LẬP TRÌNH C NÂNG CAO

- Thời lượng: 2TC (10LT + 20 BT&TL + 20TH)
- Nội dung:
  - Chương 1. Con trỏ (3 LT + 6 BT&TL + 5 TH)
  - Chương 2. Các kiểu dữ liệu do người dùng định nghĩa (3 LT + 6 BT&TL + 5 TH)
  - Chương 3. File (3 LT + 6 BT&TL + 5 TH)
  - Chương 4. Tổ chức bộ nhớ chương trình. Các chỉ thị tiền xử lý (1 LT + 2 BT&TL + 5 TH)

CHƯƠNG 1. CON TRỎ

- Khái niệm
- Khai báo
- Các phép toán trên con trỏ
- Con trỏ và mảng một chiều
- Con trỏ và mảng hai chiều
- Con trỏ và xâu kí tự
- Con trỏ hàm

Biến

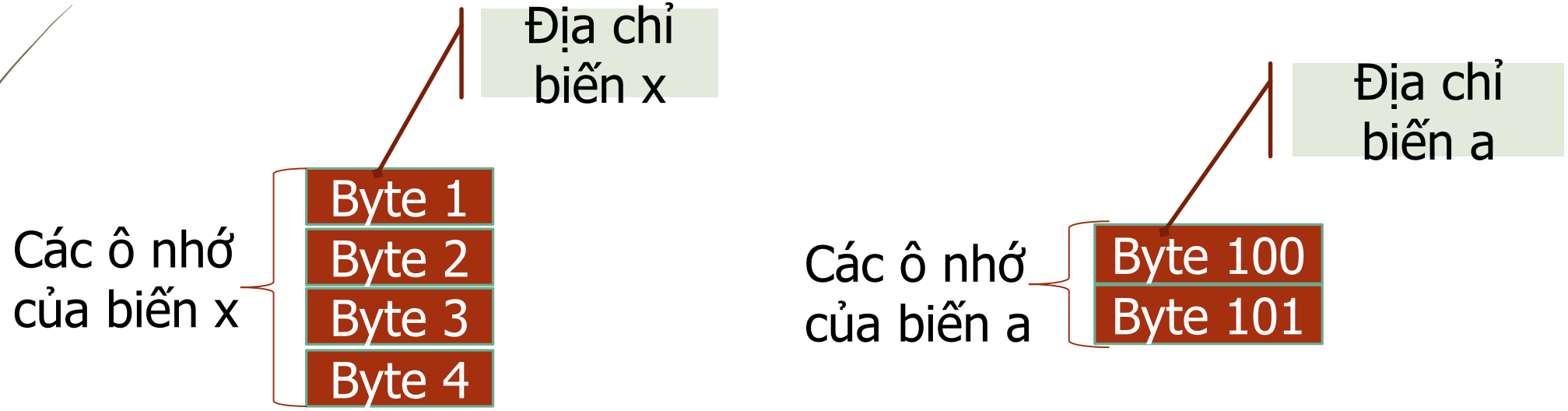
- Các biến có thể khai báo bên trong hoặc bên ngoài hàm
- Biến khai báo ngoài hàm là biến toàn cục và có vị trí bộ nhớ cố định
- Biến khai báo trong khối lệnh {}/ trong hàm:
  - Động nếu không dùng static
  - Được cấp phát khi chương trình thực thi vào khối
  - Bộ nhớ được giải phóng khi ra khỏi khối
- Thông tin của một biến bao gồm:
  - Tên biến
  - Kiểu dữ liệu của biến
  - Giá trị của biến

Mỗi biến sẽ được lưu trữ tại một vị trí xác định trong ô nhớ, nếu kích thước của biến có nhiều byte thì máy tính sẽ cấp phát một dãy các byte liên tiếp nhau, địa chỉ của biến sẽ lưu byte đầu tiên trong dãy các byte này

Địa chỉ của biến

Ví dụ khai báo:

```
float x;
int a;
```



Địa chỉ của biến

- Địa chỉ của biến luôn luôn là một số nguyên (hệ hexa) cho dù biến đó chứa giá trị là số nguyên, số thực, ký tự, …
- Cách lấy địa chỉ của biến: & tênbiến
- Ví dụ:

```
int main(){
   int x=7;
   float y=10.5;
   printf("Dia chi cua bien x = %x\n", &x);
   printf("Dia chi cua bien y = %x", &y);
   return 0;
}
```

