Nội dung chính

I Địa chỉ và Con trỏ

II Mảng

III Cấp phát bộ nhớ động

4 Bài tập


Nội dung

I **Địa chỉ và Con trỏ**

1. Địa chỉ

- Địa chỉ của một biến là *số thứ tự* của *byte đầu tiên* trong một dãy các byte liên tiếp mà máy tính dành cho biến đó khi nó được khai báo.

- Để lấy địa chỉ của một biến, dùng phép toán: &

  cú pháp: &<tên_biến>

- Ví dụ:

  int *x*;

  phép toán &*x* cho ta địa chỉ của biến *x*

Ví dụ về phép toán lấy địa chỉ của biến x

```
#include<stdio.h>
main()
{
    int x;
    printf("Dia chi cua x trong bo nho la: %d\n", &x);
}
```

2. Con trỏ

- **Con trỏ** là một biến dùng để lưu trữ địa chỉ của một biến khác. Mỗi kiểu dữ liệu thì có kiểu con trỏ tương ứng.

- Cú pháp khai báo con trỏ: <*kiểu*> *<tên_con_trỏ>

  int *x; khai báo con trỏ *x*, có kiểu *nguyên*
  float *y; khai báo con trỏ *y*, có kiểu *thực*

- *Sau khi khai báo con trỏ*, muốn con trỏ đó trỏ đến một biến cụ thể, ta dùng phép *gán* để *gán* địa chỉ của biến đó cho con trỏ. Cú pháp gán như sau:

  <tên_con_trỏ> = &<tên_biến>

- *Sau khi gán địa chỉ của biến cho con trỏ*, chúng ta có thể *làm việc với vùng nhớ của biến* thông qua *con trỏ* đó. Cụ thể hơn: lúc này *biến* và *con trỏ* cùng làm việc trên một vùng nhớ.

2. Con trỏ (tiếp)

- Nếu con trỏ *trỏ tới* một biến cụ thể, mô tả của con trỏ cho ta một số ý nghĩa. Xét ví dụ sau.
- Ví dụ:
  - Khai báo:

```
float   x =5.1 ;  // khai báo biến x và khởi tạo cho nó giá trị là 5.1
float   *px ;     // khai báo con trỏ px
px = &x;                                  (1)
```

  - Biểu thức (1) có hai cách nói:
    - con trỏ *px* trỏ tới *x*
    - gán địa chỉ của *x* cho con trỏ *px*
  - Sau khi trỏ con trỏ *px* tới *x*, thì:

px	cho ta địa chỉ của biến *x* trong bộ nhớ
**px*	cho ta dữ liệu của biến *x*, hay **px* ≡ 5.1

- Nói chung, sau câu lệnh (1) thì *px* và *x* cùng *thao tác* trên vùng nhớ chung. Có thể hiểu rằng, lúc này *px* đại diện cho *x*.

2. Con trỏ (tiếp)

- Để làm việc với biến thông qua một con trỏ nào đó, tổng quát là:
  1. Trỏ con trỏ tới biến: <ten_con_tro> = &<ten_bien>
  2. Lấy *địa chỉ* của biến qua con trỏ: <ten_con_tro>
  3. Lấy *dữ liệu* của biến qua con trỏ: *<ten_con_tro>

Ví dụ về con trỏ

```
#include<stdio.h>
main()
{
        float x;
        float *a;
        printf("Nhap vao gia tri ban dau cua x: ");
        scanf("%f", &x);
        a = &x; // tro con tro a vao bien x;
        *a = 5;   //thay gia tri 5 vao vung nho ma a tro den
        printf("Gia tri cua x sau khi bi con tro a doi la: %f\n", x);
        printf("Du lieu tai vung nho ma a tro den la: %f\n", *a);
}
```

Đổi giá trị của biến x thông qua con trỏ a

2.a. Phép toán trên con trỏ

- **Phép gán**

  Ví dụ: *int* x; *char* *p;

  p=(char*)(&x);

