CHƯƠNG 5: CHƯƠNG TRÌNH CON

5.1 Biến:

- **Biến toàn cục** : Là các biến được khai báo ở *vùng toàn cục.*

```
#include "stdafx.h"
Vùng toàn cục
Vùng định nghĩa chương trình con
int main(int argc, char* argv[])
{

        . . .
        return 0;
}
```

- Giá trị biến toàn cục dùng chung cho tất cả các chương con trong vùng *Vùng định nghĩa chương trình con.*

5.2 Chương trình con dạng *thủ tục* :

5.2.1 Khai báo và thực hiện chương trình con dạng 1:

Khai báo :

```
void P( )
{ Khai báo biến cục bộ/ địa phương của P;
  Lệnh ;
}
```

- P là tên của thủ tục đặt theo qui tắc (có phân biệt chữ in và chữ thường):

  Alphabet + Alphabet / chữ số / gạch nối

VD :

- Tên đúng : cong_hai_so, cong_2_so,
- Tên sai : 2_so, cong-2-so

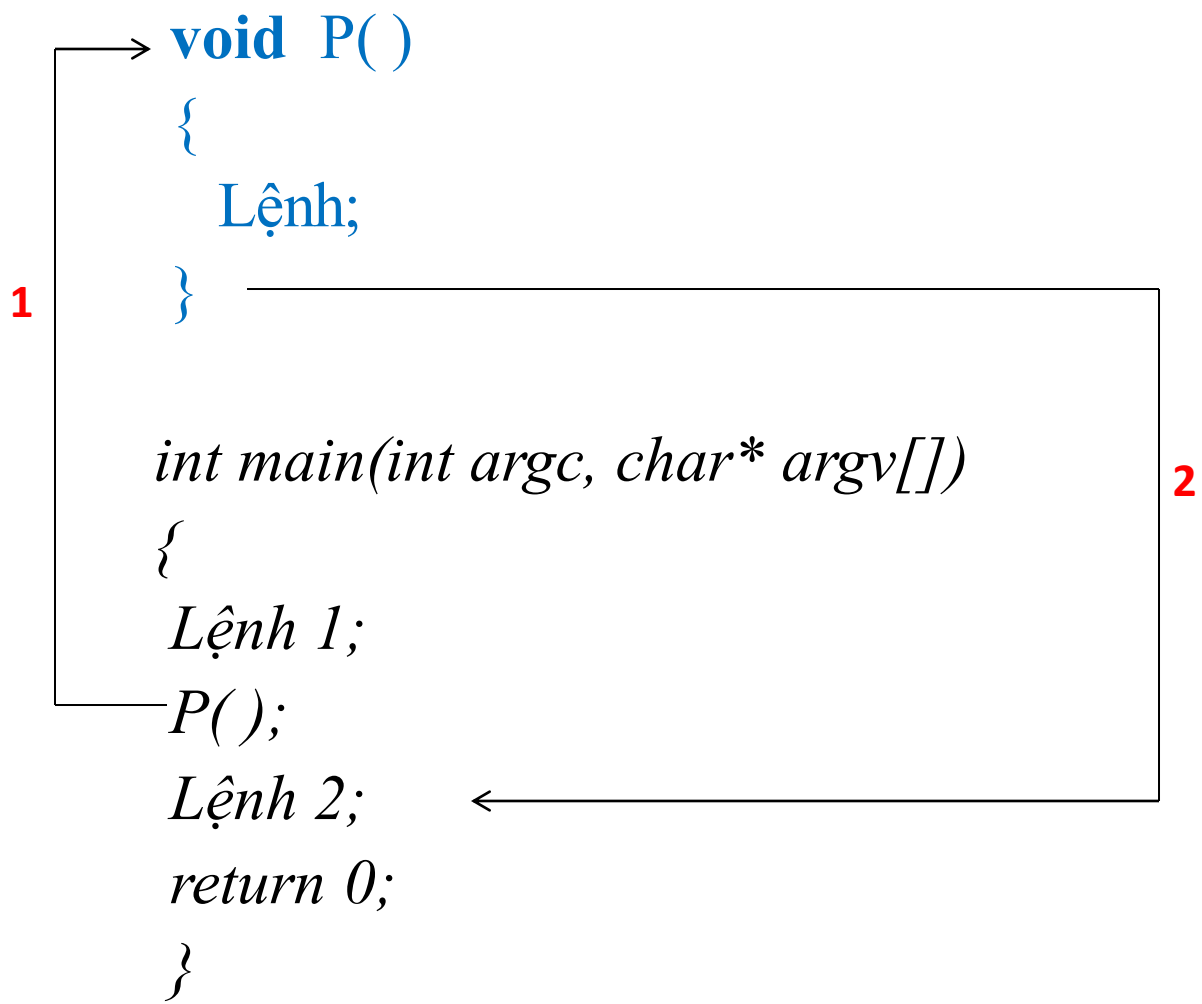
- **Biến cục bộ / địa phương** : Là các biến khai báo ở trong chương trình con.