Biến con trỏ

- Biến con trỏ là một biến để chứa địa chỉ của ô chứa dữ liệu, có nhiều loại con trỏ tùy thuộc vào địa chỉ của mỗi loại kiểu biến
- Khai báo con trỏ: **Kiểu dữ liệu * TênConTrỏ;**
- Ví dụ: int *px;
  float *pm;
- Lấy giá trị của con trỏ: ***TênConTrỏ;**
- Vì giá trị của con trỏ là địa chỉ, nên khi thay đổi giá trị đó, biến con trỏ sẽ trỏ tới một vùng nhớ khác
- Gán địa chỉ mới cho con trỏ bằng phép gán như thông thường

  Ví dụ: int *p2;
  p2 = p;
- **Toán tử địa chỉ &**: tạo ra một con trỏ bằng việc lấy địa chỉ của một biến

  Ví dụ: int a;
  int *pa = &a; /* pa trỏ tới a */

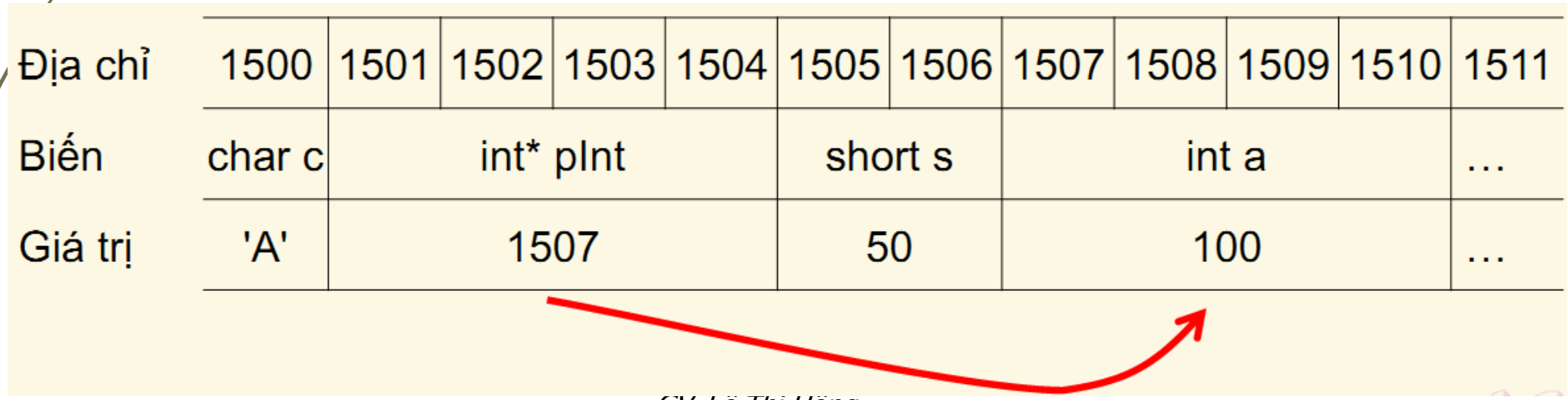
Biến con trỏ

- Ví dụ:

```
char c = 'A';
int *pInt;
short s = 50;
int a = 10;
pInt = &a;
*pInt = 100;
```

Địa chỉ các biến trong bộ nhớ theo thứ tự tăng dần ở đây chỉ có tính chất minh hoạ. Trong thực tế, stack được cấp phát từ cao xuống thấp biến khai báo sau sẽ có địa chỉ nhỏ hơn.

pInt:	1507	*pInt:	100
&a:	1507	a:	100



Con trỏ - Ví dụ

```
int main(){
    int *px, x;
    float *pa, a;
    x=10;
    a=14.5;
    p=&x; //Lấy địa chỉ của biến x gán vào p
    printf("Gia tri cua px = %d\n", *p);
    p=&a; //Lấy địa chỉ của biến a gán vào p
    printf("Gia tri cua pa = %f", *p);
    return 0;
    p=&a;
}
```