- **Phép so sánh**

  Ví dụ: p1 < p2 , tức là địa chỉ p1 trỏ tới thấp hơn địa chỉ p2 trỏ tới

- **Phép tăng(+), giảm(-)**

  Ví dụ: *short int* x[30], *p; //khai báo con trỏ *p* và mảng *x*

  p = &x[10]; //Con trỏ *p* trỏ tới phần tử *x*[10]

  suy ra:

  p + i trỏ tới phần tử x[10+i]

  p – i trỏ tới phần tử x[10-i]

Lưu ý: Phép toán + và – trên con trỏ không phải là phép toán số học thông thường. Nó là phép dịch chuyển con trỏ tới các vùng địa chỉ khác.

2.b. Ứng dụng của con trỏ

- Đại diện cho các biến khác
- Quản lý mảng
- Quản lý vùng nhớ được cấp bởi các hàm cấp phát động (dynamic memory allocation).
- Quản lý file
- Đưa địa chỉ vùng nhớ của các biến bên ngoài hàm vào bên trong hàm *để lấy các giá trị tính toán được* ra bên ngoài hàm.

❖ …


Nội dung

II **Mảng**

1. Mảng

- **Mảng** là một tập hợp các phần tử có cùng kiểu dữ liệu, chung một tên gọi.

- Khi làm việc với *một tập hợp các phần tử cùng kiểu* ta thường dùng biến mảng, trước khi khai báo biến mảng phải xác định:
  - *Kiểu dữ liệu* của các phần tử trong mảng
  - *Tên* mảng
  - *Số chiều* và *kích thước* của mỗi chiều của mảng
- Để truy cập tới các phần tử trong mảng ta dùng chỉ số chỉ vị trí phần tử.
- Vùng nhớ dành cho các phần tử được cấp phát nằm liên tiếp nhau trong bộ nhớ.

2. Khai báo mảng

- Có hai cách chủ yếu để khai báo biến mảng
  - Khai báo "tĩnh"

    Xem mục: **2.a**, **2.b**, **2.c**
  - Khai báo "động" (cấp phát động bộ nhớ để lưu trữ mảng)

    Xem mục: **cấp phát bộ nhớ động**

2.a. Mảng 1 chiều

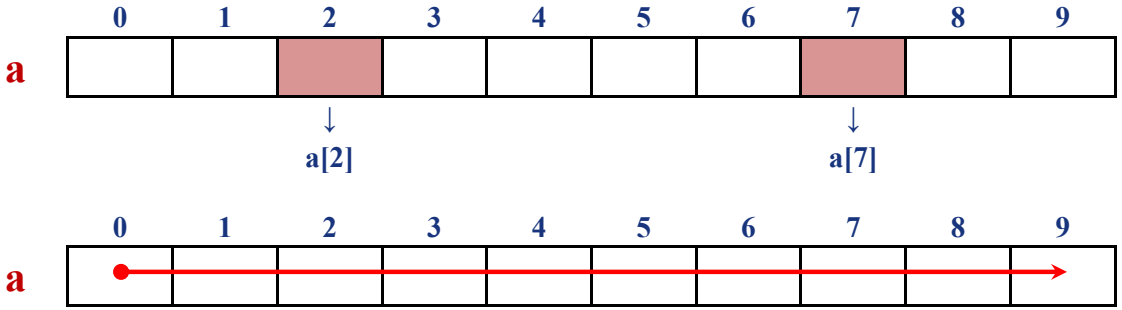
- Cú pháp: <kiểu> tên_mảng[tổng_phần_tử_của_mảng]

Ví dụ: khai báo mảng một chiều a, lưu trữ 10 số nguyên

```
int a[10];
```

- Phần tử tổng quát của mảng 1 chiều là: a[i]

- Thứ tự sắp xếp của các phần tử trong mảng một chiều *a* là:



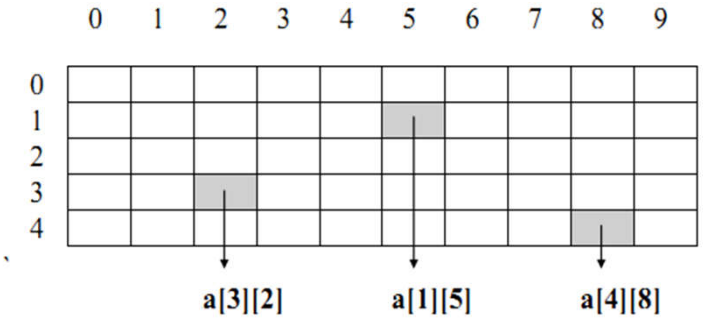
2.b. Khai báo mảng 2 chiều

- Cú pháp:

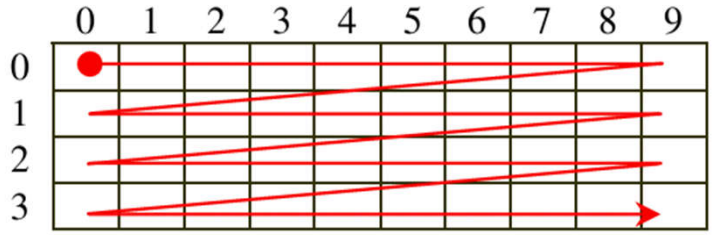
  <kiểu> tên_mảng[tổng_số_hàng][tổng_số_cột]

  Ví dụ: float **a**[5][10];

- Phần tử tổng quát của mảng hai chiều
  - ✓ Vị trí của các phần tử trong mảng hai chiều được xác định thông qua *chỉ số hàng* và *chỉ số cột*: a[i][j]
- Mô hình toán cho mảng 2 chiều và thứ tự truy xuất các phần tử
  - ✓ *Mô hình sắp xếp mảng a có 5 hàng, 10 cột*



  - ✓ *Thứ tự truy xuất các phần tử trong mảng*



2.c. Ví dụ về khai báo mảng tĩnh

- Khai báo biến mảng không khởi tạo giá trị
  - *int* a[5], c[4];
  - *float* b[3][4];
- Vừa khai báo vừa khởi tạo giá trị cho các phần tử
  - *int* a[3]={2, -3, 5}; hoặc: *int* a[]={2, -3, 5};
  - *float* b[4]={1, -4, 3, -4}; hoặc: *float* b[]={1, -4, 3, -4};
  - *float* c[2][2]={3, 2, -5, 4}; hoặc: *float* c[2][2]={{3, 2}, {-5, 4}};
  - *int* d[3][2] ={2, 4, 4, -6, 2, 3};

    hoặc: *int* d[3][2] ={{2, 4},{4, -6},{2, 3}};

3. Một số chú ý khi làm việc trên mảng

- *Tên* của mảng cho ta *địa chỉ* của phần tử đầu tiên trong mảng, nghĩa là:

  $a \equiv \&a[0]$

- Trong mảng 1 chiều, chỉ số mảng bắt đầu từ *0*, chỉ số cuối bằng *tổng số phần tử - 1*

- Sau câu lệnh khai báo, máy tính sẽ cấp bộ nhớ cho từng phần tử của mảng. *Bộ nhớ* đó được sắp xếp *liên tiếp* nhau.

- Đối với mảng tĩnh, sau câu lệnh khai báo mảng, bộ nhớ mà máy tính dành cho mảng tồn tại đến khi chương trình kết thúc.

- Nếu dùng chỉ số vượt qua kích thước thật đã được khai báo có thể gây ra lỗi khi chạy chương trình.

