Chương 2: Quản lý bộ nhớ

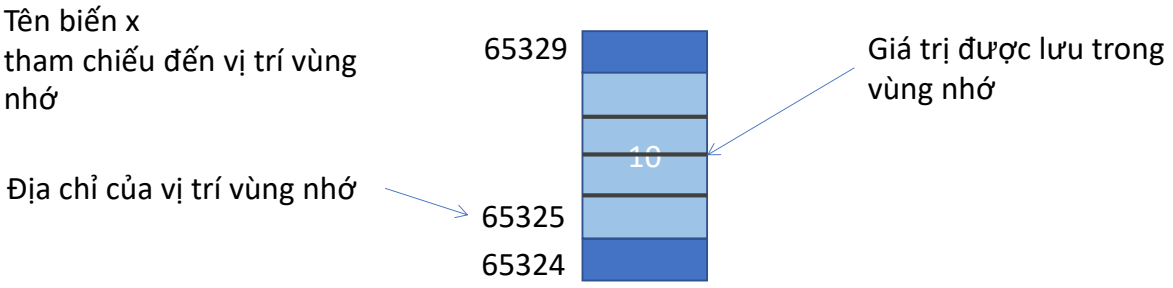
Biến: tên biến, vùng nhớ và giá trị biến

- Mỗi biến trong C có tên và giá trị tương ứng. Khi một biến được khai báo, một vùng nhớ trong máy tính sẽ được cấp phát để lưu giá trị của biến. Kích thước vùng nhớ phu thuộc kiểu của biến, ví dụ 4 byte cho kiểu int

```
int x = 10;
```

- Khi lệnh này được thực hiện, trình biên dịch sẽ thiết lập để 4 byte vùng nhớ này lưu giá trị 10.
- Phép toán & trả về địa chỉ của *x*, nghĩa là địa chỉ của ô nhớ đầu tiên trong vùng nhớ lưu trữ giá trị của *x*.

Biến: tên biến, vùng nhớ và giá trị biến



- `int x = 10;`
- `x` biểu diễn tên biến
- `&x` biểu diễn địa chỉ ô nhớ đầu tiền thuộc vùng nhớ của `x`, tức là 65325
- `*(&x)` hoặc x biểu diễn giá trị được lưu trong vùng nhớ của `x`, tức là 10

Chương trình ví dụ

```
#include<stdio.h>
int main() {
    int x = 10;
    printf("Value of x is %d\n",x);
    printf("Address of x in Hex is %p\n",&x);
    printf("Address of x in decimal is %lu\n",&x);
    printf("Value at address of x is %d\n",*(&x));
    return 0;
}
```

```
Value of x is 10
Address of x in Hex is 0061FF0C
Address of x in decimal is 6422284
Value at address of x is 10
```

Khái niệm con trỏ

- Con trỏ là một biến chứa địa chỉ vùng nhớ của một biến khác.

- Cú pháp chung khai báo biến con trỏ

```
data_type *ptr_name;
```

- Ký hiệu '*' thông báo cho trình biên dịch rằng *ptr_name* là một biến con trỏ và *data_type* chỉ định rằng con trỏ này sẽ trỏ tới địa chỉ vùng nhớ của một biến kiểu *data_type*.

Tham chiếu ngược (dereference) biến con trỏ

- Chúng ta có thể "tham chiếu ngược" một con trỏ, nghĩa là lấy giá trị của biến mà con trỏ đang trỏ vào bằng cách dùng pháp toán '*', chẳng hạn `*ptr`.

- Do đó, `*ptr` *có giá trị 10*, vì 10 đang là giá trị của *x*.

Ví dụ con trỏ

```
int x = 10;

int *p = &x;
```

Nếu con trỏ `p` giữ địa chỉ của biến `x`, ta nói `p` trỏ tới biến `x`

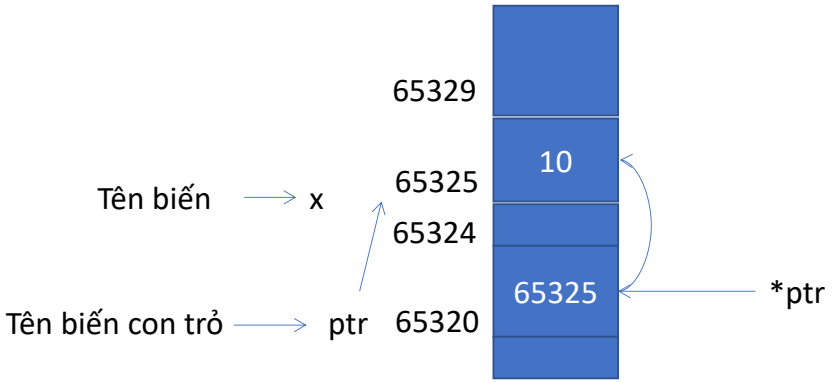
`*p` biểu diễn giá trị lưu tại địa chỉ `p`

