhi phan la: ");
    for(i=K-1;i>=0;i--)
```

```
        printf("%d",NhiPhan[i]);
    getch();
    return 0;
}
```

Ví dụ 4: Nhập vào một dãy n số và sắp xếp các số theo thứ tự tăng. Có rất nhiều giải thuật sắp xếp. Một trong số đó được mô tả như sau: Đầu tiên đưa phần tử thứ nhất so sánh với các phần tử còn lại, nếu nó lớn hơn một phần tử đang so sánh thì đổi chỗ hai phần tử cho nhau rồi tiếp tục so sánh. Sau đó tiếp tục so sánh phần tử thứ hai với các phần tử từ thứ ba trở đi ... cứ tiếp tục như vậy cho đến phần tử thứ n-1.

```
#include<conio.h>
#include<stdio.h>
int main()
{
   int b[20], N, i,j,t;
/* 1.  Nhap so phan tu cua mang*/
   printf("So phan tu thuc te cua mang N= ");
   scanf("%d",&N);

/* 2. Nhap gia tri cac pha tu cua mang*/
   for(i=0; i< N; i++)
   {
      printf("Phan tu thu %d: ",i);scanf("%d",&b[i]);
   }

/* 3. Sap xep giam dan*/
   for(i=0;i<N-1;i++)
   {
      for(int j=i+1;j<N;j++)
      {
         if (b[i]>b[j])
         {
            t=b[i];
            b[i]=b[j];
            b[j]=t;
         }
      }
   }
```

```
  /* 4. In ket qua sau khi sap xep*/
    printf("Mang SAU khi sap xep: ");
    for (i=0; i<N;i++)
        printf("%d ",b[i]);

    getch();
    return 0;
}
```

Phiên bản khác của chương trình sử dụng hàm (sẽ học ở bài sau) : viết các hàm Nhap (Nhập các số), SapXep (Sắp xếp) và InMang (In các số); các tham số hình thức của các hàm này là 1 mảng không chỉ định rõ số phần tử tối đa, nhưng ta cần có thêm số phần tử thực tế được sử dụng của mảng là bao nhiêu, đây là một giá trị nguyên.

```
#include<conio.h>
#include<stdio.h>
void Nhap(int a[],int N)
{
    int i;
    for(i=0; i< N; i++)
    {
        printf("Phan tu thu %d: ",i);scanf("%d",&a[i]);
    }
}
void InMang(int a[], int N)
{
    int i;
    for (i=0; i<N;i++)
        printf("%d ",a[i]);
    printf("\n");
}
void SapXep(int a[], int N)
{
    int t,i;
    for(i=0;i<N-1;i++)
        for(int j=i+1;j<N;j++)
            if (a[i]>a[j])
            {
                t=a[i];
```

```
            a[i]=a[j];
            a[j]=t;
        }
}
int main()
{
   int b[20], N;
   printf("So phan tu thuc te cua mang N= ");
   scanf("%d",&N);
   Nhap(b,N);
   printf("Mang vua nhap: ");
   InMang(b,N);
   SapXep(b,N); /* G?i hàm s?p x?p*/
   printf("Mang sau khi sap xep: ");
   InMang(b,N);
   getch();
   return 0;
}
```

3. Mảng nhiều chiều

Mảng nhiều chiều là mảng có từ 2 chiều trở lên. Người ta thường sử dụng mảng nhiều chiều để lưu các ma trận, các tọa độ 2 chiều, 3 chiều…

Khai báo mảng 2 chiều tường minh

Cú pháp:

<Kiểu> <Tên mảng><[Số phần tử chiều 1]><[Số phần tử chiều 2]>

Ví dụ: Người ta cần lưu trữ thông tin của một ma trận gồm các số thực. Lúc này ta có thể khai báo một mảng 2 chiều như sau:

```
float m[8][9]; /* Khai báo mảng 2 chiều có 8*9 phần tử là số thực*/
```

Trong trường hợp này, ta đã khai báo cho một ma trận có tối đa là 8 dòng, mỗi dòng có tối đa là 9 cột. Hình ảnh của ma trận này được cho trong hình 2:

Dòng\Cột	0	1	2	3	4	5	6	7	8
0	m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]	M[0][5]	m[0][6]	m[0][7]	m[0][8]

1	m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]	M[1][5]	m[1][6]	m[1][7]	m[1][8]
2	m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]	M[2][5]	m[2][6]	m[2][7]	m[2][8]
3	m[3][0]	m[3][1]	m[3][2]	m[3][3]	m[3][4]	m[3][5]	m[3][6]	m[3][7]	m[3][8]
4	m[4][0]	m[4][1]	m[4][2]	m[4][3]	m[4][4]	m[4][5]	m[4][6]	m[4][7]	m[4][8]
5	m[5][0]	m[5][1]	m[5][2]	m[5][3]	m[5][4]	m[5][5]	m[5][6]	m[5][7]	m[5][8]
6	m[6][0]	m[6][1]	m[6][2]	m[6][3]	m[6][4]	m[6][5]	m[6][6]	m[6][7]	m[6][8]
7	m[7][0]	m[7][1]	m[7][2]	m[7][3]	m[7][4]	m[7][5]	m[7][6]	m[7][7]	m[7][8]

Hình 2: Ma trận được mô tả là 1 mảng 2 chiều

Khai báo mảng 2 chiều không tường minh

Để khai báo mảng 2 chiều không tường minh, ta vẫn phải chỉ ra số phần tử của chiều thứ hai (chiều cuối cùng).

Cú pháp:

<Kiểu> <Tên mảng> <[]><[Số phần tử chiều 2]>

Cách khai báo này cũng được áp dụng trong trường hợp vừa khai báo, vừa gán trị hay đặt mảng 2 chiều là tham số hình thức của hàm.

Truy xuất từng phần tử của mảng 2 chiều

Ta có thể truy xuất một phần tử của mảng hai chiều bằng cách viết ra tên mảng theo sau là hai chỉ số đặt trong hai cặp dấu ngoặc vuông. Chẳng hạn ta viết m[2][3].

Với cách truy xuất theo cách này, Tên mảng[Chỉ số 1][Chỉ số 2] có thể coi là 1 biến có kiểu được chỉ ra trong khai báo biến mảng.

Ví dụ 1: Viết chương trình cho phép nhập 2 ma trận a, b có m dòng n cột, thực hiện phép toán cộng hai ma trận a,b và in ma trận kết quả lên màn hình. Trong ví dụ này, ta sẽ sử dụng hàm để làm ngắn gọn hơn chương trình của ta. Ta sẽ viết các hàm: nhập 1 ma trận từ bàn phím, hiển thị ma trận lên màn hình, cộng 2 ma trận.

```
#include<conio.h>
#include<stdio.h>
void Nhap(int a[][10],int M,int N)
{
   int i,j;
   for(i=0;i<M;i++)
          for(j=0; j<N; j++)
          {
             printf("Phan tu o dong %d cot %d: ",i,j);
             scanf("%d",&a[i][j]);
          }
}
void InMaTran(int a[][10], int M, int N)
{
   int i,j;
   for(i=0;i<M;i++)
   {
      for(j=0; j< N; j++)
            printf("%d ",a[i][j]);
      printf("\n");
   }
}
/* Cong 2 ma tran A & B ket qua la ma tran C*/
void CongMaTran(int a[][10],int b[][10],int M,int N,int c[][10])
{
   int i,j;
   for(i=0;i<M;i++)
      for(j=0; j<N; j++)
            c[i][j]=a[i][j]+b[i][j];
}
int main()
{
```

```
    int a[10][10], b[10][10], M, N;
    int c[10][10];/* Ma tran tong*/
    printf("So dong M= "); scanf("%d",&M);
    printf("So cot N= "); scanf("%d",&N);
    printf("Nhap ma tran A\n");
    Nhap(a,M,N);
    printf("Nhap ma tran B\n");
    Nhap(b,M,N);
    printf("Ma tran A: \n");
    InMaTran(a,M,N);
    printf("Ma tran B: \n");
    InMaTran(b,M,N);
    CongMaTran(a,b,M,N,c);
    printf("Ma tran tong C:\n");
    InMaTran(c,M,N);
    getch();
    return 0;
}
```

Ví dụ 2: Nhập vào một ma trận 2 chiều gồm các số thực, in ra tổng của các phần tử trên đường chéo chính của ma trận này. Ta nhận thấy rằng giả sử ma trận a có M dòng, N cột thì các phần tử của đường chéo chính là các phần tử có dạng: a[i][i] với i ∈ [0…min(M,N)-1].

