Chương 5. Kiểu con trỏ

5.1 Khai báo :

T *P ;

- Giá trị chứa trong P là *địa chỉ* của vùng nhớ chứa giá trị có kiểu **T**,
- Ta cũng nói P là con trỏ trỏ tới vùng nhớ chứa giá trị có kiểu **T**, hay P là con trỏ có kiểu **T**.
- Giá trị NULL : Khi P có giá trị này thì xem như P không trỏ vào đâu cả. Câu lệnh thường dùng

if (P!=NULL) { . . . }

VD :

```
int  *P; // là con trỏ có kiểu int
int x;
x = 10;
P = &x ; // P chứa địa chỉ của biến x. Vùng nhớ được trỏ bởi P là x.
        // Thường được biểu diễn  P●⟶ x
```

5.2 Truy xuất vùng nhớ trỏ bởi con trỏ:

Cho khai báo :

T *P ;

vùng nhớ được trỏ bởi P là *P.

VD :

```
int  *P; // là con trỏ có kiểu int
int x;
x = 10;
P = &x ; // P chứa địa chỉ của biến x.
prinf("%d", *P);  // Viết ra màn hình giá trị của vùng nhớ được trỏ bởi P,
```

tức là x. Trên màn hình là 10.

VD :

```
int  *P; // là con trỏ có kiểu int
int x;
 x = 10;
P = &x ; // P chứa địa chỉ của biến x.
*P = *P + 2; // Thay đổi giá trị của vùng nhớ được trỏ bởi P, tức là x.
prinf("%d", x);  // Trên màn hình là 12
```

Lệnh **scanf(“—”, &x)** : nhập giá trị vào vùng nhớ có địa chỉ là địa chỉ của x, tức là nhập giá trị cho biến x.

VD : Đoạn chương trình sau :

```
int x;
scanf(“%d”, &x);
printf(“%d”, x);
```

tương đương với

```
int x, *P;
P=&x;
scanf(“%d”, P);
printf(“%d”, x);
```

5.3 Mảng và con trỏ:

Cho khai báo mảng :

T M[n] ;

- Ta có một mảng n phần tử kiểu T. **M là con trỏ trỏ tới phần tử đầu tiên của mảng** (chứa địa chỉ của phần tử đầu tiên của mảng).
- (M + i) , i = 0, . . ., n-1, là địa chỉ của phần tử thứ i của mảng. Vậy *(M+i) là M[i].
- Mảng được khai báo như trên còn được gọi là *mảng tĩnh*.

VD :

```
int  M[3];
*(M+0)=10; // ⇔ *M=10 ; ⇔ M[0]=10 ;
*(M+1)=20; // ⇔  M[1]=20 ;
*(M+2)=30; // ⇔  M[2]=20 ;
```

VD :

```
int  M[3], *P;
P=M;
*(P+0)=10; // ⇔  M[0]=10 ;
*(P+1)=20; // ⇔  M[1]=20 ;
*(P+2)=30; // ⇔  M[2]=20 ;
```

VD :

```
int  M[3], *P;
P=M;
P[0]=10; // ⇔  M[0]=10 ;
P[1]=20; // ⇔  M[1]=20 ;
P[2]=30; // ⇔  M[2]=20 ;
```

VD :

```
int  M[3], *P;
M[0]=10;
M[1]=20;
M[2]=30;

P=(M+1); // P trỏ tới M[1]

printf("%d\n", *P); // ⇔ printf("%d\n", P[0]);
```

Chuỗi và con trỏ :

VD :

```
char  S[30], *P;
strcpy(S, “ABCDEFGH”);
P = (S+2);
printf("%s\n", P); // Trên màn màn hình là : CDEFGH
```

VD :

```
char  S[30], *P;
strcpy(S, "ABCDEFGH");
P = (S+2);
strcpy(P, "1234");
printf("%s\n", S); // Trên màn màn hình là : ?
```

5.4 Cấp phát vùng nhớ cho con trỏ :

```
#include <malloc.h>
int main(. . .)
{
        T *P;
        P=(T*)malloc(n*sizeof(T)); (1)
        . . .
}
```

- (1) sẽ cấp phát n ô nhớ có kiểu là T và con trỏ P sẽ trỏ tới ô nhớ đầu tiên,
- P là mảng kiểu T, được gọi là *mảng động*, các phần tử của mảng là P[i], i=0, . ., n-1,
- Nếu không cấp phát được thì P có giá trị NULL.