`*p` được gọi là "tham chiếu ngược" (dereference) của con trỏ `p`

- Con trỏ được dùng để lấy địa chỉ của biến nó trỏ vào, đồng thời cũng có thể lấy giá trị của biến đó

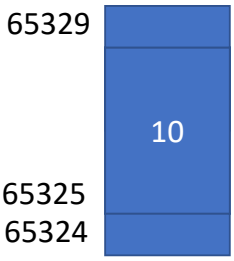
Con trỏ

```
int x = 10;
int *ptr;
ptr = &x;
```



Biến và con trỏ

Biến
int x = 10;
&x biểu diễn địa chỉ của x
x biểu diễn giá trị lưu tại địa chỉ &x



Con trỏ
int *p; p được khai báo là con trỏ int
p = &x; p giữ giá trị địa chỉ x, i.e., 65325
*p biểu diễn giá trị lưu tại p, ie., 10

```
int *p;
*p = 10; // this is not correct.
```


Ví dụ con trỏ

```
#include<stdio.h>
int main() {
  int x = 10;
  int *ptr;
  ptr = &x;
  printf("Value of x is %d\n", x);
  printf("Address of x is %lu\n", (unsigned long int) &x);
  printf("Value of pointer ptr is %lu\n", (unsigned long int)
  ptr);
  printf("Address of pointer ptr is %lu\n", (unsigned long int)
  &ptr);
  printf("Ptr pointing value is %d\n",*ptr);
  return 0;
}
```

```
Value of x is 10
Address of x is 6422284
Value of pointer ptr is 6422284
Address of pointer ptr is 6422280
Ptr pointing value is 10
```

Các phép toán trên con trỏ

- Cộng hoặc trừ với 1 số nguyên n trả về 1 con trỏ cùng kiểu, là địa chỉ mới trỏ tới 1 đối tượng khác nằm cách đối tượng đang bị trỏ n phần tử
- Trừ 2 con trỏ cho ta khoảng cách (số phần tử) giữa 2 con trỏ
- KHÔNG có phép cộng, nhân, chia 2 con trỏ
- Có thể dùng các phép gán, so sánh các con trỏ
  - Chú ý đến sự tương thích về kiểu.

Ví dụ

```
char *pchar; short *pshort; long *plong;
pchar ++; pshort ++; plong ++;
```

- Giả sử các địa chỉ ban đầu tương ứng của 3 con trỏ là 100, 200 và 300,
  - kết quả ta có các giá trị 101, 202 và 308 tương ứng

Nếu viết tiếp

```
pchar -=10; => pchar = 91
pshort +=5; => pshort = 212
plong += 5; => plong = 348
```

Kiểu dữ liệu	Kích thước (byte)	Phạm vi giá trị (ước lượng)
char	1	-128 đến 127
short	2	-32768 đến 32767
int	4	-2147483648 đến 2147483647
long	4 hoặc 8	Tùy thuộc hệ thống
long long	8	Rất lớn

sizeof

```c
//C
#include <stdio.h>
int main() {
    printf("Size of char: %d", sizeof(char));
    printf("\nSize of short: %d", sizeof(short));
    printf("\nSize of int: %d", sizeof(int));
    printf("\nSize of long: %d", sizeof(long));
    printf("\nSize of long long: %d", sizeof(long long));
    printf("\nSize of float: %d", sizeof(float));
    printf("\nSize of double: %d", sizeof(double));
    return 0;
}

//C++
#include <iostream>
using namespace std;

int main() {
    std::cout << "Size of char: " << sizeof(char) << " byte" << std::endl;
    std::cout << "Size of short: " << sizeof(short) << " byte" << std::endl;
    std::cout << "Size of int: " << sizeof(int) << " byte" << std::endl;
    std::cout << "Size of long: " << sizeof(long) << " byte" << std::endl;
    std::cout << "Size of long long: " << sizeof(long long) << " byte" << std::endl;
    std::cout << "Size of float: " << sizeof(float) << " byte" << std::endl;
    std::cout << "Size of double: " << sizeof(double) << " byte" << std::endl;
    return 0;
}
```

```
Size of char: 1 byte
Size of short: 2 byte
Size of int: 4 byte
Size of long: 8 byte
Size of long long: 8 byte
Size of float: 4 byte
Size of double: 8 byte
```