➢ Giá trị biến cục bộ chỉ sử dụng trong chương trình con khai báo nó.

Chú ý :

- Nếu chương con có ***biến cục bộ cùng tên với biến toàn cục*** thì chương trình con ưu tiên sử dụng biến cục bộ của nó.
- *int main(int argc, char* argv[])* cũng là chương trình con.

Thực hiện :


void P( )
{
Lệnh;
}
1
int main(int argc, char* argv[])
{
Lệnh 1;
P( );
Lệnh 2;
return 0;
}
2

Ví dụ 1 :

```
int x;  // Biến toàn cục

void P()
{
B3. x=x + 20;
}

int main(. . .)
{
B1. x=100;
B2. P();
B4. return 0;
}
```

Giải thích ví dụ 1 :

```
int x;  // Biến toàn cục

      void P()
      {
B3.   x=x + 20;                  // Gán 100 + 20 cho x toàn cục
      }

      int main(. . .)
     {
B1.  x=100;                      // Gán 100 cho x toàn cục
B2.  P();                        // Gọi thủ tục P
B4.  printf("%d", x);            // Trên màn hình : 120
B5.  return 0;
      }
```

Ví dụ 2 :

```
int x ; // x toàn cục

void  P ()
{
B3. x=x + 20;              // Gán giá trị x + 20  cho x toàn cục
}

 int main(. . .)
 {
      int x ;              // biến cục bộ của main( )
B1. x=100 ;                // Gán 100 cho x cục bộ của main()
B2. P() ;                  // Gọi P
B4. printf("%d", x);       // Trên màn hình : 100
B5. return 0;
 }
```

5.2.2 Khai báo và thực hiện chương trình con dạng 2:

Khai báo :

```
void  P(Tx  x , Ty  y  )  // x, y không phải là biến con trỏ
{
   Lệnh ;
}
```

- x, y là các thông số (tham số hay đối số) , được gọi là các *tham trị*.
- x, y là **biến địa phương/cục bộ của P** có kiểu T_x , T_y (int , double, . . .).

Thực hiện :

```
void  P(Tx  x , Ty  y  )
{
B3.  Lệnh ;
}// Kết thúc P
int main(. . .)
{
B1.  Lệnh 1 ;
B2.  P (a, b) ;
B4.  Lệnh 2 ;
}
```

B1. Thực hiện **Lệnh 1**
B2. ***Gán a cho x của P, gán b cho y của P***. Gọi P thực hiện.
B3. Thực hiện **Lệnh**. Kết thúc P.
B4. Thực hiện **Lệnh 2**.

- a, b : Giá trị cụ thể, giá trị của biến , giá trị của biểu thức có kiểu T_x, T_y.