```
#include<conio.h>
#include<stdio.h>
int main()
{
    float a[10][10], T=0;
    int M, N, i,j, Min;
    printf("Ma tran co bao nhieu dong? ");scanf("%d",&M);
    printf("Ma tran co bao nhieu cot? ");scanf("%d",&N);
    for(i=0;i<M;i++)
        for(j=0; j<N; j++)
        {
            printf("Phan tu o dong %d cot %d: ",i,j);
            scanf("%f",&a[i][j]);
        }
```

```
    printf("Ma tran vua nhap: \n");
    for(i=0;i<M;i++)
    {
       for(j=0; j< N; j++)
             printf("%.2f ",a[i][j]);
       printf("\n");
    }
    Min =(M>N) ? N: M; /* Tìm giá tr? nh? nh?t c?a M & N*/
    for(i=0;i<Min;i++)
       T=T+a[i][i];
    printf("Tong cac phan tu o duong cheo chinh la: %f",T);
    getch();
    return 0;
}
```

II. Con trỏ

Các biến chúng ta đã biết và sử dụng trước đây đều là biến có kích thước và kiểu dữ liệu xác định. Người ta gọi các biến kiểu này là biến tĩnh. Khi khai báo biến tĩnh, một lượng ô nhớ cho các biến này sẽ được cấp phát mà không cần biết trong quá trình thực thi chương trình có sử dụng hết lượng ô nhớ này hay không. Mặt khác, các biến tĩnh dạng này sẽ tồn tại trong suốt thời gian thực thi chương trình dù có những biến mà chương trình chỉ sử dụng 1 lần rồi bỏ.

Một số hạn chế có thể gặp phải khi sử dụng các biến tĩnh:

- Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ;
- Cấp phát ô nhớ thiếu, chương trình thực thi bị lỗi.

Để tránh những hạn chế trên, ngôn ngữ C cung cấp cho ta một loại biến đặc biệt gọi là biến động với các đặc điểm sau:

- Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình;
- Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi;
- Sau khi sử dụng xong có thể giải phóng để tiết kiệm chỗ trong bộ nhớ.

Tuy nhiên các biến động không có địa chỉ nhất định nên ta không thể truy cập đến chúng được. Vì thế, ngôn ngữ C lại cung cấp cho ta một loại biến đặc biệt nữa để khắc phục tình trạng này, đó là biến con trỏ (pointer) với các đặc điểm:

- Biến con trỏ không chứa dữ liệu mà chỉ chứa địa chỉ của dữ liệu hay chứa địa chỉ của ô nhớ chứa dữ liệu;
- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte.

1. Khai báo và sử dụng biến con trỏ

Khai báo biến con trỏ

Cú pháp:

<Kiểu> * <Tên con trỏ> ;

Ý nghĩa: Khai báo một biến có tên là Tên con trỏ dùng để chứa địa chỉ của các biến có kiểu Kiểu.

Ví dụ 1: Khai báo 2 biến a,b có kiểu int và 2 biến pa, pb là 2 biến con trỏ kiểu int.

int a, b, *pa, *pb;

Ví dụ 2: Khai báo biến f kiểu float và biến pf là con trỏ float

float f, *pf;

Lưu ý: Nếu chưa muốn khai báo kiểu dữ liệu mà con trỏ ptr đang chỉ đến, ta sử dụng:

void *ptr;

sau đó, nếu ta muốn con trỏ ptr chỉ đến kiểu dữ liệu gì cũng được. Tác dụng của khai báo này là chỉ dành ra 2 bytes trong bộ nhớ để cấp phát cho biến con trỏ ptr.

Các thao tác trên con trỏ

Gán địa chỉ của biến cho biến con trỏ

Toán tử & dùng để định vị con trỏ đến địa chỉ của một biến đang làm việc.

Cú pháp:

<Tên biến con trỏ>=&<Tên biến>;

Giải thích: Ta gán địa chỉ của biến Tên biến cho con trỏ Tên biến con trỏ.

Ví dụ: Gán địa chỉ của biến a cho con trỏ pa, gán địa chỉ của biến b cho con trỏ pb.

pa=&a; pb=&b;

Lưu ý: Khi gán địa chỉ của biến tĩnh cho con trỏ cần phải lưu ý kiểu dữ liệu của chúng. Ví dụ sau đây không đúng do không tương thích kiểu:

```
int Bien_Nguyen;
float *Con_Tro_Thuc;
...
Con_Tro_Thuc=&Bien_Nguyen;
```

Phép gán ở đây là sai vì Con_Tro_Thuc là một con trỏ kiểu float (nó chỉ có thể chứa được địa chỉ của biến kiểu float); trong khi đó, Bien_Nguyen có kiểu int.

Nội dung của ô nhớ con trỏ chỉ tới

Để truy cập đến nội dung của ô nhớ mà con trỏ chỉ tới, ta sử dụng cú pháp:

*<Tên biến con trỏ>

Ví dụ 3: Ví dụ sau đây cho phép khai báo, gán địa chỉ cũng như lấy nội dung vùng nhớ của biến con trỏ:

```
int x=100;
int *ptr;
ptr=&x;
int y= *ptr;
```

Lưu ý: Khi gán địa chỉ của một biến cho một biến con trỏ, mọi sự thay đổi trên nội dung ô nhớ con trỏ chỉ tới sẽ làm giá trị của biến thay đổi theo (thực chất nội dung ô nhớ và biến chỉ là một).

Ví dụ 4: Đoạn chương trình sau thấy rõ sự thay đổi này :

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int a,b,*pa,*pb;
   a=2;
   b=3;
   printf("\nGia tri cua bien a=%d \nGia tri cua bien b=%d ",a,b);
   pa=&a;
   pb=&b;
   printf("\nNoi dung cua o nho con tro pa tro toi=%d",*pa);
   printf("\nNoi dung cua o nho con tro pb tro toi=%d ",*pb);
   *pa=20; /* Thay doi gia tri cua *pa*/
   *pb=20; /* Thay doi gia tri cua *pb*/
   printf("\nGia tri moi cua bien a=%d \n Gia tri moi cua bien b=%d ",a,b);
   getch();
   return 0;
}
```

Cấp phát vùng nhớ cho biến con trỏ

Trước khi sử dụng biến con trỏ, ta nên cấp phát vùng nhớ cho biến con trỏ này quản lý địa chỉ. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện alloc.h.

Cú pháp:

void *malloc(size_t size):

Cấp phát vùng nhớ có kích thước là size.

void *calloc(size_t nitems, size_t size):

Cấp phát vùng nhớ có kích thước là nitems*size.

Ví dụ: Giả sử ta có khai báo:

```
int a, *pa, *pb;
pa = (int*)malloc(sizeof(int));
/* Cấp phát vùng nhớ có kích thước bằng với kích thước của một số nguyên */
pb= (int*)calloc(10, sizeof(int));
/* Cấp phát vùng nhớ có thể chứa được 10 số nguyên*/
```

Lúc này hình ảnh trong bộ nhớ như sau:

<table>
<tr><td>0</td><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td colspan="2">pa (2 byte)</td><td colspan="10">pb (2 byte)</td></tr>
</table>

Lưu ý: Khi sử dụng hàm malloc() hay calloc(), ta phải ép kiểu vì nguyên mẫu các hàm này trả về con trỏ kiểu void.

Cấp phát lại vùng nhớ cho biến con trỏ

Trong quá trình thao tác trên biến con trỏ, nếu ta cần cấp phát thêm vùng nhớ có kích thước lớn hơn vùng nhớ đã cấp phát, ta sử dụng hàm realloc().

Cú pháp:

void *realloc(void *block, size_t size)

Ý nghĩa:

- Cấp phát lại 1 vùng nhớ cho con trỏ block quản lý, vùng nhớ này có kích thước mới là size; khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại;
- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ được chỉ ra khi cấp phát ban đầu.

Ví dụ: Trong ví dụ trên ta có thể cấp phát lại vùng nhớ do con trỏ pa quản lý như sau:

```
int a, *pa;
pa= (int*)malloc(sizeof(int)); /*Cấp phát vùng nhớ có kích thước 2 byte*/
pa = realloc(pa, 6); /* Cấp phát lại vùng nhớ có kích thước 6 byte*/
```

Giải phóng vùng nhớ cho biến con trỏ : Một vùng nhớ đã cấp phát cho biến con trỏ, khi không còn sử dụng nữa, ta sẽ thu hồi lại vùng nhớ này nhờ hàm free().

Cú pháp:

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

Ý nghĩa: Giải phóng vùng nhớ được quản lý bởi con trỏ block.

Ví dụ: Ở ví dụ trên, sau khi thực hiện xong, ta giải phóng vùng nhớ cho 2 biến con trỏ pa & pb:

```
free(pa);
free(pb);
```

Một số phép toán trên con trỏ

a. Phép gán con trỏ: Hai con trỏ cùng kiểu có thể gán cho nhau.

Ví dụ

```
int a, *p, *q ; float *f;
a = 5 ; p = &a ; q = p ; /* đúng */
f = p ; /* sai do khác kiểu */
```

Ta cũng có thể ép kiểu con trỏ theo cú pháp:

```
(<Kiểu kết quả>*)<Tên con trỏ>
```

Chẳng hạn, ví dụ trên được viết lại:

```
int a, *p, *q ; float *f;
```

```
a = 5 ; p = &a ; q = p ; /* đúng */
f = (float*)p; /* Đúng nhờ ép kiểu*/
```

b. Cộng, trừ con trỏ với một số nguyên

Ta có thể cộng (+), trừ (-) 1 con trỏ với 1 số nguyên N nào đó; kết quả trả về là 1 con trỏ. Con trỏ này chỉ đến vùng nhớ cách vùng nhớ của con trỏ hiện tại N phần tử.