Ví dụ :

```
#include <malloc.h>
int main(int argc, char* argv[])
{
	int *P;
	P=(int*)malloc(1*sizeof(int)); // ⇔ P=(int*)malloc(sizeof(int));
	*P=10;
	printf("*P= %d\n",*P);			// Trên màn hình : 10

	return 0;
}
```

Ví dụ :

```
#include <malloc.h>
int main(int argc, char* argv[])
{
        int *P;
        P=(int*)malloc(sizeof(int));
        P[0]=10;
        printf("*P= %d\n",*P);                  // Trên màn hình : 10

        return 0;
}
```

Ví dụ :

```
#include <malloc.h>
int main(int argc, char* argv[])
{
	int *P;
	P=(int*)malloc(3*sizeof(int));
	P[0]=10; P[1]=100; P[2]=1000;
	printf("%d, %d, %d\n",P[0], P[1], P[2]); // Trên màn hình : 10, 100, 1000

	return 0;
}
```

Ví dụ :

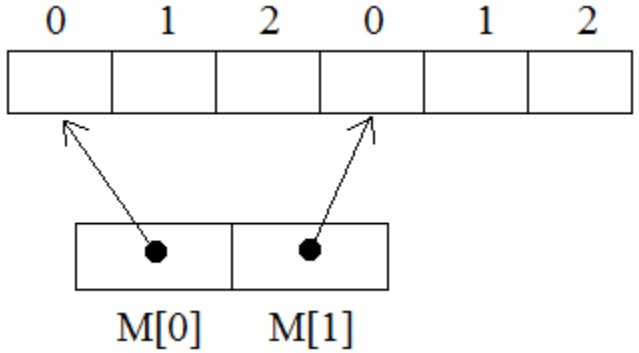
```
#include <malloc.h>
int main(int argc, char* argv[])
{
    int *P;
    P=(int*)malloc(3*sizeof(int));
    P[0]=10; P[1]=100; P[2]=1000;
    printf("%d, %d, %d\n",*P, *(P+1), *(P+2)); // Trên màn hình : 10, 100, 1000

    return 0;
}
```

5.5 Con trỏ và mảng 2 chiều :

Cho khai báo mảng 2 chiều :

int M[2][3] ;

- Ta có (M[0] + i) , i = 0, 1, 2, là địa chỉ của các phần tử M[0][i] (là biến kiểu int). Vậy *(M[0]+i) là M[0][i].
- Tương tự (M[1] + i) , i = 0, 1, 2, là địa chỉ của các phần tử M[1][i], *(M[1]+i) là M[1][i].



➢ Lưu ý : Vì M[0][i] là biến kiểu int nên (M[0]+i) là con trỏ trỏ tới vùng nhớ có kiểu int.

Ví dụ :

```
int M[2][3]={{1,2,3},{10,20,30}};
printf("%d\n", *(M[1]+1));
```

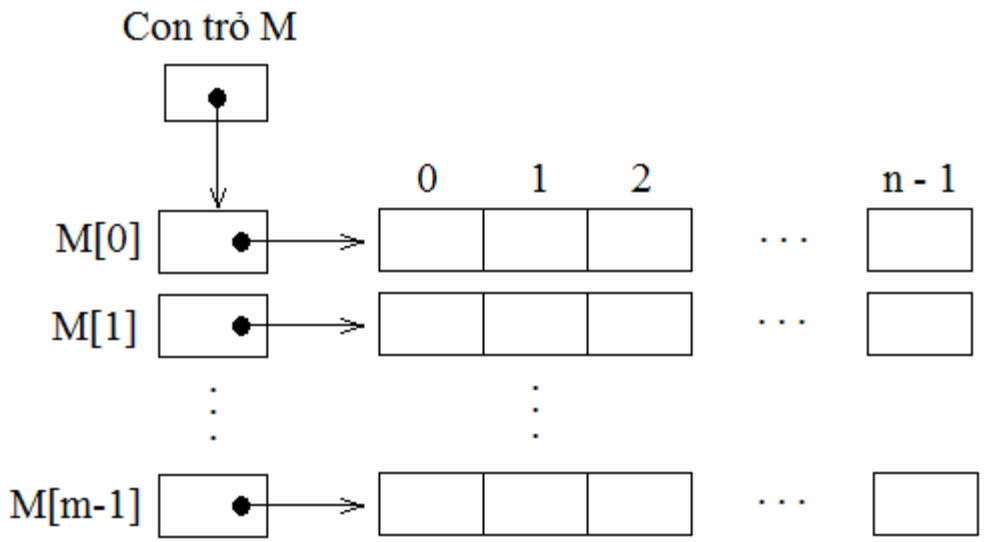
Ví dụ :

```
int M[2][3]={{1,2,3},{10,20,30}};
int *P;

P=(M[1]+1);
*P=*P+100;
printf("%d\n", M[1][1]);
```

Cho khai báo :

T M[m][n];

- M là con trỏ có kiểu con trỏ (có kiểu T).