Ví dụ :

```
void P(int x, int y)
{
    int z;
    B3. z=x+y;
    B4. printf("z = %d\n",z);
}

int main(int argc, char* argv[])
{
   int a, b;
   B1. a=5; b=7;
   B2. P(a, b);
   B5. printf("Ket thuc main.\n");
   return 0;
}
```

Ví dụ :

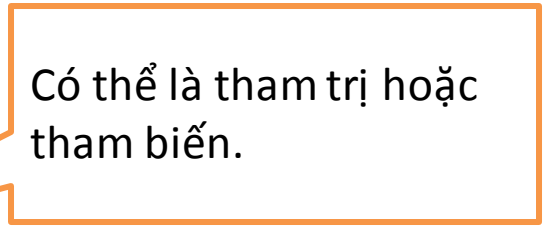
```
void P(int x, int y)
{
    int z;
    z=x+y;
    printf("z = %d\n",z);
}

int main(int argc, char* argv[])
{
   int a;
   a=5;
   P(a, 7);
   printf("Ket thuc main.\n");
   return 0;
}
```

Ví dụ :

```
void P(int x, int y)
{
    int z;
    z=x+y;
    printf("z = %d\n",z);
}

int main(int argc, char* argv[])
{
   int a, b;
   a=5; b=7;
   P(a+3, b);
   printf("Ket thuc main.\n");
   return 0;
}
```

5.2.3 Khai báo và thực hiện chương trình con dạng 3:

Khai báo :




```
void   P(T *x , . . .)
{
   Lệnh;
}
```

- x : Là thông số , là con trỏ có kiểu T, được gọi là các *tham biến* .

Ví Dụ :

```
void P(int *x)
{
        B3. *x=*x+10;
        B4. printf("x = %d\n",*x);
}

int main(int argc, char* argv[])
{
        int a, b;
        B1. a=5;
        B2. P(&a);
        B5. printf("a= %d\n",a);

        return 0;
}
```

Ví dụ : Định nghĩa chương trình con tính tổng 2 số thực.

```
void P(double x, double y , double *T )
{
        B3. *T=x + y;
}

int main(int argc, char* argv[])
{
            double a, b, KQ;
        B1. a=5; b=7;
        B2. P(a,b, &KQ);
        B4. printf("KQ= %lf\n",KQ);
        return 0;
}
```

Bài tập : Hãy cho biết kết quả được viết ra màn hình ở (1) và (2):

```
void P(int *x , int y )
 {
      y = y + *x +1 ;
      *x = *x + y ;
      printf("x = %d,  y = %d", *x, y);
 }

int main()
{
      int a, b;
      a = 2 ; b = 3;
      P (&a, b);
      printf("a = %d,  b = %d", a, b);
}
```

5.2.4 Khai báo và thực hiện chương trình con dạng 4 :

Khai báo :

```
T   P(Tham số)
{
  Lệnh 1;
  . . .
  return Giá tị kiểu T;
  . . .
  Lệnh 2;
}
int main(. . .)
{
  Lệnh 1;
  X = P(các giá trị); (1)
  Lệnh 2;
}
```

- **X=P(Các giá trị) ;** : P được thực hiện như các dạng trước,
- ***return Giá tị kiểu*** **T**; : quay về main *gán* **Giá trị kiểu T** cho X ở (1), thực hiện Lệnh 2 ở main và kết thúc.

Ví dụ :

```
int   Max(int a, int b)
{
   if (a > b) return a;
   else return b;
}
int main(. . .)
{
  int x;
  x = Max(5, 7);
  . . .
}
```

- **X=P(Các giá trị) ;** : P được thực hiện như các dạng trước,
- ***return Giá tị kiểu*** **T**; : quay về main *gán* **Giá trị kiểu T** cho X ở (1), thực hiện Lệnh 2 ở main và kết thúc.