Ví dụ: Cho đoạn chương trình sau:

```
int *pa;
pa = (int*) malloc(20); /* Cấp phát vùng nhớ 20 byte=10 số nguyên*/
int *pb, *pc;
pb = pa + 7;
pc = pb - 3;
```

Lúc này hình ảnh của pa, pb, pc như sau:

```
0  1  2  3  4  5  6  7  8  9
-------------------------------
pa          pc          pb
```

c. Con trỏ NULL: là con trỏ không chứa địa chỉ nào cả. Ta có thể gán giá trị NULL cho 1 con trỏ có kiểu bất kỳ.

d. Lưu ý:

- Ta không thể cộng 2 con trỏ với nhau;
- Phép trừ 2 con trỏ cùng kiểu sẽ trả về 1 giá trị nguyên (int). Đây chính là khoảng cách (số phần tử) giữa 2 con trỏ đó. Chẳng hạn, trong ví dụ trên pc-pa=4.

2. Con trỏ và mảng

Con trỏ và mảng 1 chiều

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

Truy cập các phần tử mảng theo dạng con trỏ

Ta có các quy tắc sau:

- &<Tên mảng>[0] tương đương với <Tên mảng>
- &<Tên mảng> [<Vị trí>] tương đương với <Tên mảng> + <Vị trí>
- <Tên mảng>[<Vị trí>] tương đương với *(<Tên mảng> + <Vị trí>)

Ví dụ 1: Cho 1 mảng 1 chiều các số nguyên a có 5 phần tử, truy cập các phần tử theo kiểu mảng và theo kiểu con trỏ.

```
#include <stdio.h>
#include <conio.h>
/* Nhap mang binh thuong */
void NhapMang(int a[], int N)
{
   int i;
   for(i=0;i<N;i++)
   {
      printf("Phan tu thu %d: ",i);scanf("%d",&a[i]);
   }
}
/* Nhap mang theo dang con tro */
void NhapContro(int a[], int N)
{
   int i;
   for(i=0;i<N;i++)
   {
      printf("Phan tu thu %d: ",i);scanf("%d",a+i);
   }
}
int main()
{
   int a[20],N,i;
   printf("So phan tu N= ");scanf("%d",&N);
   NhapMang(a,N);
   // NhapContro(a,N);
   printf("Truy cap theo kieu mang: ");
   for(i=0;i<N;i++)
      printf("%d ",a[i]);
   printf("\nTruy cap theo kieu con tro: ");
```

```
    for(i=0;i<N;i++)
        printf("%d ",*(a+i));
    getch();
    return 0;
}
```

Truy xuất từng phần tử đang được quản lý bởi con trỏ theo dạng mảng

- <Tên biến>[<Vị trí>] tương đương với *(<Tên biến> + <Vị trí>)
- &<Tên biến>[<Vị trí>] tương đương với (<Tên biến> + <Vị trí>)

Trong đó <Tên biến> là biến con trỏ, <Vị trí> là 1 biểu thức số nguyên.

Ví dụ 2: Giả sử có khai báo:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <stdlib.h>        /* Them vao so voi phien ban tren DOS*/
int main()
{
    int *a;
    int i;
    a=(int*)malloc(sizeof(int)*10);
    for(i=0;i<10;i++)
        a[i] = 2*i;
    printf("Truy cap theo kieu mang: ");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\nTruy cap theo kieu con tro: ");
    for(i=0;i<10;i++)
        printf("%d ",*(a+i));
    getch();
    return 0;
}
```

Con trỏ chỉ đến phần tử mảng

Giả sử con trỏ ptr chỉ đến phần tử a[i] nào đó của mảng a thì:

ptr + j chỉ đến phần tử thứ j sau a[i], tức a[i+j]

ptr - j chỉ đến phần tử đứng trước a[i], tức a[i-j]

Ví dụ 3: Giả sử có 1 mảng mang_int, cho con trỏ contro_int chỉ đến phần tử thứ 5 trong mảng. In ra các phần tử của contro_int & mang_int.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>                    /* Them vao so voi phien ban tren DOS*/
int main()
{
   int i,mang_int[10];
   int *contro_int;
   for(i=0;i<=9;i++)
      mang_int[i]=i*2;
   contro_int=&mang_int[5];
   printf("\nNoi dung cua mang_int ban dau=");
   for (i=0;i<=9;i++)
      printf("%d ",mang_int[i]);
   printf("\nNoi dung cua contro_int ban dau =");
   for (i=0;i<5;i++)
      printf("%d ",contro_int[i]);
   for(i=0;i<5;i++)
      contro_int[i]++;
   printf("\n--------------------");
   printf("\nNoi dung cua mang_int sau khi tang 1=");
   for (i=0;i<=9;i++)
      printf("%d ",mang_int[i]);
   printf("\nNoi dung cua contro_int sau khi tang 1=");
   for (i=0;i<5;i++)
      printf("%d ",contro_int[i]);
   if (contro_int!=NULL)
     free(contro_int);
   getch();
   return 0;
}
```

Con trỏ và mảng nhiều chiều

Ta có thể sử dụng con trỏ thay cho mảng nhiều chiều như sau: Giả sử ta có mảng 2 chiều và biến con trỏ như sau:

int a[n][m];

int *contro_int;

Thực hiện phép gán

contro_int=a;

khi đó phần tử a[0][0] được quản lý bởi contro_int;

a[0][1] được quản lý bởi contro_int+1;

a[0][2] được quản lý bởi contro_int+2;

...

a[1][0] được quản lý bởi contro_int+m;

a[1][1] được quản lý bởi contro_int+m+1;

...

a[n-1][m-1] được quản lý bởi contro_int+(n-1)*m + (m-1);

Tương tự như thế đối với mảng nhiều hơn 2 chiều.

Ví dụ 4: Sự tương đương giữa mảng 2 chiều và con trỏ.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
int main()
{
   int i,j;
   int mang_int[4][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,
   15,16,17,18,19,20};
   int *contro_int;
   contro_int=(int*)mang_int;
   printf("\nNoi dung cua mang_int ban dau=");
   for (i=0;i<4;i++)
   {
      printf("\n");
      for (j=0;j<5;j++)
            printf("%d\t",mang_int[i][j]);
   }
   printf("\n----------------------------");
   printf("\nNoi dung cua contro_int ban dau \n");
```

```
    for (i=0;i<20;i++)
        printf("%d ",contro_int[i]);
    for(i=0;i<20;i++)
        contro_int[i]++ ;
    printf("\n----------------------------");
    printf("\nNoi dung cua mang_int sau khi tang 1=");
    for (i=0;i<4;i++)
    {
        printf("\n");
        for (j=0;j<5;j++)
            printf("%d\t",mang_int[i][j]);
    }
    printf("\nNoi dung cua contro_int sau khi tang 1=\n");
    for (i=0;i<20;i++)
        printf("%d ",contro_int[i]);
    if (contro_int!=NULL)
        free(contro_int);
    getch();
    return 0;
}
```

3. Con trỏ và tham số hình thức của hàm

Khi tham số hình thức của hàm là một con trỏ thì theo nguyên tắc gọi hàm ta dùng tham số thực tế là 1 con trỏ có kiểu giống với kiểu của tham số hình thức. Nếu lúc thực thi hàm ta có sự thay đổi trên nội dung vùng nhớ được chỉ bởi con trỏ tham số hình thức thì lúc đó nội dung vùng nhớ được chỉ bởi tham số thực tế cũng sẽ bị thay đổi theo.

Ví dụ : Xét hàm hoán vị được viết như sau :

```
#include<stdio.h>
#include<conio.h>
void HoanVi(int *a, int *b)
{
    int c=*a;
    *a=*b;
    *b=c;
}
int main()
{
    int m=20,n=30;
```

```
    printf("Truoc khi goi ham m= %d, n= %d\n",m,n);
    HoanVi(&m,&n);
    printf("Sau khi goi ham m= %d, n= %d",m,n);
    getch();
    return 0;
}
```

IV. Tóm tắt nội dung bài học

I. Mảng
   1. Mảng trong C
   2. Mảng một chiều
   3. Mảng nhiều chiều

II. Con trỏ
   1. Khai báo và sử dụng biến con trỏ
   2. Con trỏ và mảng
   3. Con trỏ và tham số hình thức của hàm

V. Bài tập về nhà và hướng dẫn thực hành - MẢNG VÀ CON TRỎ

A. Bài tập làm theo yêu cầu

1. Tìm phần tử lớn nhất của mảng

Yêu cầu: Viết chương trình cho phép nhập vào một mảng, tìm phần tử lớn nhất và in ra màn hình

Soạn thảo văn bản chương trình như sau:

```
#include<conio.h>
#include<stdio.h>
int main()
{
    int b[20], N;
    int i, ln;
    printf("So phan tu thuc te cua mang N= ");
    scanf("%d",&N);
    for(i=0; i< N; i++)
    {
       printf("Phan tu thu %d: ",i);scanf("%d",&b[i]);
    }
    ln = b[0];
    for(i=1; i< N; i++)
    {
```

```
        if (b[i]>ln)
        {
            ln=b[i];
        }
    }
    printf("Gia tri lon nhat la: %d", ln);
    getch();
    return 0;
}
```

Thử nghiệm 1:

1. Nhận F9 để chạy chương trình, khi đó giao diện xuất hiện như sau:

2. Nhập các giá trị cho N (số phần tử của mảng) và các giá trị tương ứng cho mỗi phần tử của mảng như sau: 4 ↵ 8 ↵ 5 ↵ 9 ↵ 1 ↵, khi đó kết quả nhận được như sau:

3. Nhận xét về kết quả đạt được.

Thử nghiệm 2: Đổi việc tìm phần tử lớn nhất thành phần tử nhỏ nhất.

Thử nghiệm 3: Thêm vào yêu cầu là tìm phần tử lớn nhất và vị trí của phần tử đó trong mảng.

2. Tính ma trận tích

Vấn đề: Viết chương trình cho phép nhập vào mảng dữ liệu liệu hai chiều, tính tích hai ma trận và đưa ma trận kết quả ra màn hình.

Soạn thảo văn bản chương trình như sau:

```
#include <stdio.h>
#include <conio.h>
#define N 2
#define M 3
int main()
{
   int i, j,k;
        int a[N][M], b[M][N], c[N][N];
   //nhap du lieu cho mang a
   printf("Nhap ma tran A[%d][%d]:\n",N,M);
   for(i=0;i<N;i++)
   {
                for(j=0;j<M;j++)
                {
                        printf("a[%d][%d]:",i,j);
                        scanf("%d",&a[i][j]);
                }
   }
   //nhap du lieu cho mang b
   printf("Nhap ma tran b[%d][%d]:\n",M,N);
   for(i=0;i<M;i++)
   {
                for(j=0;j<N;j++)
                {
                        printf("b[%d][%d]:",i,j);
                        scanf("%d",&b[i][j]);
                }
   }
   //Nhan hai ma tran
   for(i=0;i<N;i++)
   {
                for(j=0;j<N;j++)
                {
                        c[i][j]=0;
                        for(k=0;k<M;k++)
                        {
                                c[i][j]+=a[i][k]*b[k][j];
                        }
```

```
            }
        }
        //ghi ra man hinh
        printf("Ma tran tich\n");
        for(i=0;i<N;i++)
        {
            for(j=0;j<N;j++)
            {
                printf("%5d",c[i][j]);
            }
            printf("\n");
        }
    getch();
    return 0;
}
```

Thử nghiệm 1:

Nhấn F9, nhập hai ma trận sau:

a

1	2	3
3	2	1

b

3	3
2	0
1	-1

- Hãy nhận xét về kết quả nhận được

Thử nghiệm 2:

Thay dòng 49 thành

```
//printf("\n");
```

Nhập lại giá trị ở trên, ghi nhận sự thay đổi kết quả in ra màn hình

Thử nghiệm 3:

Để nguyên dòng printf("\n");

Thay dòng 47 thành

```
printf("%d",c[i][j]);
```

Nhập lại giá trị ở trên, ghi nhận sự thay đổi kết quả in ra màn hình

Thử nghiệm 4:

- Giữ nguyên các dòng như văn bản ban đầu
- Thay đổi dòng 34 thành

```
//c[i][j]=0;
```

Thực hiện chương trình, nhập dữ liệu mảng ở trên, nhận xét kết quả in ra màn hình

Thử nghiệm 5:

- Giữ nguyên các dòng như văn bản ban đầu
- Thay đổi dòng 4 thành

```
#define M 2
```

Thực hiện chương trình, và nhận xét về số lượng phần tử nhập vào, và kết quả in ra màn hình

B. Bài tập tự làm

1. Viết chương trình nhập vào một dãy n số thực a[0], a[1],..., a[n-1], sắp xếp dãy số theo thứ tự giảm dần. In dãy số sau khi sắp xếp.
2. Viết chương trình sắp xếp một mảng theo thứ tự tăng dần sau khi đã loại bỏ các phần tử trùng nhau.
3. Viết chương trình nhập vào một mảng, hãy xuất ra màn hình:

   - Phần tử lớn nhất của mảng.

   - Phần tử nhỏ nhất của mảng.

   - Tính tổng của các phần tử trong mảng .
4. Viết chương trình nhập vào một dãy các số theo thứ tự tăng, nếu nhập sai quy cách thì yêu cầu nhập lại. In dãy số sau khi đã nhập xong. Nhập thêm một số mới và chèn số đó vào dãy đã có sao cho dãy vẫn đảm bảo thứ tự tăng. In lại dãy số để kiểm tra.
5. Viết chương trình nhập vào một ma trận (mảng hai chiều) các số nguyên, gồm m hàng, n cột. In ma trận đó lên màn hình. Nhập một số nguyên khác vào và xét xem có phần tử nào của ma trận trùng với số này không ? Ở vị trí nào ? Có bao nhiêu phần tử ?
6. Viết chương trình để chuyển đổi vị trí từ dòng thành cột của một ma trận (ma trận chuyển vị) vuông 4 hàng 4 cột. Sau đó viết cho ma trận tổng quát cấp m*n.
7. Viết chương trình nhập vào một mảng số tự nhiên. Hãy xuất ra màn hình:

   - Dòng 1 : gồm các số lẻ, tổng cộng có bao nhiêu số lẻ.

   - Dòng 2 : gồm các số chẵn, tổng cộng có bao nhiêu số chẵn.

   - Dòng 3 : gồm các số nguyên tố.

   - Dòng 4 : gồm các số không phải là số nguyên tố.
8. Viết chương trình tính tổng bình phương của các số âm trong một mảng các số nguyên.
9. Viết chương trình thực hiện việc đảo một mảng một chiều.

   Ví dụ : 1 2 3 4 5 7 9 10 đảo thành 10 9 7 5 4 3 2 1 .

   (Không dùng mảng phụ)
10. Viết chương trình nhập vào hai ma trận A và B có cấp m, n. In hai ma trận lên màn hình. Tổng hai ma trận A và B là ma trận C được tính bởi công thức:

    $c_{ij} = a_{ij} + b_{ij}$ (i=0,1,2,...m-1; j=0,1,2...n-1)

    Tính ma trận tổng C và in kết quả lên màn hình.

11. Viết chương trình nhập vào hai ma trận A có cấp m, k và B có cấp k, n. In hai ma trận lên màn hình. Tích hai ma trận A và B là ma trận C được tính bởi công thức:

$$c_{ij} = a_{i1} * b_{1j} + a_{i2} * b_{2j} + a_{i3} * b_{3j} + ... + a_{ik} * b_{kj} \quad (i=0,1,2,...m-1; j=0,1,2...n-1)$$

Tính ma trận tích C và in kết quả lên màn hình.

12. Xét ma trận A vuông cấp n, các phần tử a[i, i] (i= 1 ... n) được gọi là đường chéo chính của ma trận vuông A. Ma trận vuông A được gọi là ma trận tam giác nếu tất cả các phần tử dưới đường chéo chính đều bằng 0. Định thức của ma trận tam giác bằng tích các phần tử trên đường chéo chính. Hãy biến đổi ma trân A về ma trận tam giác. In kết quả từng bước lên màn hình.
13. Viết chương trình thực hiện việc trộn hai dãy có thứ tự thành một dãy có thứ tự. Yêu cầu không được trộn chung rồi mới sắp thứ tự. Khi trộn phải tận dụng được tính chất đã sắp của hai dãy con.
14. Thực hiện các bài tập ở phần *Mảng* , bằng cách sử dụng con trỏ.

Bài 6b - XÂU KÝ TỰ

I. Khai báo

- Chuỗi ký tự là một dãy gồm các ký tự hoặc một mảng các ký tự được kết thúc bằng ký tự '\0' (còn được gọi là ký tự NULL trong bảng mã ASCII).
- Các hằng chuỗi ký tự được đặt trong cặp dấu nháy kép "".

1. Khai báo theo mảng

Cú pháp:

char <Biến> [Chiều dài tối đa];

Ví dụ: Trong chương trình, ta có khai báo:

char Ten[12];

Trong khai báo này, bộ nhớ sẽ cung cấp 12+1 bytes để lưu trữ nội dung của chuỗi ký tự Ten; byte cuối cùng lưu trữ ký tự '\0' để chấm dứt chuỗi.

Lưu ý:

- Chiều dài tối đa của biến chuỗi là một hằng nguyên nằm trong khoảng từ 1 đến 255 bytes.
- Chiều dài tối đa không nên khai báo thừa để tránh lãng phí bộ nhớ, nhưng cũng không nên khai báo thiếu.

2. Khai báo theo con trỏ

Cú pháp:

char *<Biến>;

Ví dụ: Trong chương trình, ta có khai báo:

char *Ten;

Trong khai báo này, bộ nhớ sẽ dành 2 byte để lưu trữ địa chỉ của biến con trỏ Ten đang chỉ đến, chưa cung cấp nơi để lưu trữ dữ liệu. Muốn có chỗ để lưu trữ dữ liệu, ta phải gọi đến hàm malloc() hoặc calloc() có trong "alloc.h", sau đó mới gán dữ liệu cho biến.

3. Vừa khai báo vừa gán giá trị

Cú pháp:

char <Biến>[]=<"Hằng chuỗi">

Ví dụ:

```
#include<stdio.h>
#include<conio.h>
int main()
{
   char Chuoi[]="Mau nang hay la mau mat em" ;
   printf("Vua khai bao vua gan tri : %s",Chuoi) ;
   getch();
   return 0;
}
```

Lưu ý: Chuỗi được khai báo là một mảng các ký tự nên các thao tác trên mảng có thể áp dụng đối với chuỗi ký tự.

II. Nhập xuất chuỗi

1. Nhập chuỗi từ bàn phím

Để nhập một chuỗi ký tự từ bàn phím, ta sử dụng hàm gets()

Cú pháp:

gets(<Biến chuỗi>);

Ví dụ:
```
char Ten[20];
gets(Ten);
```

Ta cũng có thể sử dụng hàm scanf() để nhập dữ liệu cho biến chuỗi, tuy nhiên lúc này ta chỉ có thể nhập được một chuỗi không có dấu khoảng trắng.

Ngoài ra, hàm cgets() (trong conio.h) cũng được sử dụng để nhập chuỗi.

2. Xuất chuỗi lên màn hình

Để xuất một chuỗi (biểu thức chuỗi) lên màn hình, ta sử dụng hàm puts().

Cú pháp:

puts(<Biểu thức chuỗi>);

Ví dụ: Nhập vào một chuỗi và hiển thị trên màn hình chuỗi vừa nhập.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Ten[12];
```

```
    printf("Nhap chuoi: ");gets(Ten);
    printf("Chuoi vua nhap: ");puts(Ten);
    getch();
    return 0;
}
```

Ngoài ra, ta có thể sử dụng hàm printf(), cputs() (trong conio.h) để hiển thị chuỗi lên màn hình.