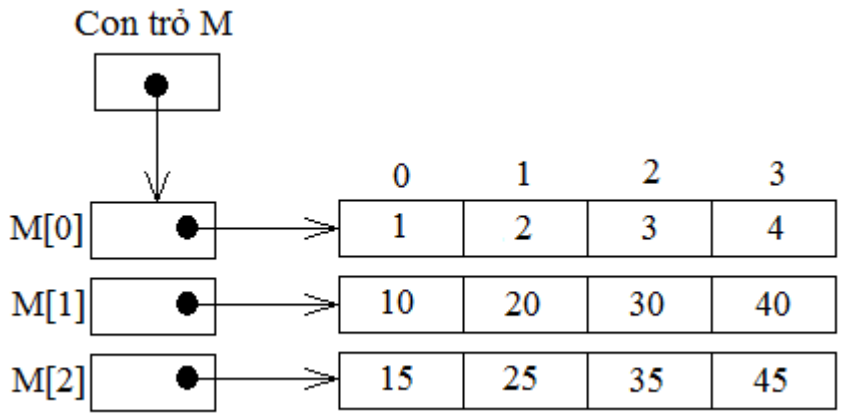


- (M+i), i=0, . ., m-1 là *địa chỉ* của M[i],
- *(M+i) là M[i].
- *(M+i) + j , j=0, . ., n-1 , là địa chỉ của M[i][j],
- *(*(M+i) + j) , j=0, . ., n-1 , là M[i][j].

VD :

```
int M[3][4]={{1,2,3,4},{10,20,30,40}, {15,25,35,45}};
*(*(M+1)+2)=10*(*(*(M+1)+2));
printf("%d\n", M[1][2]);
```

➢ Kết quả : 300

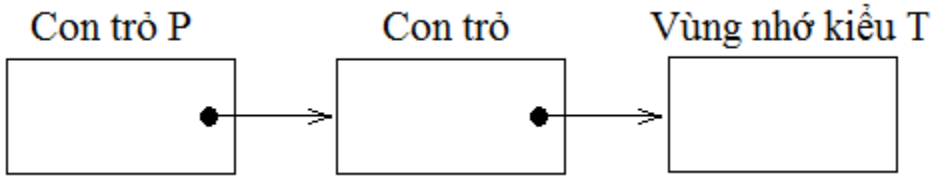


5.6 con trỏ có kiểu con trỏ (kiểu T) :

5.6.1 Khai báo con trỏ có kiểu con trỏ :

T **P;

- P là con trỏ (chứa địa chỉ) trỏ tới vùng nhớ, (vùng nhớ này) là con trỏ trỏ tới vùng nhớ chứa giá trị có kiểu T.



- *P là vùng nhớ được trỏ bởi P, gọi vùng nhớ này là Q, thì **P là *Q, là vùng nhớ kiểu T.

Ví dụ :

```
int x , *Q, **P;
x = 5;
Q = &x;
P = &Q;
printf(“%d\n”, **P);
```

Ví dụ :

```
int x , *Q, **P;
x = 5;
Q = &x;
P = &Q;
**P = **P +1;
printf(“%d\n”, x);
```

5.6.2 Cấp phát vùng nhớ cho *Con trỏ có kiểu con trỏ* (kiểu T) :

Cho khai báo :

T **P;

P=(T**)malloc(m*sizeof(T*)); (1)

P[i] = (T*)malloc(n_i*sizeof(T)); // i = 0, . . ., m-1 (2)

- (1) : P là con trỏ, trỏ tới phần tử đầu tiên của mảng m phần tử. Mỗi phần tử trong mảng có kiểu *con trỏ kiểu T*, P[i],
- (2) : Cấp phát mảng n_i phần tử kiểi T cho P[i] (P[i] trỏ tới phần tử đầu tiên của mảng).

5.6.3 Cấp phát vùng nhớ cho *Con trỏ có kiểu con trỏ* (kiểu T) :

Cho khai báo :

T **P;

P=(T**)malloc(m*sizeof(T*)); (1)

P[i] = (T*)malloc(n_i*sizeof(T)); // i = 0, . . ., m-1

VD :

```
int **P;
P=(int**)malloc(2*sizeof(int*));
P[0] = (int*)malloc(2*sizeof(int)); // ⇔ P[0][0], P[0][1]
P[1] = (int*)malloc(3*sizeof(int)); // ⇔ P[1][0], P[1][1], P[1][2]
*(P[1]+1)=10;
printf("kq = %d\n",P[1][1]);
```

5.7 Mảng các con trỏ kiểu T :

Cho khai báo :

T *M[n];

- M[i], i=0, . . ., n-1, là con trỏ kiểu T,
- M[i] = (T*)malloc(n_i*sizeof(T)); // i = 0, . . ., n-1

VD :

```
int *M[2];
M[0] = (int*)malloc(2*sizeof(int)); // ⇔ M[0][0], M[0][1]
M[1] = (int*)malloc(3*sizeof(int)); // ⇔ M[1][0], M[1][1], M[1][2]
*(M[1]+1)=10;
printf("kq = %d\n",M[1][1]);
```

5.8 Giải phóng vùng nhớ cấp phát cho p :

```
T *p;
p=(T*)malloc(N*sizeof(T));
. . .
free(T);
```