Ví dụ: Phép toán trên con trỏ

```c
#include<stdio.h>
int main() {
  char *pchar; short *pshort; long *plong;
  pchar = (char *)100;
  pshort = (short *)200;
  plong = (long *)300;

  printf("Value of pchar is %d\n", pchar);
  printf("Value of pshort is %d\n", pshort);
  printf("Value of plong is %d\n", plong);

  pchar ++; pshort ++; plong ++;

  printf("Value of pchar is %d\n", pchar);
  printf("Value of pshort is %d\n", pshort);
  printf("Value of plong is %d\n", plong);


  pchar -=10;
  pshort +=5;
  plong += 5;

  printf("Value of pchar is %d\n", pchar);
  printf("Value of pshort is %hd\n", pshort);
  printf("Value of plong is %ld\n", plong);

  return 0;
}
```

```
/tmp/NXhWrFFicw.o
Value of pchar is 100
Value of pshort is 200
Value of plong is 300
Value of pchar is 101
Value of pshort is 202
Value of plong is 308
Value of pchar is 91
Value of pshort is 212
Value of plong is 348


=== Code Execution Successful ===
```

Con trỏ *void

- Là con trỏ không định kiểu. Nó có thể trỏ tới bất kì một loại biến nào.
- Thực chất một con trỏ `void` chỉ chứa một địa chỉ bộ nhớ mà không biết rằng tại địa chỉ đó có đối tượng kiểu dữ liệu gì.
  - Do đó không thể truy cập nội dung của một đối tượng thông qua con trỏ `void`.
- Để truy cập được đối tượng thì trước hết phải ép kiểu biến trỏ void thành biến trỏ có định kiểu của kiểu đối tượng

Con trỏ *void

```
float x; int y;
void *p; // khai báo con trỏ void
p = &x; // p chứa địa chỉ số thực x.
*p = 2.5; // báo lỗi vì p là con trỏ void
(Chứa địa chỉ cả x nhưng không biết rằng tại
địa chỉ đó có đối tượng kiểu dữ liệu gì)
/*-> cần phải ép kiểu con trỏ void trước khi
truy cập đối tượng qua con trỏ */
*((float*)p) = 2.5; //-> x = 2.5
//Tương tự
p = &y; // p chứa địa chỉ số nguyên y
*((int*)p) = 2; //-> y = 2
```

Con trỏ *void

```cpp
//pointer_void.cpp
#include<stdio.h>
int main() {
    float x; int y;
    void *p; // khai báo con trỏ void
    p = &x; // p chứa địa chỉ số thực x
    // *p = 2.5; // báo lỗi vì p là con trỏ void
    /* cần phải ép kiểu con trỏ void trước khi truy cập đối tượng qua con trỏ
       */
    *((float*)p) = 2.5; // x = 2.5
    printf("x=%f\n",x);
    printf("&x=%p, p=%p\n",&x,p);
    printf("*(&x)=%f\n",*(&x));
    // p = &x;
    printf("*p=%f\n",*(float *)p);
    p = &y; // p chứa địa chỉ số nguyên y
    *((int*)p) = 2; // y = 2
    printf("y=%d\n",y);
    printf("*p=%d",*(int *)p);
  return 0;
}
```

```
/tmp/tExB44kQnW.o
x=2.500000
&x=0x7ffe69a90ae4, p=0x7ffe69a90ae4
*(&x)=2.500000
*p=2.500000
y=2
*p=2

=== Code Execution Successful ===
```


Con trỏ và mảng

- Giả sử ta có `int a[30];` thì `&a[0]` là địa chỉ phần tử đầu tiên của mảng đó, đồng thời là địa chỉ của mảng.
- Trong C, tên của mảng chính là một hằng địa chỉ bằng địa chỉ của phần tử đầu tiên của mảng

```
a = &a[0];
a+i = &a[i];
```