III. Một số hàm xử lý chuỗi

Các hàm này có trong string.h

1. Cộng chuỗi

- Hàm strcat()

Cú pháp:

```
char *strcat(char *des, const char *source)
```

Hàm này có tác dụng ghép chuỗi nguồn vào chuỗi đích.

Ví dụ: Nhập vào họ lót và tên của một người, sau đó in cả họ và tên của họ lên màn hình.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char HoLot[30], Ten[12];
   printf("Nhap Ho Lot: ");gets(HoLot);
   printf("Nhap Ten: ");gets(Ten);
   strcat(HoLot,Ten); /* Ghep Ten vao HoLot*/
   printf("Ho ten la: ");puts(HoLot);
   getch();
   return 0;
}
```

2. Xác định độ dài chuỗi

- Hàm strlen()

Cú pháp:

```
int strlen(const char* s)
```

Ví dụ: Sử dụng hàm strlen xác định độ dài một chuỗi nhập từ bàn phím.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char Chuoi[255];
   int Dodai;
   printf("Nhap chuoi: ");gets(Chuoi);
   Dodai = strlen(Chuoi);
   printf("Chuoi vua nhap: ");puts(Chuoi);
   printf("Co do dai %d",Dodai);
   getch();
   return 0;
}
```

3. Đổi một ký tự thường thành ký tự hoa

- Hàm toupper(): Hàm toupper() (trong ctype.h) được dùng để chuyển đổi một ký tự thường thành ký tự hoa.

Cú pháp:

```
char toupper(char c)
```

4. Đổi chuỗi chữ thường thành chuỗi chữ hoa

- Hàm strupr(): Hàm struppr() được dùng để chuyển đổi chuỗi chữ thường thành chuỗi chữ hoa, kết quả trả về của hàm là một con trỏ chỉ đến địa chỉ chuỗi được chuyển đổi.

Cú pháp:

```
char *strupr(char *s)
```

Ví dụ: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàm strupr() để chuyển đổi chúng thành chuỗi chữ hoa.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
```

```
    char Chuoi[255],*s;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    s=strupr(Chuoi) ;
    printf("Chuoi chu hoa: ");
    puts(s);
    getch();
    return 0;
}
```

5. Đổi chuỗi chữ hoa thành chuỗi chữ thường

- Hàm strlwr(): Muốn chuyển đổi chuỗi chữ hoa thành chuỗi toàn chữ thường, ta sử dụng hàm strlwr(), các tham số của hàm tương tự như hàm strupr().

Cú pháp:

```
char *strlwr(char *s)
```

6. Sao chép chuỗi

- Hàm strcpy(): Hàm này được dùng để sao chép toàn bộ nội dung của chuỗi nguồn vào chuỗi đích.

Cú pháp:

```
char *strcpy(char *Des, const char *Source)
```

Ví dụ: Viết chương trình cho phép chép toàn bộ chuỗi nguồn vào chuỗi đích.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char Chuoi[255],s[255];
   printf("Nhap chuoi: ");
   gets(Chuoi);
   strcpy(s,Chuoi);
   printf("Chuoi dich: ");
   puts(s);
   getch();
   return 0;
}
```

7. Sao chép một phần chuỗi

- Hàm strncpy(): Hàm này cho phép chép n ký tự đầu tiên của chuỗi nguồn sang chuỗi đích.

Cú pháp:

```
char *strncpy(char *Des, const char *Source, size_t n)
```

8. Trích một phần chuỗi

- Hàm strchr(): Để trích một chuỗi con của một chuỗi ký tự bắt đầu từ một ký tự được chỉ định trong chuỗi cho đến hết chuỗi, ta sử dụng hàm strchr().

Cú pháp :

```
char *strchr(const char *str, int c)
```

Lưu ý :

- Nếu ký tự đã chỉ định không có trong chuỗi, kết quả trả về là NULL ;
- Kết quả trả về của hàm là một con trỏ, con trỏ này chỉ đến ký tự c được tìm thấy đầu tiên trong chuỗi str.

9. Tìm kiếm nội dung chuỗi

- Hàm strstr(): Hàm strstr() được sử dụng để tìm kiếm sự xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1.

Cú pháp:

```
char *strstr(const char *s1, const char *s2)
```

Kết quả trả về của hàm là một con trỏ chỉ đến phần tử đầu tiên của chuỗi s1 có chứa chuỗi s2 hoặc giá trị NULL nếu chuỗi s2 không có trong chuỗi s1.

Ví dụ: Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi “hoc”.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char Chuoi[255],*s;
   printf("Nhap chuoi: ");
   gets(Chuoi);
```

```
    s=strstr(Chuoi,"hoc");
    printf("Chuoi trich ra: ");
    puts(s);
    getch();
    return 0;
}
```

10. So sánh chuỗi

- Hàm strcmp(): Để so sánh hai chuỗi theo từng ký tự trong bảng mã Ascii, ta có thể sử dụng hàm strcmp().

Cú pháp:

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

Hai chuỗi s1 và s2 được so sánh với nhau, kết quả trả về là một số nguyên (số này có được bằng cách lấy ký tự của s1 trừ ký tự của s2 tại vị trí đầu tiên xảy ra sự khác nhau).

- Nếu kết quả là số âm, chuỗi s1 nhỏ hơn chuỗi s2;
- Nếu kết quả là 0, hai chuỗi bằng nhau;
- Nếu kết quả là số dương, chuỗi s1 lớn hơn chuỗi s2.

11. So sánh chuỗi

- Hàm stricmp(): Hàm này thực hiện việc so sánh trong n ký tự đầu tiên của 2 chuỗi s1 và s2, giữa chữ thường và chữ hoa không phân biệt.

Cú pháp:

int stricmp(const char *s1, const char *s2)

Kết quả trả về tương tự như kết quả trả về của hàm strcmp()

12. Khởi tạo chuỗi

- Hàm memset(): Hàm này được sử dụng để đặt n ký tự đầu tiên của chuỗi là ký tự c.

Cú pháp:

memset(char *Des, int c, size_t n)

Đổi từ chuỗi ra số, hàm atoi(), atof(), atol() (trong stdlib.h): Để chuyển đổi chuỗi ra số, ta sử dụng các hàm trên.

Cú pháp :

int atoi(const char *s) : chuyển chuỗi thành số nguyên

long atol(const char *s) : chuyển chuỗi thành số nguyên dài

float atof(const char *s) : chuyển chuỗi thành số thực

Nếu chuyển đổi không thành công, kết quả trả về của các hàm là 0. Ngoài ra, thư viện string.h còn hỗ trợ các hàm xử lý chuỗi khác, ta có thể đọc thêm trong phần trợ giúp.

V. Tóm tắt nội dung bài học

I. Giới thiệu
II. Khai báo
III. Nhập/Xuất
IV. Các hàm xử lý

VI. Bài tập về nhà và hướng dẫn thực hành - XÂU KÝ TỰ

A. Bài tập làm theo yêu cầu

1. Tạo xâu chuyển động trên màn hình

Vấn đề: Hãy tạo một dòng chữ chuyển động trên màn hình
Soạn thảo đoạn chương trình sau:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX_LENTH 100
int main()
{
        char st[MAX_LENTH];
        char sttemp[MAX_LENTH];
        int length;
        strcpy(st,"Lop hoc lap trinh co ban ");
        length=strlen(st);
        printf("Chuoi truoc khi xu ly (%s)\n",st);
        strcpy(sttemp,&st[1]);
        sttemp[length-1]=st[0];
        sttemp[length]=0;
        strcpy(st,sttemp);
        printf("Chuoi sau khi xu ly (%s)\n",st);
   getch();
   return 0;
}
```

Thử nghiệm 1:

- Dự đoán mục tiêu chương trình
- Chạy thử chương trình.
- Chương trình thực hiện in ra hai dòng dữ liệu khác nhau như thế nào?