Kết quả
Gia tri cua px = 10
Gia tri cua pa = 14.5

Lấy giá trị của con trỏ p

Con trỏ - Ví dụ

```
int main(){
    int *px, y;
    float *pa, b;
    y=10;
    b=14.5;
    px=&y;
    pa=&b;
    printf("Dia chi cua bien y = %x\n", &y);
    printf("Dia chi cua bien px = %x\n", px);
    printf("Dia chi cua bien b = %x\n", &b);
    printf("Dia chi cua bien pa = %x\n", pa);
    return 0;
}
```

Sử dụng biến con trỏ

```
int main(){
  int *px;
  printf("Nhap vao gia tri cho con tro px: ");
  scanf("%d", px);
  printf("Gia tri cua px = %d", *px);
  return 0;
}
```

!!!Chưa cấp phát bộ nhớ trước khi sử dụng

1 error C4700: uninitialized local variable 'px' used

```
int main(){
  int *px;
  px = (int *)calloc(1, sizeof(int)); //alloc.h
  printf("Nhap vao gia tri cho con tro px: ");
  scanf("%d", px);
  printf("Gia tri cua px = %d", *px);
}
```

Cấp phát bộ nhớ cho px

Kết quả

Nhap vao gia tri cho con tro px: 16
Gia tri cua px = 16

Cấp phát và giải phóng vùng nhớ

Biến con trỏ phải được cấp phát vùng nhớ trước khi sử dụng

- **Cách 1: dùng calloc**

```
int *p; //khai báo con trỏ p
//cấp phát 100 ô nhớ kiểu int cho con trỏ p (mỗi ô chiếm 2bytes)
p=(int *) calloc (100, sizeof (int));  //thư viện alloc.h
```

- **Cách 2: dùng malloc**

```
int *px;
//Cấp phát 100 ô nhớ kiểu int cho con trỏ px
px = (int *) malloc (100);  //thư viện malloc.h
```

- Sau khi sử dụng xong thì giải phóng vùng nhớ bằng hàm free

```
free(px) ; // giải phóng vùng nhớ cho con trỏ px
```

Ví dụ: Cấp phát và giải phóng vùng nhớ

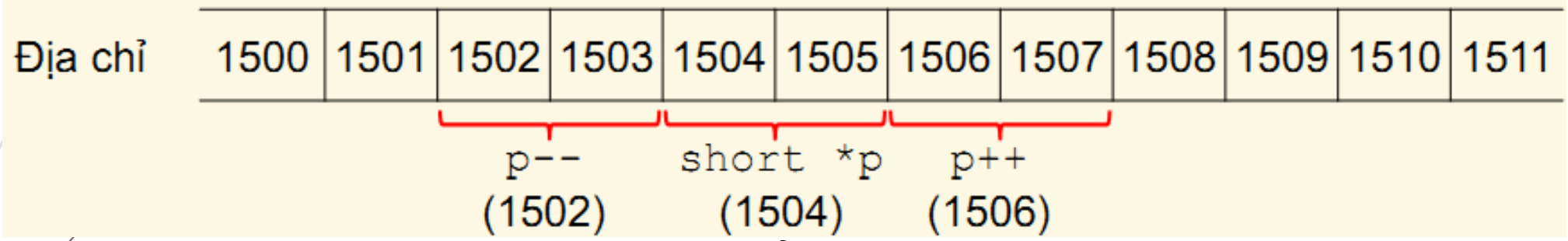
```
int main(){
  int *px;
  px = (int *)malloc(1);
  printf("Nhap vao gia tri cho con tro px: ");
  scanf("%d", px);
  printf("Gia tri cua px = %d", *px);
  free(px);
  return 0;
}
```