Ví dụ: Nhập dữ liệu cho mảng 1 chiều

```
# include <stdio.h>

main()
{
    int a[50], i,n;
    puts("Nhap n");
    scanf("%d",&n);
    for(i = 0;i < n;i++)
    {
        printf("Phan tu thu %d= ",i);
        scanf("%d", &a[i]);
    }

    for(i=0;i< n;i++)
        printf("%d\t",a[i])

}
```

Khai báo mảng a có 50 phần tử

Nhập từng phần tử mảng từ bàn phím

In ra các phần tử của mảng

Ví dụ: Trung bình cộng n số nguyên

```
# include <stdio.h>
int main()
{
    int  i, n, sum = 0, a[50];
    printf("Nhap vao gia tri n: ");
    scanf("%d", &n);                    // n<50
    for(i = 0; i < n; i++)
    {
       printf("Nhap vao phan tu thu %d: ", i + 1);
       scanf("%d", &a[i]);
       sum = sum + a[i];
    }
    printf("Trung binh cong: %f \n", (float)sum/n);
    return 0;
}
```

Khai báo mảng a có 50 phần tử

Nhập từng phần tử của mảng

Tính tổng các phần tử

Tính trung bình cộng

Ví dụ: Nhập dữ liệu cho mảng 2 chiều

```
# include <stdio.h>
main()
{
    int   i, j, m, n, a[50][50];
    puts("Nhap m, n");
    scanf("%d%d", &m, &n);
    for(i = 0; i < m; i++)
    {
        for(j = 0;j < n;j++)
            {
             printf("Phan tu a[%d][%d]= ", i, j);
             scanf("%d", &a[i][j]);
            }
    }

    for(i=0;i<m;i++)
    {
       for(j=0; j<n;j++)
            printf("%d  ",a[i][j]);
       printf("\n");
    }
}
```

Khai báo mảng a có 50 x 50 phần tử

Nhập giá trị cho các phần tử của mảng

In ra các phần tử của mảng dạng *hàng - cột*

4. Con trỏ với mảng 1 chiều

- Các phần tử của **mảng một chiều** có thể được xác định thông qua con trỏ.
  - Dùng trực tiếp tên mảng để truy xuất phần tử

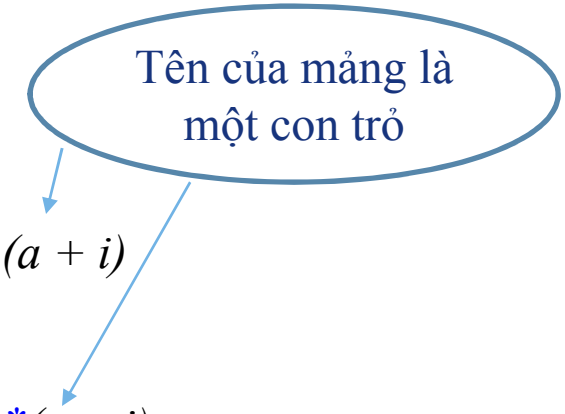
    Ví dụ: *float a[10];*

    ✓ Lấy địa chỉ của phần tử thứ i trong mảng:

    &a[i] hoặc *(a + i)*

    ✓ Lấy giá trị của phần tử thứ i trong mảng:

    a[i] hoặc **(a + i)*



  - Dùng một con trỏ khác để truy xuất phần tử trong mảng

    Ví dụ: *float a[10], *p ;* // khai báo mảng a, con trỏ p

    p = a; // trỏ p tới a chú ý dòng lệnh này

    ✓ Lấy địa chỉ của phần tử thứ i trong mảng a:

    &a[i] hoặc *(a + i)* hoặc *(p + i)*

    ✓ Lấy dữ liệu của phần tử thứ i trong mảng a:

    a[i] hoặc **(a + i)* hoặc **(p + i)* *hoặc* *p[i]*

Truy xuất các phần tử trong mảng 1 chiều

Chỉ số của phần tử:	0	1	2	i	n
Lấy ĐỊA CHỈ của phần tử trong mảng					
kiểu viết cơ bản:	&a[0]	&a[1]	&a[2]	&a[i]	&a[n]
qua tên mảng như một con trỏ:	(a + 0)	(a + 1)	(a + 2)	(a + i)	(a + n)
dùng con trỏ p (p = a):	(p + 0)	(p + 1)	(p + 2)	(p + i)	(p + n)
Lấy DỮ LIỆU của phần tử trong mảng					
Kiểu viết cơ bản:	a[0]	a[1]	a[2]	a[i]	a[n]
qua tên mảng (như 1 contrỏ):	*(a + 0)	*(a + 1)	*(a + 2)	*(a + i)	*(a + n)
dùng con trỏ p (p=a):	*(p + 0)	*(p + 1)	*(p + 2)	*(p + i) p[i]	*(p + n)

Con trỏ với mảng 2 chiều

Xét ví dụ:

- Khai báo mảng 2 chiều có 3 hàng, 2 cột chứa các số nguyên. In ra màn hình địa chỉ các phần tử của mảng.

Con trỏ với mảng 2 chiều

```
#include<stdio.h>
main()
{
        int i, j, a[3][2];
        printf("Dia chi cac phan tu:\n");
        for(i = 0; i < 3; i++)
        {
                for( j = 0; j < 2 ; j++)
                {
                        printf("%d
",&a[i][j]);
                }
                printf("\n");
        }
}
```

Địa chỉ này phụ thuộc vào từng máy khi chạy chương trình

Kết quả chạy chương trình:

```
Dia chi cac phan tu:
6487552 6487556
6487560 6487564
6487568 6487572
```

Quan sát địa chỉ của các phần tử in ra trên màn hình, ta thấy:

- Mỗi phần tử *int* chiếm 4 byte, chúng được sắp xếp liền kề nhau trong bộ nhớ.
- Các hàng của mảng 2 chiều được sắp xếp liền nhau, hàng sau được nối tiếp vào hàng trước.

Con trỏ với mảng 2 chiều

- Trong bộ nhớ máy tính, các hàng của mảng 2 chiều được bố trí xếp liền kề nhau, hàng sau nối tiếp vào hàng trước. Do đó có thể dùng mảng một chiều để lưu trữ mảng 2 chiều.
  - Ví dụ: *int a[7][5]* ; //mảng a có 7 hàng, **5** cột, có 7x5 = 35 phần tử
  - Ta có thể khai báo mảng trên dưới dạng mảng 1 chiều

int a[35]. Khi đó, phần tử *a[i][j]* được viết thành: *a[i*5 + j]*

Tổng quát:

- Với mảng 2 chiều có *m* hàng *n* cột, thì phần tử *a[i][j]* tương đương với biểu diễn *a[i*n+j]* trong mảng 1 chiều có 1 hàng, *m x n* cột

Truy xuất các phần tử trong mảng 2 chiều dùng con trỏ

- Mảng 2 chiều *a* ban đầu có *m* hàng, *n* cột, nếu chuyển sang dùng mảng 1 chiều *b* để lưu trữ thì mảng *b* này có 1 hàng, *m* x *n* cột. Khi đó:

	Mảng 2 chiều a: m hàng, n cột	Mảng 1 chiều b: 1 hàng, m x n cột
Lấy địa chỉ	&a[i][j]	(b + i*n + j)
Lấy dữ liệu	a[i][j]	*(b + i*n + j)