Thử nghiệm 2: Soạn thảo đoạn chương trình sau:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include<time.h>
#define MAX_LENTH 100
int main()
{
	int i=0;
	clock_t goal;
	system("cls");
	do
	{
		printf("%d",i);
		i++;
		goal= clock()+100;
		while (goal > clock());
		system("cls");
	} while (!kbhit());
	return 0;
}
```

- Chạy thử chương trình
- Chương trình tiến hành tạo vòng lặp đếm các số và in ra màn hình, mỗi vòng lặp dừng 100 mili giây trước khi in ra số tiếp theo.
- Thử bỏ lệnh system(“cls”); xem kết quả

Thử nghiệm 3:

- Phân tích ý nghĩa của

```
		goal= clock()+100;
		while (goal > clock());
```

trong đoạn chương trình ở trên, thử thay đổi giá trị 100 bởi giá trị khác xem hiệu ứng.

2. Lấy Tên từ họ tên người Việt

Yêu cầu: Giả sử Họ tên người Việt luôn có dạng “Họ Đệm Tên”, trong đó các phần Họ, Đệm, Tên luôn cách nhau 1 dấu cách; phần Đệm có thể có hoặc không hoặc có nhiều hơn 1 từ. Hãy viết chương trình cho phép nhập vào *hoten*, rồi in ra phần *Tên.*

Soạn thảo văn bản chương trình như sau

```
#include<conio.h>
```

```
#include<stdio.h>
#include<string.h>
int main()
{
   char Chuoi[255],s[255];
   int n,i,k;
   printf("Nhap chuoi: ");
   gets(Chuoi);
   n=strlen(Chuoi);
   for (i=n-1;i>-1;i--)
   {
      if (Chuoi[i]==32)
      {
        k=i;
        break;
      }
   }
   printf("ket qua\n");
   strncpy(s,Chuoi+k+1,n-k-1);
   puts(s);
   getch();
   return 0;
}
```

Thử nghiệm 1:

1. Nhấn F9 chạy thử chương trình, khi đó giao diện xuất hiện như hình sau:

2. Nhập giá trị cho chuỗi là Hoang Duc Van ↵, khi đó kết quả nhận được như sau:

3. Nhận xét về kết quả đạt được.

Thử nghiệm 2: Chạy và nhập chuỗi vào là "Hoang Duc Van " (có chứa 2 dấu cách ở cuối), nhận xét về kết quả đạt được.

Thử nghiệm 3: Chạy và nhập chuỗi vào là "HoangDucVan" (không có dấu cách), nhận xét về kết quả đạt được.

B. Bài tập tự làm

1. Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình mã Ascii của từng ký tự có trong chuỗi.
2. Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình chuỗi đảo ngược của chuỗi đó. Ví dụ đảo của "abcdx_egh" là "hge_xdcba".
3. Viết chương trình nhập một chuỗi ký tự và kiểm tra xem chuỗi đó có đối xứng không. Ví dụ : Chuỗi ABCDEDCBA là chuỗi đối xứng.
4. Nhập vào một chuỗi bất kỳ, hãy đếm số lần xuất hiện của mỗi loại ký tự.
    a. Ví dụ:
    b. Chuoi1[] = "abcdeaabbdca"
    c. Chuoi2[]="a b c d e"
    d. SLXH[] = 4,3,2,2,1
5. Viết chương trình nhập vào một chuỗi.
    a. In ra màn hình từ bên trái nhất và phần còn lại của chuỗi. Ví dụ: "Nguyễn Văn Minh" in ra thành:
        i. Nguyễn
        ii. Văn Minh
    b. In ra màn hình từ bên phải nhất và phần còn lại của chuỗi. Ví dụ: "Nguyễn Văn Minh" in ra thành:
        i. Minh
        ii. Nguyễn Văn

6. Viết chương trình nhập vào một chuỗi rồi xuất chuỗi đó ra màn hình dưới dạng mỗi từ một dòng. Ví dụ: "Nguyễn Văn Minh" in ra :
    i. Nguyễn
    ii. Văn
    iii. Minh
7. Viết chương trình nhập vào một chuỗi, in ra chuỗi đảo ngược của nó theo từng từ. Ví dụ : chuỗi "Nguyễn Văn Minh" đảo thành "Minh Văn Nguyễn"
8. Viết chương trình đổi số tiền từ số thành chữ. Ví dụ: 123 thành chữ là "mot tram hai muoi ba".
9. Viết chương trình nhập vào họ và tên của một người, cắt bỏ các khoảng trống không cần thiết (nếu có), tách tên ra khỏi họ và tên, in tên lên màn hình. Chú ý đến trường hợp cả họ và tên chỉ có một từ.
10. Viết chương trình nhập vào họ và tên của một người, cắt bỏ các khoảng trắng bên phải, trái và các khoảng trắng không có nghĩa trong chuỗi. In ra màn hình toàn bộ họ tên người đó dưới dạng chữ hoa, chữ thường.
11. Viết chương trình nhập vào một danh sách họ và tên của n người theo kiểu chữ thường, đổi các chữ cái đầu của họ, tên và chữ lót của mỗi người thành chữ hoa. In kết quả lên màn hình.
12. Viết chương trình nhập vào một danh sách họ và tên của n người, tách tên từng người ra khỏi họ và tên rồi sắp xếp danh sách tên theo thứ tự từ điển. In danh sách họ và tên sau khi đã sắp xếp.

Bài 6c - KIỂU BẢN GHI

1 Khái niệm

Vấn đề: Làm thế nào để mô tả một SINH VIÊN với các thông tin:

- Mã số sinh viên;
- Họ tên;
- Ngày tháng năm sinh;
- Giới tính;
- Địa chỉ thường trú

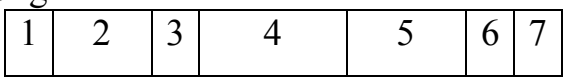
Hoặc làm thế nào để mô tả NGÀY THÁNG bao gồm các thông tin:

- Ngày;
- Tháng;
- Năm

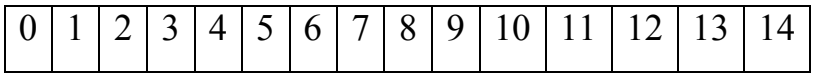
=> Hầu hết các ngôn ngữ lập trình trong đó có C/C++ cho phép người lập trình tự định nghĩa ra cấu trúc mới theo nhu cầu sử dụng từ những kiểu dữ liệu đã có hoặc đã định nghĩa trước đó.

Kiểu cấu trúc (Structure) là kiểu dữ liệu bao gồm nhiều thành phần có kiểu khác nhau, mỗi thành phần được gọi là một trường (field).

Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng là cùng kiểu còn các phần tử của kiểu cấu trúc có thể có kiểu khác nhau. Hình ảnh sau là của kiểu cấu trúc với 7 trường



còn kiểu mảng có dạng:



Cú pháp 1:

```
struct <Tên cấu trúc>
{
    <Kiểu> <Tên trường 1> ;
    <Kiểu> <Tên trường 2> ;
    ……..
    <Kiểu> <Tên trường n> ;
};
```

Cú pháp 2:

```
typedef struct
{
    <Kiểu> <Tên trường 1> ;
    <Kiểu> <Tên trường 2> ;
```

……..

<Kiểu> <Tên trường n> ;

} <Tên cấu trúc>;

Trong đó:

- <Tên cấu trúc>: là một tên được đặt theo quy tắc đặt tên của danh biểu; tên này mang ý nghĩa sẽ là tên kiểu cấu trúc;
- <Kiểu> <Trường i> (i=1..n): mỗi trường trong cấu trúc có dữ liệu thuộc kiểu gì (tên của trường phải là một tên được đặt theo quy tắc đặt tên của danh biểu).

Ví dụ 1: Để quản lý ngày, tháng, năm của một ngày trong năm ta có thể khai báo kiểu cấu trúc gồm 3 thông tin: ngày, tháng, năm.

```
struct KieuNgayThang
{
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
};
typedef struct
{
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
} KieuNgayThang;
```

Ví dụ 2: Mỗi sinh viên cần được quản lý bởi các thông tin: Mã số sinh viên, họ tên, ngày tháng năm sinh, giới tính, địa chỉ thường trú. Lúc này ta có thể khai báo một struct gồm các thông tin trên.

```
struct KieuSinhVien
{
char MSSV[10];
char HoTen[40];
struct KieuNgayThang NgaySinh;
int Phai;
char DiaChi[40];
};
```

```
typedef struct
{
char MSSV[10];
char HoTen[40];
KieuNgayThang NgaySinh;
int Phai;
char DiaChi[40];
} KieuSinhVien;
```

- Mỗi thành phần giống là một biến riêng thuộc cấu trúc, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.
- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy (;).
- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.
- Một biến có kiểu cấu trúc sẽ được cấp phát bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

2. Khai báo biến cấu trúc

Việc khai báo biến cấu trúc cũng tương tự như khai báo biến thuộc kiểu dữ liệu chuẩn.

Cú pháp:

- Đối với cấu trúc được định nghĩa theo cách 1:

  struct <Tên cấu trúc> <Biến 1> [, <Biến 2>…];

- Đối với các cấu trúc được định nghĩa theo cách 2:

  <Tên cấu trúc> <Biến 1> [, <Biến 2>…];

Ví dụ: Khai báo biến NgaySinh có kiểu cấu trúc KieuNgayThang; biến SV có kiểu cấu trúc KieuSinhVien.