Con trỏ và mảng

Tuy vậy cần chú ý rằng `a` là 1 hằng nên không thể dùng nó trong câu lệnh gán hay toán tử tăng, giảm như `a++;`

- Xét con trỏ: `int *pa;`

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

Khi đó pa trỏ vào phần tử thứ nhất của mảng và

`pa +1` sẽ trỏ vào phần tử thứ 2 của mảng

`*(pa+i)` sẽ là nội dung của `a[i]`

Con trỏ và chuỗi

- Khai báo:
  - C1: Chuỗi là mảng các ký tự `char s[30] ="Da Lat";`
  - C2: Dựa trên hằng chuỗi (string literal)

```
char *s;
s="Da lat";
```

Hoặc: `char *s ="Da lat";`

- `s` vừa chứa nội dung vừa chứa địa chỉ của chuỗi

```
printf("Noi dung=%s, Dia chi=%p",s,s);
```

- Thao tác tương tự như trên mảng

```
*(s+3) = 'L'
```

tương đương với `s[3] = 'L'`

- Chú ý:
  - Phép thay đổi nội dung chuỗi chỉ thực hiện được trên mảng các ký tự (C1), và không thực hiện được trên hằng chuỗi (C2).
  - Cố tình thay đổi → lỗi `Segmentation fault`

Con trỏ và chuỗi

```cpp
//pointer_vs_string.cpp
#include<stdio.h>
int main() {
    char s[30] ="Da lat"; //Mang cac ky tu
    // char *s =“Da lat”; //Gan bang hang chuoi (string literal)

    printf("Dia chi=%p, Noi dung=%s",s,s);

    *(s+3) = 'L';
    // s[3] = 'L';
    printf("\nDia chi=%p, Noi dung=%s",s,s);

  return 0;
}
```

```
/tmp/BCFHKDc2S9.o
Dia chi=0x7ffe83608ea0, Noi dung=Da lat
Dia chi=0x7ffe83608ea0, Noi dung=Da Lat

=== Code Execution Successful ===
```

```cpp
//pointer_vs_string.cpp
#include<stdio.h>
int main() {
    // char s[30] ="Da lat"; //Mang cac ky tu
    char *s =“Da lat”; //Gan bang hang chuoi (string literal)

    printf("Dia chi=%p, Noi dung=%s",s,s);

    *(s+3) = 'L';
    // s[3] = 'L';
    printf("\nDia chi=%p, Noi dung=%s",s,s);

  return 0;
}
```

```
/tmp/EnFf9HFBOS.o
Segmentation fault


=== Code Exited With Errors ===
```


Mảng các con trỏ

- Con trỏ cũng là một loại dữ liệu nên ta có thể tạo một mảng các phần tử là con trỏ theo dạng thức.

  `<kiểu> *<mảng con trỏ>[số phần tử];`

Ví dụ: `char *ds[10];`

  - `ds` là 1 mảng gồm 10 phần tử, mỗi phần tử là 1 con trỏ kiểu char, được dùng để lưu trữ được của 10 xâu ký tự nào đó

- Cũng có thể khởi tạo trực tiếp các giá trị khi khai báo

  `char *ds[10] = {“mot”,”hai”,”ba”...};`

Mảng các con trỏ

```
//pointer_stringarr_initialize_sort.cpp
#include <stdio.h>
#include <string.h>
#define MAXLEN 5
int main () {
    char *ds[MAXLEN] = {"mot","hai","ba","bon","nam"};
    char *tmp;
    for (int i=0;i < MAXLEN - 1; i++)
        for (int j =i+1;j < MAXLEN; j++)
        if (strcmp(ds[i], ds[j])>0) {
            tmp = ds[i];
               ds[i] = ds[j];
               ds[j] = tmp;
       }
    for (int i=0;i<MAXLEN; i++)
        printf("\n %d : %s", i+1, ds[i]);
    return 0;
}
```


Mảng các con trỏ

- Một ưu điểm khác của mảng trỏ là ta có thể hoán chuyển các đối tượng (mảng con, cấu trúc..) được trỏ bởi con trỏ này bằng cách hoán đổi các con trỏ
- Ưu điểm tiếp theo là việc truyền tham số trong hàm
- Ví dụ: Vào danh sách lớp theo họ và tên, sau đó sắp xếp để in ra theo thứ tự ABC.