biểu diễn i*n trong b cho ta vị trí của phần tử đầu mỗi hàng tương ứng trong a


III Cấp phát bộ nhớ động

1. Cấp phát bộ nhớ động cho mảng

- Trong phương pháp khai báo mảng *tĩnh*, bộ nhớ dành cho mảng luôn tồn tại đến khi chương trình kết thúc. Có trường hợp khi công việc liên quan đến mảng đó đã kết thúc, bộ nhớ dành cho mảng đó vẫn bị trưng dụng một cách vô ích (chiếm nhưng không sử dụng). Ngoài ra, trong phương pháp khai báo mảng tĩnh, số phần tử dành cho mảng là không "linh hoạt", kích thước bị giới hạn (stack memory).
- Để tối ưu hóa việc sử dụng bộ nhớ (cần bộ nhớ thì được cấp, khi không dùng thì trả cho máy tính phục vụ công việc khác, không trưng dụng bộ nhớ một cách vô ích, số phần tử dành cho mảng là tùy ý lúc chạy chương trình) ta có thể sử dụng các hàm cấp phát động bộ nhớ để cấp phát bộ nhớ.
- Cấp phát động bộ nhớ cho mảng có thể xem là khai báo mảng động.
- Để quản lý vùng nhớ được cấp phát động bởi các hàm phải sử dụng con trỏ

2. Các hàm cấp phát động bộ nhớ

- Cấp phát động bộ nhớ:
  - malloc
  - calloc
  - realloc
- Giải phóng bộ nhớ đã được cấp phát, sau khi sử dụng xong:
  - free
- Thư viện chứa các hàm trên:
  - *#include< stdlib.h >*

Các hàm cấp phát bộ nhớ (tiếp)

- malloc: Cấp phát vùng nhớ n byte, nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp. Nếu không thành công hàm trả về NULL. Các giá trị trong ô nhớ không được khởi tạo.

- prototype: *void *malloc(unsigned n)*

- Để cấp phát bộ nhớ động cho mảng *a* có N phần tử thường dùng cú pháp sau:
  1. Khai báo con trỏ mảng: `type *a ;`
  2. Cấp phát: `a = (type*)malloc(N*sizeof(type)) ;`

Trong đó: type là kiểu dữ liệu của mảng a

Các hàm cấp phát bộ nhớ (tiếp)

- calloc: Cấp phát bộ nhớ cho n đối tượng có kích cỡ size byte (cấp phát vùng nhớ n x size byte), nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp. Nếu không thành công hàm trả về NULL. Các giá trị trong ô nhớ được khởi tạo là 0
- prototype: *void *calloc(unsigned n, unsigned size)*

- Để cấp phát bộ nhớ động cho mảng a có N phần tử thường dùng cú pháp sau:
  1. Khai báo con trỏ mảng: `type *a ;`
  2. Cấp phát: `a = (type*)calloc(N, sizeof(type));`

Trong đó:

type là *kiểu* dữ liệu của mảng a

Các hàm cấp phát bộ nhớ (tiếp)

- calloc: Cấp phát bộ nhớ cho n đối tượng có kích cỡ size byte (cấp phát vùng nhớ n x size byte), nếu thành công hàm trả về địa chỉ đầu vùng nhớ được cấp. Nếu không thành công hàm trả về NULL. Các giá trị trong ô nhớ được khởi tạo là 0
- Cú pháp: *void *calloc(unsigned n, unsigned size)*

- Để cấp phát bộ nhớ động cho mảng a có N phần tử thường dùng cú pháp sau:
  1. Khai báo con trỏ mảng: type *a ;
  2. Cấp phát: a = (type*)calloc(N, sizeof(type));

Trong đó:

type là *kiểu* dữ liệu của mảng a

Giải phóng bộ nhớ

- free: Để giải phóng bộ nhớ đã cấp phát sau khi sử dụng xong

- Cú pháp: *void *free(np)* ;

  Trong đó: np là tên con trỏ chứa địa chỉ đầu của vùng nhớ đã được cấp phát

- Ví dụ: Để giải phóng vùng nhớ đã được cấp phát cho con trỏ a bằng hàm *malloc* hoặc *calloc* đề cập ở trên:

  free(a);


3. Các bước để cấp phát động bộ nhớ

- Khai báo tên mảng dưới dạng con trỏ
- Xác định (nhập) số phần tử của mảng
- Cấp phát động bộ nhớ bằng các hàm đã cho