Cấp phát bộ nhớ cho con trỏ px

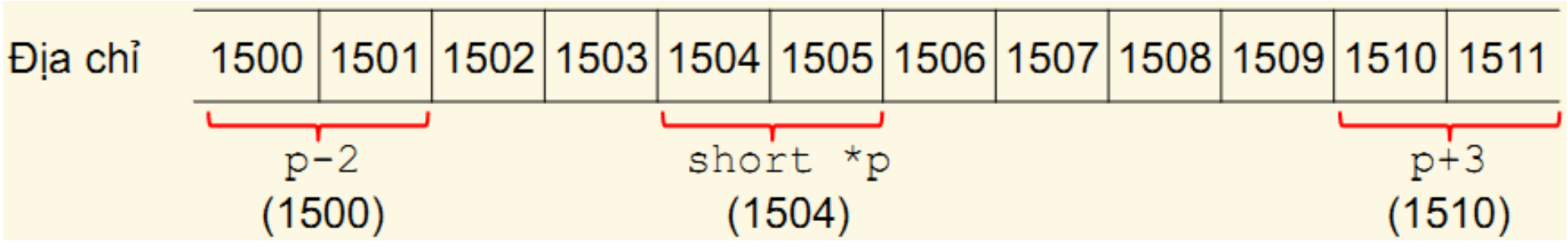
Giải phóng con trỏ px

Các phép toán với con trỏ

- **Tăng giảm**: để thay đổi con trỏ trỏ tới vị trí tiếp theo (tương ứng với kích thước kiểu nó trỏ tới)



- **Cộng địa chỉ**: cũng tương ứng với kiểu nó trỏ tới



- **So sánh**: 2 con trỏ cùng kiểu có thể được so sánh địa chỉ với nhau như 2 số nguyên (lớn, nhỏ, bằng)
- Hai con trỏ cùng kiểu có thể trừ cho nhau để ra số phần tử sai khác

Con trỏ và mảng

- Mảng là một con trỏ tĩnh (không thể thay đổi địa chỉ), chứa địa chỉ (trỏ) tới phần tử đầu tiên của nó. Có thể thao tác với biến kiểu mảng như thao tác với con trỏ, chỉ trừ việc gán địa chỉ mới cho nó

```
int a[] = {1, 2, 3, 4, 5};
int x;
*a = 10;  //a[0] = 10;
printf("%d", *(a+2)); //a[2]
a = &x; //lỗi
```

- Con trỏ cũng có thể hiểu là mảng và có thể được thao tác như một mảng

```
int *p = a;
p[2] = 20; //a[2] = 20;
p = a+2;  //p = &a[2];
p[0] = 30;  //a[2] = 30; hoặc: *p = 30;
```

- **Con trỏ và mảng có thể dùng thay thế cho nhau**, tuỳ trường hợp mà dùng cái nào cho thuận tiện

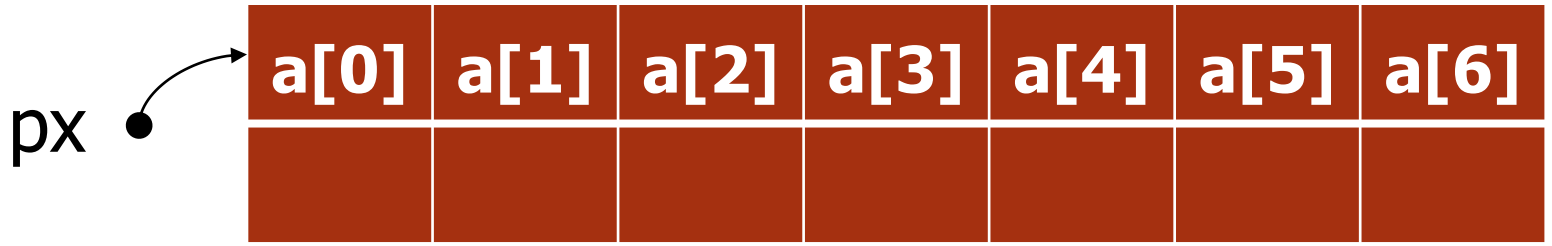
Con trỏ và mảng

- Khác biệt:
  - Không gán được địa chỉ mới cho biến kiểu mảng
  - Biến kiểu mảng được cấp phát bộ nhớ cho các phần tử (trong stack) ngay từ khi khai báo
  - Toán tử sizeof() với mảng cho biết kích thước thực của mảng (tổng các phần tử), trong khi dùng với con trỏ thì cho biết kích thước của bản thân nó (chứa địa chỉ)

```
float a[5]; //sizeof(a) trả    về  20 (5*4)
float* p = a; //sizeof(p) trả về 4 với hệ thống 32 bit
sizeof(a)/sizeof(a[0]) //số phần tử của mảng
```