Chú ý : Thứ tự truyền tham số:

```
void  P( x , y , z)
{
    . . .
}

int main()
{
   . . .
   P(a, b, c);
   . . .
}
```

a tương ứng với x

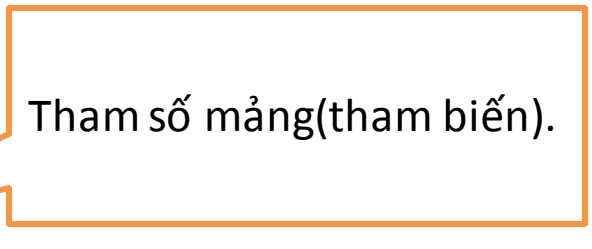
b tương ứng với y

c tương ứng với z

5.3 Chương trình con với tham số là mảng:

5.3.1 Mảng 1 chiều :

Khai báo :




```
void   P(T *x , . . .)
{
   Lệnh;
}
```

Ví dụ :

```
void P(int *x)
{
        x[0]=10; x[1]=11; x[2]=12;
}

int main(int argc, char* argv[])
{
        int M[3];
        P(M);
        printf("%d, %d, %d\n",M[0], M[1], M[2]);

       return 0;
}
```

Kết quả trên màn hình : 10, 11, 12

Ví dụ : Dùng P tính tổng mảng, không cho thay đổi giá trị các phần tử mảng (Khai báo ***const*** *int* **x*).

```
void P(const int *x, int *T)
{
        *T = x[0]+x[1]+x[2];
}

int main(int argc, char* argv[])
{
    int M[3]={10, 20, 30}, KQ;

    P(M, &KQ);
    printf("%d, %d, %d, Sum = %d\n",M[0], M[1], M[2], KQ);
    return 0;
}
```

Kết quả trên màn hình : 10, 20, 30, Sum = 60

Ví dụ :

```
void P(int x[]) // ⇔ int *x
{
        x[0]=10; x[1]=11; x[2]=12;
}

int main(int argc, char* argv[])
{
        int M[3];

        P(M);
        printf("%d, %d, %d\n",M[0], M[1], M[2]);
        return 0;
}
```

Ví dụ :

```
void P(int x[])
{
        x[0]=10; x[1]=11; x[2]=12;
}

int main(int argc, char* argv[])
{
        int *M;  // Sử dụng mảng động

        M=(int*)malloc(3*sizeof(int));
        P(M);
        printf("%d, %d, %d\n",M[0], M[1], M[2]);

        return 0;
}
```

Ví dụ :

```
void P(int *x)
{
        x=(int*)malloc(3*sizeof(int));
        x[0]=10; x[1]=11; x[2]=12;

}
int main(int argc, char* argv[])
{
        int M[3]={100, 110,120} ;
        P(M);
        printf("%d, %d, %d\n",M[0], M[1], M[2]);
        return 0;
}
```

Kết quả trên màn hình : 100, 110, 120

Ví dụ : Dùng P cấp phát vùng nhớ và gán giá trị cho M.

```
void P(int **x)
{
    *x=(int*)malloc(3*sizeof(int));
    (*x)[0]=10; (*x)[1]=11; (*x)[2]=12;
}
int main(int argc, char* argv[])
{
        int *M;
        P(&M);
        printf("%d, %d, %d\n",M[0], M[1], M[2]);
        return 0;
}
```

Chú ý :

- (*x)[0]=10 ≠ *x[0]=10
- Trong VD (*x) là M và (*x)[0] là M[0],
- *x[0] có nghĩ là x[0] là con trỏ, trong VD đang xét nó không tồn tại.