```
struct KieuNgayThang NgaySinh;
struct KieuSinhVien SV;
KieuNgayThang NgaySinh;
KieuSinhVien SV;
```

3. Các thao tác trên biến kiểu cấu trúc

Truy xuất đến từng trường của biến cấu trúc

Cú pháp:

<Biến cấu trúc>.<Tên trường>;

Khi sử dụng cách truy xuất theo kiểu này, các thao tác trên <Biến cấu trúc>.<Tên trường> giống như các thao tác trên các biến của kiểu dữ liệu của <Tên trường>.

Ví dụ1: Viết chương trình cho phép đọc dữ liệu từ bàn phím cho biến mẩu tin SinhVien và in biến mẩu tin đó lên màn hình:

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
typedef struct
{
   unsigned char Ngay;
   unsigned char Thang;
   unsigned int Nam;
} KieuNgayThang;
typedef struct
{
   char MSSV[10];
   char HoTen[40];
   KieuNgayThang NgaySinh;
   int Phai;
   char DiaChi[40];
} KieuSinhVien;
/* Hàm in lên màn hình 1 m?u tin SinhVien*/

void InSV(KieuSinhVien s)
{
   printf("MSSV: | Ho va ten | Ngay Sinh | Dia chi\n");
   printf("%s | %s | %d-%d-%d | %s\n",s.MSSV,s.HoTen,
s.NgaySinh.Ngay,s.NgaySinh.Thang,s.NgaySinh.Nam,s.DiaChi);
}
int main()
{
   KieuSinhVien SV, s;
   printf("Nhap MSSV: ");gets(SV.MSSV);
   printf("Nhap Ho va ten: ");gets(SV.HoTen);
   printf("Sinh ngay: ");scanf("%d",&SV.NgaySinh.Ngay);
```

```
    printf("Thang: ");scanf("%d",&SV.NgaySinh.Thang);
    printf("Nam: ");scanf("%d",&SV.NgaySinh.Nam);
    printf("Gioi tinh (0: Nu), (1: Nam): ");scanf("%d",&SV.Phai);
    fflush(stdin);
    printf("Dia chi: ");gets(SV.DiaChi);
    InSV(SV);
    s=SV;
    InSV(s);
    getch();
    return 0;
}
```

Lưu ý:

- Các biến cấu trúc có thể gán cho nhau. Thực chất đây là thao tác trên toàn bộ cấu trúc không phải trên một trường riêng rẽ nào. Chương trình trên dòng s=SV là một ví dụ;
- Với các biến kiểu cấu trúc ta không thể thực hiện được các thao tác sau đây:
    - Sử dụng các hàm xuất nhập trên biến cấu trúc;
    - Các phép toán quan hệ, các phép toán số học và logic.

Ví dụ 2: Nhập vào hai số phức và tính tổng của chúng. Ta biết rằng số phức là một cặp (a,b) trong đó a, b là các số thực, a gọi là phần thực, b là phần ảo. (Đôi khi người ta cũng viết số phức dưới dạng a + ib trong đó i là một đơn vị ảo có tính chất i^2=-1). Gọi số phức c1=(a1, b1) và c2=(a2,b2) khi đó tổng của hai số phức c1 và c2 là một số phức c3 mà c3=(a1+a2, b1+b2). Với hiểu biết như vậy ta có thể xem mỗi số phức là một cấu trúc có hai trường, một trường biểu diễn cho phần thực, một trường biểu diễn cho phần ảo. Việc tính tổng của hai số phức được tính bằng cách lấy phần thực cộng với phần thực và phần ảo cộng với phần ảo.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
typedef struct
{
   float Thuc;
   float Ao;
```

```
} SoPhuc;
/* Ham in so phuc len man hinh*/
void InSoPhuc(SoPhuc p)
{
   printf("%.2f + i%.2f\n",p.Thuc,p.Ao);
}
int main()
{
   SoPhuc p1,p2,p;
   printf("Nhap so phuc thu nhat:\n");
   printf("Phan thuc: ");scanf("%f",&p1.Thuc);
   printf("Phan ao: ");scanf("%f",&p1.Ao);
   printf("Nhap so phuc thu hai:\n");
   printf("Phan thuc: ");scanf("%f",&p2.Thuc);
   printf("Phan ao: ");scanf("%f",&p2.Ao);
   printf("So phuc thu nhat: ");
   InSoPhuc(p1);
   printf("So phuc thu hai: ");
   InSoPhuc(p2);
   p.Thuc = p1.Thuc+p2.Thuc;
   p.Ao = p1.Ao + p2.Ao;
   printf("Tong 2 so phuc: ");
   InSoPhuc(p);
   getch();
   return 0;
}
```

Khởi tạo cấu trúc

Việc khởi tạo cấu trúc có thể được thực hiện trong lúc khai báo biến cấu trúc. Các trường của cấu trúc được khởi tạo được đạt giữa 2 dấu { và }, chúng được phân cách nhau bởi dấu phẩy (,).

Ví dụ: Khởi tạo biến cấu trúc NgaySinh:

```
struct KieuNgayThang NgaySinh ={29, 8, 1986};
```

4. Con trỏ cấu trúc

Khai báo

Việc khai báo một biến con trỏ kiểu cấu trúc cũng tương tự như khi khai báo một biến con trỏ khác, nghĩa là đặt thêm dấu * vào phía trước tên biến.

Cú pháp:

struct <Tên cấu trúc> * <Tên biến con trỏ>;

Ví dụ: Ta có thể khai báo một con trỏ cấu trúc kiểu NgayThang như sau:

```
struct NgayThang *p;
/* NgayThang *p; // Nếu có định nghĩa kiểu */
```

Sử dụng các con trỏ kiểu cấu trúc

Khi khai báo biến con trỏ cấu trúc, biến con trỏ chưa có địa chỉ cụ thể. Lúc này nó chỉ mới được cấp phát 2 byte để lưu giữ địa chỉ và được ghi nhận là con trỏ chỉ đến 1 cấu trúc, nhưng chưa chỉ đến 1 đối tượng cụ thể. Muốn thao tác trên con trỏ cấu trúc hợp lệ, cũng tương tự như các con trỏ khác, ta phải:

- Cấp phát một vùng nhớ cho nó (sử dụng hàm malloc() hay calloc);
- Hoặc, cho nó quản lý địa chỉ của một biến cấu trúc nào đó.

Ví dụ: Sau khi khởi tạo giá trị của cấu trúc:

```
struct NgayThang Ngay = {29,8,1986};
p = &Ngay;
```

lúc này biến con trỏ p đã chứa địa chỉ của Ngay.

Truy cập các thành phần của cấu trúc đang được quản lý bởi con trỏ

Để truy cập đến từng trường của 1 cấu trúc thông qua con trỏ của nó, ta sử dụng toán tử dấu mũi tên (->: dấu - và dấu >). Ngoài ra, ta vẫn có thể sử dụng đến phép toán * để truy cập vùng dữ liệu đang được quản lý bởi con trỏ cấu trúc để lấy thông tin cần thiết.

Ví dụ: Sử dụng con trỏ cấu trúc.

```
#include<conio.h>
#include<stdio.h>
typedef struct
{
   unsigned char Ngay;
   unsigned char Thang;
   unsigned int Nam;
} NgayThang;
int main()
{
   NgayThang Ng={29,8,1986};
```

```
    NgayThang *p;
    p=&Ng;
    printf("Truy cap binh thuong %d-%d-%d\n",
    Ng.Ngay,Ng.Thang,Ng.Nam);
    printf("Truy cap qua con tro %d-%d-%d\n",
    p->Ngay,p->Thang,p->Nam);
    printf("Truy cap qua vung nho con tro %d-%d-%d\n",
    (*p).Ngay,(*p).Thang,(*p).Nam);
    getch();
    return 0;
}
```

- Cho phép sử dụng cấu trúc, con trỏ cấu trúc là đối số của hàm như các loại biến khác
- Cho phép hàm xây dựng trả về là kiểu cấu trúc

Ví dụ : Xử lý danh sách sinh viên, sử dụng hàm với đối số là cấu trúc

```
#include <stdio.h>
#include <conio.h>
//Ngay thang
struct Ngaythang {
int ng ;
int th ;
int nam ;
};
//Sinh vien
struct Sinhvien {
char hoten[25] ;
Ngaythang ns;
int gt;
float diem ;
} ;
//cac ham xu ly
int sua(Sinhvien &r) ;
int in(Sinhvien x) ;
int nhap(Sinhvien *p) ;
int nhapds(Sinhvien *a, int n) ;
int suads(Sinhvien *a, int n) ;
int inds(const Sinhvien *a, int n) ;
//
```

```
struct Sinhvien a[10];

int main()
{
   nhap(a);
//  in(a[1]);
//  sua(a[1]);
   nhapds(a,9);
   suads(a,9);
   inds(a,9);
   getch();
   return 0;
}
///trien khai cac ham
int sua(Sinhvien &r)
{
 int chon;
 do {
   printf("1: Sua ho ten\n2: Sua ngay sinh\n3:Sua gioi tinh\n4:Sua diem\n5:In\n0: Ket thuc\n");
   scanf("%d",&chon);
   switch (chon)
   {
        case 1:
             printf("Nhap ho ten:");
             scanf("%s",r.hoten);
             break;
        case 2:
             printf("Nhap ngay thang nam sinh:");
             scanf("%d%d%d",&r.ns.ng,&r.ns.th,&r.ns.nam);
             break;
        case 3:
             printf("Nhap gioi tinh 0:Nu 1:Nam:");
             scanf("%d",&r.gt) ;
             break;
        case 4:
             printf("Nhap diem:");
             scanf("%f",&r.diem);
```

```
                break;
            case 5:
                printf("Sinh vien:");
                in(r);
                break;
            case 0:
                break;
            default:
                    printf("Nhap gia tri khong dung\n");
                    break;
        }
     } while (chon) ;
     return 0;
    }
    ////////
    int in(Sinhvien x)
    {
    printf("Ho ten :%s\nNgay sinh %d/%d/%d\n",x.hoten,x.ns.ng, x.ns.th,
x.ns.nam) ;
    printf("Gioi tinh :%s\nDiem :%f\n",(x.gt==1) ?"Nam" :"Nu",x.diem) ;
    return 0;
    }