Con trỏ và mảng một chiều

```
int a[7];
int *px;
px = a;        //px trỏ tới phần tử thứ 0
px = px + 4;   //px trỏ tới  phần tử thứ 4
```



Khi đó 4 cách viết sau là tương đương:

a[i] *(a+i) *(px+i) px[i]

Ví dụ: Viết chương trình sử dụng con trỏ nhập vào một dãy số nguyên và sau đó xuất dãy số đó ra màn hình

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

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

void xuatmang(int *a, int n){
  for(int i=0; i<n; i++)
     printf("%d\t", *(a+i));
}
```

Hoặc &a[i]

Hoặc a[i]

```
int main(){
     int *a, n=4;
     a=(int *)malloc(n);
     nhapmang(a, n);
     xuatmang(a, n);
     free(a);
     return 0;
}
```

Thay đổi kích thước của con trỏ đã cấp phát

- Cú pháp: **realloc(tên con trỏ, kích thước mới);**

```
int main(){
    int *a, n=4;
    a=(int *)malloc(n);
    nhapmang(a, n);
    xuatmang(a, n);
    n=7;
    realloc(a, n);
    nhapmang(a, n);
    xuatmang(a, n);
    free(a);
    return 0;
}
```

Cấp phát vùng nhớ động trên môi trường C++

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

int maxn(int *a, int n){
    int i, max =a[0];
    for(i=1; i<n; i++)
        if(a[i]>max)
            max = a[i];
    return max;
}
```

```
int main(){
    int *a, n=5;
    a=new int[n];
    a[0]=1; a[1]=12; a[2]=6;
    a[3]=9; a[4]=3;
    printf("max=%d", maxn(a, n));
    delete[] a;
    getch();
    return 0;
}
```

Ví dụ: Viết chương trình sử dụng con trỏ nhập vào một dãy số nguyên từ tệp cau2.in và sau đó xuất dãy số đó ra tệp cau2.dat. Đưa số hạng lớn nhất của dãy ra màn hình.

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

//Nhap vao 1 day so nguyen
void doc(int *a, int &n, FILE *f){
    f=fopen("cau2.in","rt");
    if(f==NULL){
        printf("Loi mo tep!");
        exit(0);
    }
    fscanf(f,"%d",&n);
    for(int i=0; i<n; i++)
        fscanf(f,"%d", a+i);//&a[i]
    fclose(f);
}
```

```
void ghi(int *a, int n, FILE *f){//Ghi 1 day so
    f=fopen("Cau2.dat","wt");
    for(int i=0; i<n; i++)   fprintf(f,"%8d",*(a+i));//a[i]
    fclose(f);
}
int maxn(int *a, int n){
    int i, max =a[0];
    for(i=1; i<n; i++)  if(a[i]>max)     max = a[i];
    return max;
}
int main(){
        int *a, n;  FILE *f;
        a=new int[n]; //a=(int *)malloc(20);
        doc(a, n, f); ghi(a,n,f);
        delete[] a; //free(a);
        return 0;
}
```

Con trỏ và mảng 2 chiều

- Giả sử ta có mảng hai chiều a[2][3] có 6 phần tử ứng với sáu địa chỉ liên tiếp trong bộ nhớ được xếp theo thứ tự:

Phần tử:	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
Địa chỉ	1	2	3	4	5	6

- Tên mảng a biểu thị địa chỉ đầu tiên của mảng. Phép cộng địa chỉ ở đây được thực hiện như sau :

  a trỏ phần tử thứ nhất của mảng : phần tử a[0][0]

  a+1 trỏ phần tử đầu hàng thứ hai: phần tử a[1][0] *(Không phải là a[0][1]!!!)*

Con trỏ và mảng 2 chiều

- Để lần lượt duyệt trên các phần tử của mảng hai chiều ta cũng có thể dùng con trỏ:

```
float *p, a[2][3];
p = (float*)a;
```

Lúc đó:

p	trỏ tới a[0][0]	p+3	trỏ tới a[1][0]
p+1	trỏ tới a[0][1]	p+4	trỏ tới a[1][1]
p+2	trỏ tới a[0][2]	p+5	trỏ tới a[1][2]

Trong khi:

a	trỏ tới a[0][0]
a+1	trỏ tới a[1][0]

- **Lưu ý: Đối với mảng 2 chiều thì a+i khác với p+i.**

Ví dụ: Viết chương trình sử dụng hàm, con trỏ và cấp phát động nhập ma trận cấp mxn với 0<m,n<100 (từ bàn phím và từ tệp)

Xuất ma trận vừa nhập (ra màn hình và ra tệp). Tìm phần tử lớn nhất trong ma trận và xuất ra màn hình

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

//Nhap ma tran
void nhapmt(int *a, int m, int n){int i, j;
    for(i = 0; i < m; i++)
    for(j = 0; j < n; j++){
        printf("a[%d,%d]= ", i, j);
        scanf("%d", a+i*n+j);//a[i]j]
    }
}
```

```
//Xuat ma tran
void xuatmt(int* a, int m, int n){
    int i, j;
    for(i = 0; i < m; i++){
            printf("\n");
        for(j = 0; j < n; j++)
            printf("%5d", *(a+i*n+j));//a[i*n+j]
    }
}
```

```
//Nhap ma tran tu tep bai4.in
void docmt(int *a, int &m, int &n, FILE *f){
    int i, j;
    f=fopen("bai4.in", "rt");
    if(f==NULL){
        printf("Loi mo tep.");
        exit(0);
     }
     fscanf(f,"%d",&m);
     fscanf(f,"%d",&n);
     for(i = 0; i < m; i++)
     for(j = 0; j < n; j++)
        fscanf(f,"%d", a+i*n+j);//a[i]j]
     fclose(f);
}
```

```
//Xuat ma tran ra tep bai4.out
void ghimt(int* a, int m, int n, FILE *f){
    int i, j;
    f=fopen("bai4.out","wt");
    for(i = 0; i < m; i++){
        fprintf(f,"\n");
        for(j = 0; j < n; j++)
          fprintf(f,"%5d", *(a+i*n+j));//a[i*n+j]
    }
    fclose(f);
}
```

```
//tim phan tu lon nhat trong ma tran
int maxmt(int *a, int m, int n){
      int i, j, max=*a;
    for(i = 0; i < m; i++)
    for(j = 0; j < n; j++)
        if(*(a +i*n+j)>max)
          max=*(a +i*n+j);
    return max;
}
```

```
int main(){
    int m, n, *a;
    FILE *f;
    //printf("Nhap so dong: "); scanf("%d",&m);
    //printf("Nhap so cot: "); scanf("%d",&n);
    a = (int *)malloc(50*sizeof(int));
    //nhapmt(a,m,n);
    docmt(a,m,n,f);
    printf("\nMa tran vua nhap la:\n");
    xuatmt(a,m,n);
    ghimt(a,m,n,f);
    printf("\nGia tri lon nhat cua ma tran la: %d",
          maxmt(a,m,n));
    free(a);
    return 0;
}
```

Con trỏ và chuỗi ký tự

- Khi gặp một xâu ký tự, máy sẽ cấp phát một khoảng nhớ cho một mảng kiểu **char** đủ lớn để chứa các ký tự của xâu và chứa thêm ký tự '\0' là ký tự dùng làm ký tự kết thúc của một xâu ký tự. Mỗi ký tự của xâu được chứa trong một phần tử của mảng.
- Cũng giống như tên mảng, xâu ký tự là một hàng địa chỉ biểu thị địa chỉ đầu của mảng chứa nó. Vì vậy nếu ta khai báo biến **xau** như một con trỏ kiểu char:

  char *xau; thì phép gán: xau = "Ha noi"; là hoàn toàn có nghĩa.
- Sau khi thực hiện câu lệnh này trong con trỏ **xau** sẽ có địa chỉ đầu của mảng (kiểu **char**) đang chứa xâu ký tự bên phải. Khi đó các câu lệnh:

```
puts("Ha noi");
puts(xau);
```

sẽ có cùng một tác dụng là cho hiện lên màn hình dòng chữ **Ha noi**

Con trỏ và chuỗi ký tự

- So sánh mảng kiểu char và con trỏ kiểu char:

```
char *xau, ten[15];
ten = "Ha noi";
gets(xau);
```