5.3.2 Mảng 2 chiều :

Khai báo :

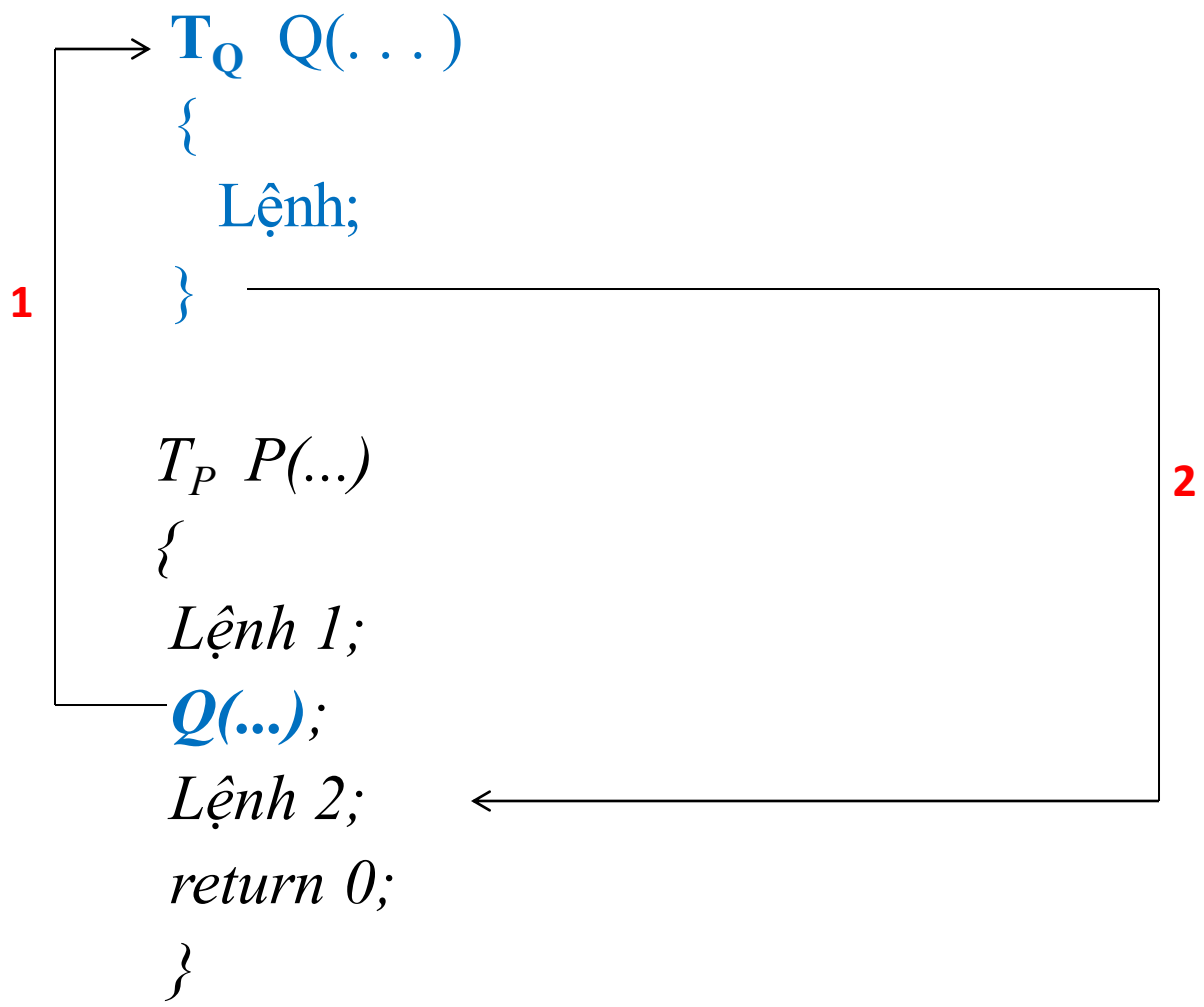
```
void   P(T x[ ][n] , . . .)
{
   Lệnh;
}
```

- n là hằng nguyên ≥ 1.