Mảng các con trỏ

```
//pointer_stringarr_sortby_pointer.cpp
#include <stdio.h>
#include <string.h>
#define MAXHS 50
#define MAXLEN 30

int main () {
    int cnt = 0, n = 2;
    char ds[MAXHS][MAXLEN];
    char *ptr[MAXHS], *tmp;
    while (cnt < n) {
        printf("Enter the name of student %d: ", cnt + 1);
        fgets(ds[cnt], MAXLEN, stdin);
        if (strlen(ds[cnt]) == 0) break;
        ptr[cnt] = ds[cnt];
        ++cnt;
    }
    for (int i=0;i < n - 1; i++)
        for (int j =i+1;j < n; j++)
            if (strcmp(ptr[i], ptr[j])>0) {
                tmp = ptr[i];
                ptr[i] = ptr[j];
                ptr[j] = tmp;
            }
    for (int i=0;i<n; i++)
        printf("\n %d : %s", i+1, ptr[i]);
    return 0;
}
```


Con trỏ trỏ tới con trỏ

- Bản thân con trỏ cũng là một biến, vì vậy nó cũng có địa chỉ và có thể dùng một con trỏ khác để trỏ tới địa chỉ đó.

```
<Kiểu dữ liệu> ** <Tên biến trỏ>;
```

- Ví dụ:

```
int x = 12;
int *p1 = &x;
int **p2 = &p1;
```

```
1  //pointer2pointer.cpp
2  #include <stdio.h>
3  int main () {
4      int x = 12;
5      int *p1 = &x;
6      int **p2 = &p1;
7      int ***p3 = &p2;
8      int ****p4 = &p3;
9
10     printf("Add of x, p1=%p\n",p1);
11     printf("Add of p1, p2=%p\n",p2);
12     printf("Add of p2, p3=%p\n",p3);
13     printf("Add of p3, p4=%p\n",p4);
14
15     return 0;
16 }
```

```
/tmp/GBGcNOA8QD.o
Add of x, p1=0x7ffdf7813dd4
Add of p1, p2=0x7ffdf7813dc8
Add of p2, p3=0x7ffdf7813dc0
Add of p3, p4=0x7ffdf7813db8


=== Code Execution Successful ===
```

Ma trận được biểu diễn dạng Con trỏ trỏ tới con trỏ

Ví dụ: in ra một ma trận vuông và cộng mỗi phần tử của ma trận với 10

```
#include <stdio.h>
#define rows 3
#define cols 3
int main() {
    int arr[rows][cols] = {{7,8,9}, {10,13,15}, {2,7,8}};
    for (int i = 0; i < rows; i++) {
        for (int j=0; j < cols; j++)
            printf("%d\t", arr[i][j]);
        printf("\n");
    }
    printf("\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            *(*(arr + i) + j) = *(*(arr + i) + j) + 10;
            printf("%d\t", *(*(arr + i) + j));
        }
         printf("\n"); }
}
```


Ma trận được biểu diễn dạng Con trỏ trỏ tới con trỏ (Chi tiết)