  - Lệnh **ten="Ha noi";** không hợp lệ vì **ten** là một hằng địa chỉ, không thể gán một hằng địa chỉ này cho một hằng địa chỉ khác.
  - Lệnh **gets(xau);** không thực hiện được, mục đích của câu lệnh là đọc từ bàn phím một dãy ký tự và lưu vào một vùng nhớ mà con trỏ **xau** trỏ tới. Song nội dung của con trỏ **xau** còn chưa xác định.
  - Nếu trỏ **xau** đã trỏ tới một vùng nhớ nào đó thì câu lệnh này hoàn toàn có ý nghĩa. Chẳng hạn:

```
char *xau, ten[15];
xau = ten;
```

thì 2 cách viết: **gets(ten);** hay **gets(xau);** đều có tác dụng như nhau.

Con trỏ và chuỗi ký tự

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

int main(){
    char *ten;
    ten = (char *) malloc (100*sizeof(char));
    printf("Ten ban la gi? ");
    gets(ten);
    printf("Chao ban %s\n", ten);
    getch();
    return 0;
}
```

Con trỏ và chuỗi ký tự

Ví dụ: Nhập vào một chuỗi và 1 ký tự bất kỳ, in ra các vị trí của ký tự đó trong chuỗi, đếm số lần nó xuất hiện và xóa ký tự đó có trong chuỗi.

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

//Tim vi tri, dem so ky tu c
void timvitri(char *chuoi, char c){
    int dem =0;
    for(int i=0;chuoi[i]!='\0';i++)
        if(chuoi[i] == c){
            dem++;
            printf("\nKy tu %c o vi tri do
                la %d",c,i);
        }
    printf("\nSo lan no xuat hien
                ky tu %c la %d",c,dem);
}
```

Con trỏ và chuỗi ký tự

```
//Xoa ky tu c
void xoakytu(char *chuoi, char c){
    int i,j;
    for(i=0;chuoi[i]!='\0';i++){
        if(chuoi[i]==c){
            j = i;
            while(chuoi[j+1]!='\0'){
                chuoi[j] = chuoi[j+1];
                j++;
            }
            chuoi[j] = '\0';
            i--;
        }
    }
}
```

```
int main(){
    char *chuoi, c;
    chuoi = (char *) malloc (100*sizeof(char));
    printf("\nNhap chuoi: "); gets(chuoi);
    printf("\nnhap 1 ky tu: "); scanf("%c",&c);
    timvitri(chuoi,c);
    printf("\nchuoi sau khi xoa ky tu %c la: ",c);
    xoakytu(chuoi,c); puts(chuoi);
    return 0;
}
```

Con trỏ hàm

- Khai báo: <kdl> (*Tên hàm)(tham số);
- VD: int *Tong(int a, int b);

Thích hợp cho việc tùy chọn (switch) gọi thực hiện trong danh sách các hàm

```
int chuvi(int a, int b){
    return (a + b) * 2;
}
int dientich(int a, int b){
    return a*b;
}
int tinh(int a, int b, int (*ham)(int, int)){
    int kq = (*ham)(a, b);
    return kq;
}
```

```
int main(){
    int a = 10, b = 6;
    int (*ham)(int, int) = chuvi;
    int p = tinh(a, b, ham);
    printf("Chu vi cua hcn = %d", p);
    int (*ham)(int, int) = dientich;
    int s = tinh(a, b, ham);
    printf("Dien tich cua hcn = %d",
    s);
}
```

Con trỏ và hàm

Đối của hàm là con trỏ:

```
#include <stdio.h>
#include <conio.h>
void  hoanvi(int* x, int* y){
    int c =*x;  *x=*y;  *y=c;
}
int main(){
    int x = 3, y = 5;
    hoanvi(&x, &y);
    printf("x = %d, y = %d\n", x, y);
    getch();
    return 0;
}
```

```
#include <stdio.h>
#include <conio.h>
int maxn(int *a, int n){
    int i, max =a[0];
    for(i=1; i<n; i++)
        if(a[i]>max)      max = a[i];
    return max;
}
int main(){
    int x[5] = {1,6,13,7,9};
    printf("max=%d", maxn(x, 5));
    getch();
    return 0;
}
```