    ///////
    int nhap(Sinhvien *p)
    {
            printf("Nhap ho va ten: ");
            scanf("%s",p->hoten);
            printf("Nhap ngay sinh (ngay thang nam): ");
            scanf("%d%d%d", &p->ns.ng ,&p->ns.th,&p->ns.nam);
            printf("Gioi tinh 0: nu, 1: nam: ");
            scanf("%d",& p->gt);
            printf("Diem: ");
            scanf("%f",& p->diem);
            return 0;
    }
    //////
    int nhapds(Sinhvien *a, int n)
```

```
{

 for (int i=1; i<=n; i++) nhap(&a[i]) ;
 return 0;
}
/////
int suads(Sinhvien *a, int n)
{
    int chon;
    do
    {
       printf("\nNhap phan tu duoc su tu 1 den %d, gia tri khac thoat:",n);
       scanf("%d",&chon);
       if(chon>0 &&chon<=n)
       {
           sua(a[chon]) ;
       }
    }while(chon>0 && chon<=n);
    return 0;
}
//////////
int inds(const Sinhvien *a, int n)
{
   for (int i=1; i<=n; i++)
      in(a[i]) ;
   return 0;
}
```

5. Cấu trúc với thành phần kiểu bit

a. Trường bit

Để tiết kiệm trong lưu trữ, trong ngôn ngữ lập trình C cho phép khai báo các trường của cấu trúc với số lượng bit xác định không phụ thuộc vào số lượng bit các kiểu dữ liệu chuẩn.

Một trường bit là một khai báo trường int và thêm dấu: cùng số bit n theo sau, trong đó $0 \leq n < 15$. Ví dụ do độ lớn của ngày không vượt quá 31, tháng không vượt quá 12 nên 2 trường này trong cấu trúc ngày tháng có thể khai báo tiết kiệm hơn bằng 5 và 4 bit như sau:

```
struct Date {
```

```
int ng: 5;
int th: 4;
int nam:14;
} ;
```

b. Đặc điểm

Cần chú ý các đặc điểm sau của một cấu trúc có chứa trường bit:

− Các bit được bố trí liên tục trên dãy các byte.

− Kiểu trường bit phải là int (signed hoặc unsigned).

− Độ dài mỗi trường bit không quá 16 bit.

− Có thể bỏ qua một số bit nếu bỏ trống tên trường, ví dụ:

```
struct tu {
    int: 8;
    int x:8;
}
```

mỗi một biến cấu trúc theo khai báo trên gồm 2 byte, bỏ qua không sử dụng byte thấp và trường x chiếm byte (8 bit) cao.

− Không cho phép lấy địa chỉ của thành phần kiểu bit.

− Không thể xây dựng được mảng kiểu bit.

− Không được trả về từ hàm một thành phần kiểu bit. Ví dụ nếu b là một thành phần của biến cấu trúc x có kiểu bit thì câu lệnh sau là sai:

```
return x.b ;    // sai
```

tuy nhiên có thể thông qua biến phụ như sau:

```
int tam = x.b ; return tam ;
```

− Tiết kiệm bộ nhớ

− Dùng trong kiểu union để lấy các bit của một từ (xem ví dụ trong phần kiểu hợp).

6. Câu lệnh typedef

Để thuận tiện trong sử dụng, thông thường các kiểu được NSD tạo mới sẽ được gán cho một tên kiểu bằng câu lệnh typedef như sau:

```
typedef <kiểu> <tên_kiểu> ;
```

Ví dụ: Để tạo kiểu mới có tên Bool và chỉ chứa giá trị nguyên (thực chất chỉ cần 2 giá trị 0, 1), ta có thể khai báo:

```
typedef int Bool;
```

khai báo này cho phép xem Bool như kiểu số nguyên.

hoặc có thể đặt tên cho kiểu ngày tháng là Date với khai báo sau:

```
typedef struct Date {
 int ng;
 int th;
 int nam;
};
```

khi đó ta có thể sử dụng các tên kiểu này trong các khai báo (ví dụ tên kiểu của đối, của giá trị hàm trả lại …).

7. Hàm sizeof()

Hàm trả lại kích thước của một biến hoặc kiểu. Ví dụ:

```
Bool a, b;
Date x, y, z[50];
printf("%d,%d,%d",sizeof(a),sizeof(b),sizeof(Bool)) ;     // in 2,2,2
printf("Số phần tử của z =%d ",sizeof(z) / sizeof(Date));  // in 50
```

V. Bài tập về nhà và hướng dẫn thực hành - KIỂU BẢN GHI

A. Bài tập làm theo yêu cầu

1. Biểu diễn và thực hiện các phép toán phân số

Yêu cầu: Xây dựng cấu trúc biểu diễn phân số; cho phép nhập vào 2 phân số, thực hiện các phép cộng, trừ, nhân, chia 2 phân số đó.

Soạn thảo văn bản chương trình như sau

```
#include <stdio.h>
#include <conio.h>

typedef struct fraction
{
   int numerator;
   int denominator;
};
///-- khai bao cac ham
fraction add(fraction x, fraction y) ;
fraction sub(fraction x, fraction y) ;
fraction multiply(fraction x, fraction y) ;
fraction divide(fraction x, fraction y) ;
void printFraction(fraction x);
```

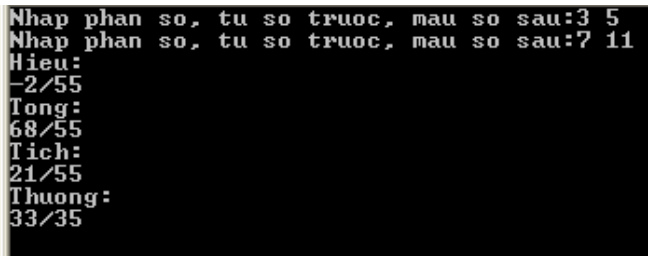
```
int main()
{
   fraction a,b;

   printf("Nhap phan so, tu so truoc, mau so sau:");
   scanf("%d%d",&a.numerator, &a.denominator);
   printf("Nhap phan so, tu so truoc, mau so sau:");
   scanf("%d%d",&b.numerator, &b.denominator);
   printf("Hieu:\n");
   printFraction(sub(a,b));
   printf("\nTong:\n");
   printFraction(add(a,b));
   printf("\nTich:\n");
   printFraction(multiply(a,b));
   printf("\nThuong:\n");
   printFraction(divide(a,b));
   getch();
   return 0;
}
///--------------- trien khai cac ham
fraction add(fraction x, fraction y)
{
   fraction t;
   t.numerator=x.numerator * y.denominator + x.denominator *
y.numerator;
   t.denominator = x.denominator * y.denominator;
   return t;
}
///------------------
fraction sub(fraction x, fraction y)
{
   fraction t;
   t.numerator=x.numerator * y.denominator - y.numerator *
x.denominator;
   t.denominator = x.denominator * y.denominator;
   return t;

}
```

```
fraction multiply(fraction x, fraction y)
{
    fraction t;
    t.numerator=x.numerator * y.numerator;
    t.denominator = x.denominator * y.denominator;
    return t;
}
fraction divide(fraction x, fraction y)
{
    fraction t;
    t.numerator=x.numerator * y.denominator;
    t.denominator = x.denominator * y.numerator;
    return t;
}
void printFraction(fraction x)
{
    printf("%d/%d",x.numerator,x.denominator);
}
```

Thử nghiệm 1:



Thử nghiệm 2:

Thử nghiệm với hai giá trị là 3/4, và 1/2 đưa ra nhận xét, cần cải tiến như thế nào?

Thử nghiệm 3:

Thử nghiệm với hai giá trị là 3/4, và 0/2 đưa ra nhận xét, cần cải tiến như thế nào?

B. Bài tập tự làm

1. Hãy định nghĩa kiểu:

```
struct Hoso{
char HoTen[40];
float Diem;
```

```
char Loai[10];
};
```

Viết chương trình nhập vào họ tên, điểm của n học sinh. Xếp loại văn hóa theo cách sau:

Điểm Xếp loại

9, 10 Giỏi

7, 8 Khá

5, 6 Trung bình

dưới 5 Không đạt

In danh sách lên màn hình theo dạng sau:

XEP LOAI VAN HOA

HO VA TEN	DIEM	XEPLOAI
Nguyen Van A	7	Kha
Ho Thi B	5	Trung binh
Dang Kim C	4	Khong dat

.....

2. Xem một phân số là một cấu trúc có hai trường là tử số và mẫu số. Hãy viết chương trình thực hiện các phép toán cộng, trừ, nhân, chia hai phân số. (Các kết quả phải tối giản).
3. Tạo một danh sách cán bộ công nhân viên, mỗi người người xem như một cấu trúc bao gồm các trường Ho, Ten, Luong, Tuoi, Dchi. Nhập một số người vào danh sách, sắp xếp tên theo thứ tự từ điển, in danh sách đã sắp xếp theo mẫu sau:

DANH SACH CAN BO CONG NHAN VIEN

STT	HO VA TEN	LUONG	TUOI	DIACHI
1	Nguyen Van	333.00	26	Can Tho
2	Dang Kim B	290.00	23	Vinh Long