matrix_detail.cpp

```
#include <stdio.h>
#define rows 3
#define cols 3
int main() {
    char M[rows][cols] = {{'a','b','c'}, {'d','e','f'}, {'g','h','i'}};
    for (int i = 0; i < rows; i++) {
        for (int j=0; j < cols; j++)
            printf("%c\t", M[i][j]);//%d se in ra ma ASCII
        printf("\n");
    }
    printf("Add cua M:%p",M);
    int i=1,j=2;
    //Dia chi cua hang i
    printf("\nC1: Add hang i, M[i]    =%p",M[i]);
    printf("\nC2: Add hang i, &M[i][0]=%p",&M[i][0]);
    printf("\nC3: Add hang i, M+i     =%p", M+i);

    //Dia chi cua tung phan tu [i][j]
    printf("\nC1: Add &M[i][j]=%p, Value M[i][j]    =%c",&M[i][j],M[i][j]);
    printf("\nC2: Add M[i]+j  =%p, Value *(M[i]+j)  =%c",M[i]+j,*(M[i]+j));
    char *ra_i;
    ra_i=(char *)(M+i);
    printf("\nC3: Add ra_i+j  =%p, Value *(ra_i+j)  =%c",ra_i+j,*(ra_i+j));

    //LUU Y
    //M + i dịch con trỏ đến hàng thứ i.
    //*(M + i) dereference con trỏ, trả về địa chỉ của phần tử đầu tiên trong hàng thứ i (địa chỉ của
        M[i]) -> Giong voi M + i
    //Tuy nhien, muon dich dia chi den cot j trong hang i, co the dung *(M+i)+j, nhung khong the dung (M
        +i)+j. Vi (M+i)+j duoc hieu la M+(i+j) -> nghia la dich den hang (i+j)
    printf("\nAdd *(M+i)+j.   =%p, Add (M+i)+j=%p",*(M+i)+j,(M+i)+j);
    //Voi kieu du lieu char (1 byte), ma tran co 3 cot --> moi hang chiem 3 byte -> (M+i)+j = Add cua M
        + 3*(i+j)

    return 0;
}
```

```
/tmp/iLPQD9Z37j.o
a   b   c
d   e   f
g   h   i
Dia chi cua M:0x7ffc7ac6c4cf
C1: Add hang i, M[i]    =0x7ffc7ac6c4d2
C2: Add hang i, &M[i][0]=0x7ffc7ac6c4d2
C3: Add hang i, M+i     =0x7ffc7ac6c4d2
C1: Add &M[i][j]=0x7ffc7ac6c4d4, Value M[i][j]    =f
C2: Add M[i]+j  =0x7ffc7ac6c4d4, Value *(M[i]+j)  =f
C3: Add ra_i+j  =0x7ffc7ac6c4d4, Value *(ra_i+j)  =f
Add *(M+i)+j.   =0x7ffc7ac6c4d4, Add (M+i)+j=0x7ffc7ac6c4d8

=== Code Execution Successful ===
```

Quản lý bộ nhớ - Bộ nhớ động

- Cho đến lúc này ta chỉ dùng bộ nhớ tĩnh: tức là khai báo mảng, biến và các đối tượng khác một cách tường minh trước khi thực hiện chương trình.
- Trong thực tế nhiều khi ta không thể xác định trước được kích thước bộ nhớ cần thiết để làm việc, và phải trả giá bằng việc khai báo dự trữ quá lớn
- Nhiều đối tượng có kích thước thay đổi linh hoạt

Quản lý bộ nhớ - Bộ nhớ động

- Việc dùng bộ nhớ động cho phép xác định bộ nhớ cần thiết trong quá trình thực hiện của chương trình, đồng thời giải phóng chúng khi không còn cần đến để dùng bộ nhớ cho việc khác
- Trong C: dùng các hàm `malloc, calloc, realloc` và `free` để xin cấp phát, tái cấp phát và giải phóng bộ nhớ.
- Trong C++: dùng `new` và `delete`

Cấp phát bộ nhớ động

- Cú pháp xin cấp phát bộ nhớ:
  - Cho một biến đơn

```
<biến trỏ> = new <kiểu dữ liệu>;
```

  - Cho một biến mảng

```
<biến trỏ> = new <kiểu dữ liệu>[số phần tử];
```

- Giải phóng bộ nhớ

```
delete ptr;
```
// xóa 1 biến đơn

```
delete[] ptr;
```
// xóa 1 biến mảng

Cấp phát bộ nhớ động

- Bộ nhớ động được quản lý bởi hệ điều hành, được chia sẻ giữa hàng loạt các ứng dụng, vì vậy có thể không đủ bộ nhớ. Khi đó toán tử new sẽ trả về con trỏ NULL
- Ví dụ:

```
int *ptr; ptr = new int [200];
if (ptr == NULL) {
    // thông báo lỗi và xử lý
}
```