Bài tập

Bài 1: Viết chương trình **sử dụng hàm, con trỏ và cấp phát động** thực hiện các yêu cầu sau:

1. Nhập vào một dãy số gồm n số nguyên với 0<n<100 (từ bàn phím và từ tệp)
1. Nhập vào một dãy số gồm n số nguyên với 0<n<100 (từ bàn phím và từ tệp)
2. Xuất dãy vừa nhập (ra màn hình và ra tệp)
3. Tính tổng các số âm và tổng các số dương trong dãy
4. Tính tổng các phần tử lẻ và tổng các phần tử chẵn trong dãy
5. Tính tổng các phần tử trong dãy là bội của 3 và 5
6. Tính tổng các số lớn hơn trung bình cộng của các số trong dãy
7. Đếm các phần tử âm, dương trong dãy
8. Đếm các phần tử chẵn, lẻ trong dãy
9. Tìm vị trí phần tử âm đầu tiên trong dãy, nếu không có phần tử âm trả về -1
10. Nhập vào một số z, tìm vị trí của số hạng đầu tiên trong dãy có giá trị bằng z, nếu không có số hạng bằng x trả về giá trị -1

Bài tập

11. Tìm số cuối cùng trong dãy lớn hơn trung bình cộng của các số trong dãy
12. Tìm vị trí của phần tử lớn nhất, bé nhất trong dãy
13. Tìm vị trí phần tử âm lớn nhất của dãy, nếu không có phần tử âm trả về -1
14. Tìm và đổi chỗ phần tử lớn nhất với phần tử nhỏ nhất trong dãy
15. Liệt kê các phần tử là số nguyên tố trong dãy
16. Tính tổng các số nguyên tố trong dãy
17. Viết hàm in vị trí các phần tử nguyên tố lớn hơn số a (với a nhập từ bàn phím)
18. Sắp xếp dãy số theo thứ tự tăng dần
19. Sắp xếp các phần tử lẻ tăng dần
20. Sắp xếp dãy số sao cho các số chia hết cho 3 lên đầu dãy

Bài tập

21. Sắp xếp dãy số sao cho các số chẵn lên đầu dãy và các số chẵn được sắp xếp theo thứ tự tăng dần
22. Đảo ngược dãy số
23. Nhập vào 2 số nguyên x và k (với 0<k<n), chèn số x vào vị trí k trong dãy
24. Nhập vào một số x, chèn x vào dãy sau khi đã sắp xếp tăng sao cho không làm mất tính tăng dần của dãy
25. Nhập vào một số k (với 0<k<n), xóa số hạng tại vị trí thứ k trong dãy
26. Nhập vào một số y, xóa số hạng đầu tiên bằng y trong dãy (nếu có)
27. Nhập vào một số y, xóa tất cả các số hạng bằng y trong dãy (nếu có)
28. Xóa tất cả các số hạng lớn hơn số hạng đầu tiên trong dãy

Bài tập

Bài 2: Viết chương trình **sử dụng hàm, con trỏ và cấp phát động** thực hiện các yêu cầu sau:

1. Nhập ma trận cấp mxn với 0<m,n<100 (từ bàn phím và từ tệp)
2. Xuất ma trận vừa nhập (ra màn hình và ra tệp)
3. Tìm phần tử lớn nhất trong ma trận
4. Tìm vị trí phần tử lớn nhất trong ma trận
5. Tìm vị trí phần tử chẵn cuối cùng trong ma trận
6. Tính tổng các phần tử chẵn có trong ma trận
7. Tìm các phần tử là số nguyên tố có trong ma trận
8. Tìm phần tử chẵn dương và nhỏ nhất trong ma trận
9. Tìm các phần tử nằm trên 2 đường chéo
10. Tính giá trị trung bình của các phần tử nhỏ nhất trên mỗi dòng