3.a. Cấp phát động bộ nhớ cho mảng

- **Mảng một chiều**

Ví dụ:
```
float   *a;  int n;
printf("Nhap vao gia tri n: ");
scanf("%d", &n);
a = (float *)calloc(n, sizeof(float));
```
hoặc:
```
a = (float*)malloc(n*sizeof(float));
```

- **Mảng hai chiều**

Ví dụ:
```
float   *b;  int m, n;
printf("Nhap vao gia tri m, n: ");
scanf("%d%d", &m, &n);
b = (float*)calloc(m*n, sizeof(float));
```
hoặc:
```
b = (float*)malloc(m*n*sizeof(float));
```

Ví dụ: Tính tổng n số thực

```
# include <stdio.h>
# include <stdlib.h>
main()
{
     int i, N; float  s = 0, *a;
     printf("Nhap vao gia tri n: ");
     scanf("%d", &N);
     a = (float *) malloc(N*sizeof(float));
     if(a==NULL)
     {
              exit(1);
     }
     for(i=0; i < N; i++)
     {
          printf("\na[%d]= ",i);
          scanf("%f", (a+i));
          s = s + *(a+i);
     }
     printf("\n Tong =%8.2f",s);
     free(a);
}
```

Khai báo con trỏ a

Cấp phát *N*sizeof(loat)* byte cho a

Nhập và tính tổng các phần tử

Ví dụ: Tìm số lớn nhất trong ma trận

```
# include <stdio.h>
# include <stdlib.h>
main()
{
     int   m, n, max, i, j, *a;
     printf("Nhap vao gia tri m, n: ");
     scanf("%d%d", &m,&n);
     a = (int *) calloc(m*n, sizeof(int));
     if(a!=NULL)
     {
        max=a[0];
       for(i=0;i<m;i++)
            for(j=0;j<n;j++)
                 {
                    printf("\na[%d, %d]= ",i, j);
                    scanf("%d",&a[i*n+j]);
//hoặc:               scanf("%d", (a + i*n + j));
                    if (max <= a[i*n+j]) max = a[i*n+j];
                 }
       printf("\n Max la:  =%d",max);
      }
      free(a);
}
```

Khai báo con trỏ a

Cấp phát bộ nhớ cho *m x n* phần tử

Nhập mảng và tìm *max*

tìm *max*

Tóm tắt bài học

- Địa chỉ
- Con trỏ
  - Các phép toán trên con trỏ
- Mảng một chiều và hai chiều
  - Khai báo mảng tĩnh
  - Cách truy xuất phần tử trong mảng một chiều, hai chiều.
  - Làm việc với mảng 1D, 2D,… dùng con trỏ.
- Cấp phát động bộ nhớ
  - Cấp phát động bộ nhớ cho mảng một chiều, hai chiều

Nội dung chính đã học

1. Địa chỉ và Con trỏ
2. Mảng
3. Cấp phát động bộ nhớ
4. Bài tập


Bài tập

1. Nhập vào hai ma trận A, B kích thước 2 x 3. Tính và cho hiển thị A+B.
2. Nhập vào một dãy n phần tử. In ra dãy số ngược lại với dãy nhập vào
3. Nhập vào một ma trận nguyên kích thước m x n. Hãy tìm phần tử lớn thứ nhì trong ma trận trên.
4. Nhập vào một ma trận nguyên kích thước m x n. In ra các số lẻ trong ma trận đó.
5. Nhập vào một mảng n phần tử. Sắp xếp lại mảng theo thứ tự tăng dần và in ra màn hình.

Bài tập

6. Nhập vào ma trận A kích thước m x n và ma trận B kích thước n x p. Tính và hiển thị tích hai ma trận.
7. Tìm ước chung lớn nhất của hai số a và b.
8. Nhập và kiểm tra xem số n có phải là số nguyên tố hay không?
9. Nhập vào một dãy n phần tử và một số m bất kỳ. Hãy đếm số lần xuất hiện của số m trong dãy trên.