Ví dụ :

```
void P(int x[ ][3])
{
    x[0][0]=10; x[0][1]=11; x[0][2]=12;
    x[1][0]=20; x[1][1]=21; x[1][2]=22;
}

int main(. . .)
{
    int M[2][3];
    P(M);
    printf("%d, %d, %d \n", M[0][0], M[0][1], M[0][2]);
    printf("%d, %d, %d \n", M[1][0], M[1][1], M[1][2]);
}
```

5.4 Chương trình con gọi chương trình con :



Ví dụ :

```
void max_2so(int a1, int b1, int *kq1)
{ *kq1 = a1; if (b1 > a1) *kq1 = b1; }

void max_3so(int a2, int b2, int c2, int *kq2)
{ int t;
  2. max_2số(a2, b2, &t);
  3. max_2so(t, c2, kq2); // max_2so(t, c2, &kq2) : SAI
 }
void main()
{ int kq;
  1. max_3so(1, 2, 3, &kq);
}
```

5.4 Gọi đệ qui :

```
T_P  P(...)
{
 Lệnh 1;
 P(...);
 Lệnh 2;
 return 0;
 }
```

Ví dụ : Tính tổng mảng a có n số nguyên. Thực hiện :

Bước 1. Tính tổng T = a[0] + a[1] + . . . + a[n-2]

Bước 2. kq = T + a[n-1]

Nhận xét : Nếu gọi *void tong_mang(int a[], int m, int *T)* là chương trình con tính tổng a[0] + a[1] + . . . + a[m-1] chứa kết quả vào T thì Bước 1 sẽ là :

Bước 1. tong_mang(a, n-1, &T)

Ví dụ : Tính tổng mảng a có n số nguyên. Thực hiện :
Bước 1. Tính tổng T = a[0] + a[1] + . . . + a[n-2]
Bước 2. kq = T + a[n-1]

```
void tong_mang(int  a[], int m, int *T)
{
    if (m > 0) {
        tong_mang(a, m-1, T);
        *T = *T + a[m-1];
    }
    else *T=0;
}
void main()
{  int kq, a[4] = {1, 2, 3, 4};
   tong_mang(a,  4, &kq);
   printf(kq);
}
```

Ví dụ : Tính tổng mảng a có n số nguyên. Thực hiện :

```
int  tong_mang(int  a[], int m)
{    int T ;
     if (m > 0) {
        T = tong_mang(a,  m-1);
        T = T + a[m-1];
        return    T;
     }
     else  return  0;
}
void main()
{  int kq, a[4] = {1, 2, 3, 4};
   kq = tong_mang(a,  4);
   printf(kq);
}
```

⇔ return *tong_mang(a, m-1)* + a[m-1];

Bài tập : Tìm max của mảng a có n số nguyên.

HD :

Bước 1. Tìm max của a[0] , a[1] , . . . , a[n-2], lưu kết quả vào T

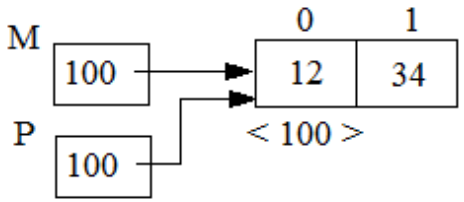
Bước 2. Tìm max của T và a[n-1].

Gọi *void max_mang(int a[], int m, int *T)* là thủ tục tìm max của a[0] , a[1] , . . . , a[m-1], lưu kết quả vào T.

5.5 Con trỏ là tham số của chương trình con :

Trường hợp 1 :

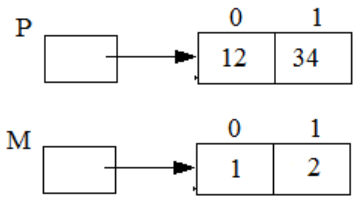
```
void CT_con(int *P)
{
        P[0]=12;
        P[1]=34;
}
```




```
int main(int argc, char* argv[])
{
        int M[2] = {1, 2};

        CT_con(M);

        printf("M[0]=%d", M[0]);
        printf("M[1]=%d", M[1]);
}
```