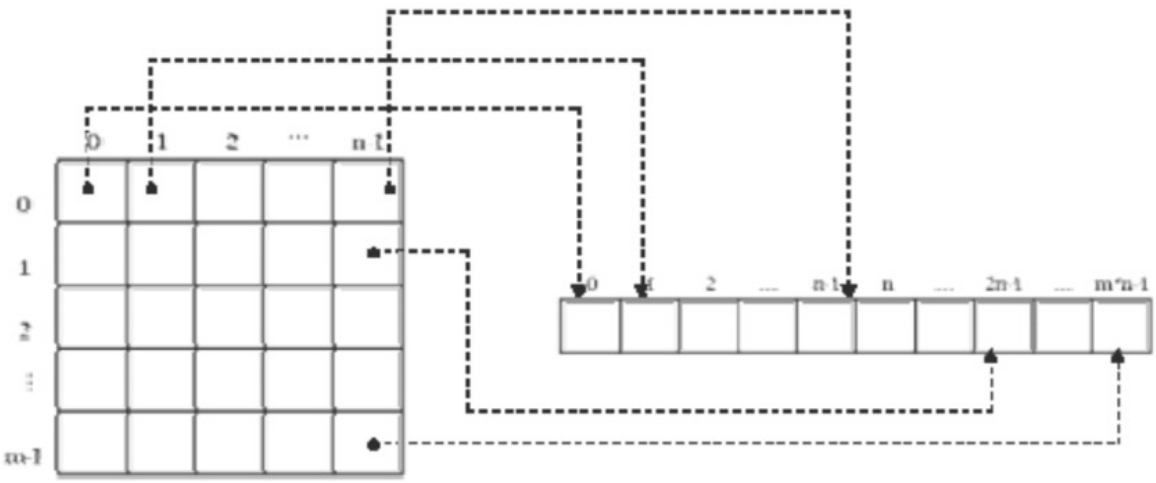
Ví dụ

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n;
    int total=100, x, *arr;
    printf("Nhap n = "); scanf("%d",&n);
    arr = new int [n];
    if (arr==NULL) {
        printf("Cap phat khong thanh cong\n");
        exit(1);
    }
    for (int i=0; i < n; i++){
        printf("\n Vao so thu %d: ", i+1 );
        scanf("%d", &arr[i] );
    }
    printf("Danh sach cac so: \n");
    for (int i=0; i < n; i++) printf("%d ", arr[i]);
    delete[] arr;
    return 0;
}
```


Bộ nhớ động cho mảng 2 chiều

- Cách 1: Biểu diễn mảng 2 chiều thành mảng 1 chiều
- Gọi X là mảng hai chiều có kích thước m dòng và n cột. A là mảng một chiều tương ứng, khi đó

```
X[i][j] = A[i*n+j]
```



Biểu diễn mảng 2 chiều thành mảng 1 chiều

```
#include <stdio.h>

int main() {
    int rows;
    int cols;

    printf("Nhap rows = "); scanf("%d",&rows);
    printf("Nhap cols = "); scanf("%d",&cols);

    int *arr;
    arr = new int [rows*cols];

    for (int i = 0; i < rows; i++) {
        printf("Nhap cac phan tu cua hang %d\n",i);
        for(int j=0; j < cols; j++){
            printf("\tNhap phan tu thu %d: ",j);
            scanf("%d",&arr[i*cols+j]);
        }
    }

    for (int i = 0; i < rows; i++) {
        for (int j=0; j < cols; j++)
            printf("%d\t", arr[i*cols+j]);
        printf("\n");
    }

    return 0;
}
```


Bộ nhớ động cho mảng 2 chiều

- Cách 2: Dùng con trỏ của con trỏ
- Ví dụ: Với mảng số nguyên 2 chiều có kích thước là rows * cols
- Xin cấp phát

```
int **mt;
mt = new int *[rows];
for (int i = 0; i < rows; i++){
   mt[i] = new int[cols];
}
```

- Giải phóng:

```
for (int i = 0; i < rows; i++) {
   delete[] mt[i];
}
delete[] mt;
```

Biểu diễn mảng 2 chiều dùng con trỏ của con trỏ

```
#include <stdio.h>

int main() {
    int rows;
    int cols;

    printf("Nhap rows = "); scanf("%d",&rows);
    printf("Nhap cols = "); scanf("%d",&cols);

    int **mt;
    mt = new int *[rows];

    for (int i = 0; i < rows; i++) {
        mt[i] = new int[cols];
    }

    for (int i = 0; i < rows; i++) {
        printf("Nhap cac phan tu cua hang %d\n",i);
        for(int j=0; j < cols; j++){
            printf("\tNhap phan tu thu %d: ",j);
            scanf("%d",&mt[i][j]);
        }
    }
    for (int i = 0; i < rows; i++) {
        for (int j=0; j < cols; j++)
            printf("%d\t", mt[i][j]);
        printf("\n");
    }

    // Giải phóng bộ nhớ
    for (int i = 0; i < rows; i++) {
        delete[] mt[i];
    }
    delete[] mt;

    return 0;
}
```