Trường hợp 2 :

```
void CT_con (int *P)
{
   P=(int*)malloc(2*sizeof(int));

   P[0]=12;
   P[1]=34;
}
```




```
int main(int argc, char* argv[])
{
  int M[2] = {1,2};

   CT_con (M);

   printf("M[0]=%d", M[0]);
   printf("M[1]=%d", M[1]);
}
```

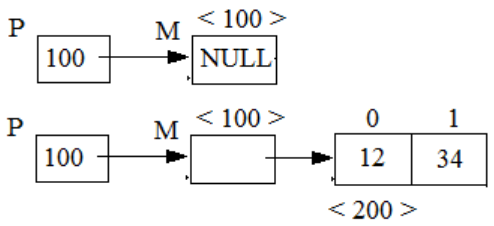
Trường hợp 3 : Cấp phát vùng nhớ cho M bằng chương trình con. M là con trỏ.

```
void CT_con (int *P)
{
   P=(int*)malloc(2*sizeof(int));

   P[0]=12;
   P[1]=34;
}
```

```
int main(int argc, char* argv[])
{
  int *M;

   CT_con (M);   // Error

   printf("M[0]=%d", M[0]);
   printf("M[1]=%d", M[1]);
}
```

Trường hợp 4 : Cấp phát vùng nhớ cho M bằng chương trình con. M là con trỏ.

```
void CT_con (int **P)
{
   *P=(int*)malloc(2*sizeof(int));

   (*P)[0]=12;
   (*P)[1]=34;
}
```




```
int main(int argc, char* argv[])
{
  int *M;

   CT_con (&M);

   printf("M[0]=%d", M[0]); //  12
   printf("M[1]=%d", M[1]); //  34
}
```

Trường hợp 5 : Dùng C++ cấp phát vùng nhớ cho M bằng chương trình con. M là con trỏ.

```
void PCT_con (int*  &P)
{
   P=(int*)malloc(2*sizeof(int));

   P[0]=12;
   P[1]=34;
}
```

```
int main(int argc, char* argv[])
{
  int *M;

   CT_con (M);

   printf("M[0]=%d", M[0]); //  12
   printf("M[1]=%d", M[1]); //  34
}
```

Trường hợp 5 : Dùng C++ cấp phát vùng nhớ cho M bằng chương trình con. M là con trỏ.

```
typedef   int*  INT_PTR;

void CT_con (INT_PTR &P)
{
   P=(INT_PTR)malloc
                          (2*sizeof(int));
//  P=(int*)malloc(2*sizeof(int));

   P[0]=12;
   P[1]=34;
}
```

```
int main(int argc, char* argv[])
{
  INT_PTR   M;  // int *M;

   CT_con (M);

   printf("M[0]=%d", M[0]); //  12
   printf("M[1]=%d", M[1]); //  34
}
```

Trường hợp 6 : Dùng C++ cấp phát vùng nhớ cho M bằng chương trình con. M là con trỏ.

```
void CT_con (int **&P)
{
   P=(int**)malloc(2*sizeof(int*));
   P[0]=(int*)malloc(3*sizeof(int));
   P[1]=(int*)malloc(3*sizeof(int));
}
```

```
int main(int argc, char* argv[])
{
   int **M;

    CT_con (M);
    . . .
}
```

Trường hợp 6 : Dùng C++ cấp phát vùng nhớ cho M bằng chương trình con. M là con trỏ.

```
typedef   int**  INT_PTR;

void CT_con (INT_PTR &P)
{
   P=(INT_PTR)malloc(2*sizeof(int*));
// P=(int**)malloc(2*sizeof(int*));
   P[0]=(int*)malloc(3*sizeof(int));
   P[1]=(int*)malloc(3*sizeof(int));
}
```

```
int main(int argc, char* argv[])
{
   int **M;  //  INT_PTR   M;

    P(M);
    . . .
}
```