Chương II

trong KTLT

(6LT – 2BT)

Chương II

1. nh
2. c
3. ng
4. ng

c CT

- t.
- nh con.
- c *i ND.*
- c *c*.

p)

- ng:
  - m
  - c
-

  .
-

  o.
- ch chung .

p)

- m trong C

```
int GT(int n)
{ if ( n==0) return 1;
  else return n * GT(n-1);
 }
```

- c trong C

```
void nhapmang(Mang V, int n)
{int i;
 for (i=0;i <n;i++)
   { printf("\n V[ %d ]=",i);
     scanf("%d",&V[i]);
     }
 }
```

p)

-
- >
- cuu duong than cong . com
- .
- void main(void) == main()
- cuu duong than cong . com
- .

c

- i

  CT con.

- a CT con.

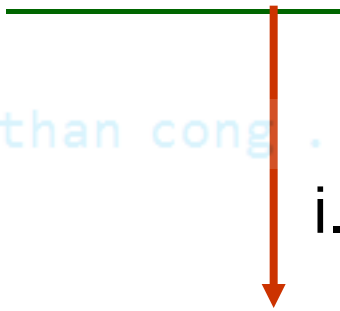
c

```
int GT(int n)
    { if ( n==0) return 1;
      else return n * GT(n-1);
     }
    ….
    Printf(“\3!= %d”,GT(3));
```


c

i

- c (called) “ ”
  c.
- n trong LT).
- :
  - (by val): c



i.

  - c!!!

p)

-
  c.
  i (– side effect).
- => C c!!!
- TD:
- n theo tên (macro)
-
  n sau).

```
#include <stdio.h>
#include <conio.h>
swap (int a, int b);
{  int temp = a;
   a = b;
   b= temp);
}
main()
{  int c = 5;
   int d = 7;
   clrscr();
   swap (c,d);
   printf("\n c= %d   d=%d",c,d);
   getch();
   }
```

in ra

c = 5 d = 7

```
#include <stdio.h>
#include <conio.h>
swap (int *a, int *b);
{  int temp = *a;
   *a = *b;
   *b= temp;
}
main()
{  int c = 5;
   int d = 7;
   clrscr();
   swap (&c,&d);
   printf("\n c= %d   d=%d",c,d);
   getch();
   }
```

in ra

c = 7 d = 5

Tổ chức CT con

- Để tiện sử dụng CT con được tổ chức theo nhiều hình thức khác nhau:

  1. *Trong cùng 1 chương trình với CT chính*
  2. *Ghép thành đơn vị CT*
  3. *Ghép thành mô đun (đơn thể chương trình)*

- Cách tổ chức thứ 2 và 3 tiện dụng hơn vì tính tái sử dụng. Các CT con của ND có thể chuyển vào Thư Viện chương trình của NNLT đó (các NNLT đều có công cụ hỗ trợ việc này.
- TD: minh họa qua C/ Pascal.

Chương II

1. nh
2. c
3. i
4.

ng

c (global)

• Khi 1 CT được gọi nó được nạp vào bộ nhớ và thường trú trong bộ nhớ đến khi kết thúc thực hiện. Đó chính là vòng đời của CT => Do vậy các đại lượng định nghĩa trong CT đó cũng kết thúc vòng đời của mình. => Nảy sinh khái niệm biến cục bộ và biến toàn cục.

• Biến cục bộ (local variables): Các biến được định nghĩa trong 1 CT và chỉ được sử dụng trong CT con đó. *Nó có cùng vòng đời với CT sinh ra nó*. Khái niệm cục bộ cũng là tương đối và phụ thuộc vào cách tổ chức CT. Nó là cục bộ của CT con đó song là toàn cục với CT con của nó.

c

(tiếp)

- Biến toàn cục (global variables): Các biến được định nghĩa trong 1 CT và được sử dụng trong CT con đó và các CT con của nó. => Đ/n ở 1 nơi và sử dụng ở nơi khác.

- Một loại nữa là *static*: Nó là cục bộ của 1

.

- Chú ý cách dùng

Thí dụ

```
int V[…];
void inmang(int n)
 {
  int i;
     for (i=0;i <n;i++)
    printf(" %4d",V[i]);
    }
main ()
{int i,j,n, tam;
 clrscr();
 printf(" So Phan tu cua mang:\n");
 scanf("%d",&n);
 nhapmang(n);
 printf("\n Day so vua nhap:");
 inmang(n);
}
```

a CT con inmang

a main

i CT con!!!

Chương II

1.
2.
3.
4.

i

- . TD:

1.

ng trên.

- i
- c Union

```
 struct WORDREGS {
   unsigned int    ax, bx, cx, dx, si, di, cflag, flags;
};

struct BYTEREGS {
   unsigned char   al, ah, bl, bh, cl, ch, dl, dh;
};

union   REGS    {
   struct  WORDREGS x;
   struct  BYTEREGS h;
};
```

p)

```
Typedef union {
    struct {  long abscisse ;
              long ordonne;
            }cart;
    struct {
              float rho;
              float theta;
            }pol;
```


p2
m:

P1.cart.abscisse, p2.pol.theta

p (file)

- p.

- y).

- c nhau.

- phân (binary file).

phân

```
void main()
 { int i , j, n;
   mang A, B;
   FILE *fp;
   char filename[]="Mang.Txt";
   clrscr();
   if ((fp=fopen(filename,"w+"))==NULL)
      printf("\n Khong mo duoc tep!");
   else
   { do
      { printf("\n so phan tu (1 <n <10)");
      scanf("%d",&n);
      } while ((n < 1) ||(n>10));
     printf("\n nhap cac phan tu:");
     for (i =0;i<n;i++)
       for (j=0;j<n;j++)
           {printf("\n A[%d,%d]=",i,j);
      scanf("%d",&A[i][j]);
      }
     fwrite(&A,sizeof(int), MAX * MAX,fp);
```

phân

```
if ((fp=fopen(filename,"r+"))==NULL)
     printf("\n Khong mo duoc tep!");
    else
    { fread(&B,sizeof(int), MAX * MAX,fp);
      fclose(fp);
    }// ket thuc else
      printf("\n Mang doc ra tu tep:");
      for (i = 0;i<n;i++)
    { for (j = 0;j<n;j++)      printf("\ %d  ",B[i][j]);
      printf("\n");
    }
```

n

```
void main()
 { char c;
   FILE *fv, *fr;
   char *filename="D_ghitep.cpp";
   clrscr();
   if ((fv=fopen(filename,"r+"))==NULL)
      printf("\n Khong mo duoc tep!");
   else
   {
      filename= "D_ghitep.Txt";
      if ((fr = fopen(filename,"w+")) == NULL);
      do { c = fgetc(fv);
            fputc(c,fr);
        } while (c != EOF);
      fclose(fv);
      fclose(fr);  }
```

Chương II

1. nh
2. c
3. i
4. ng

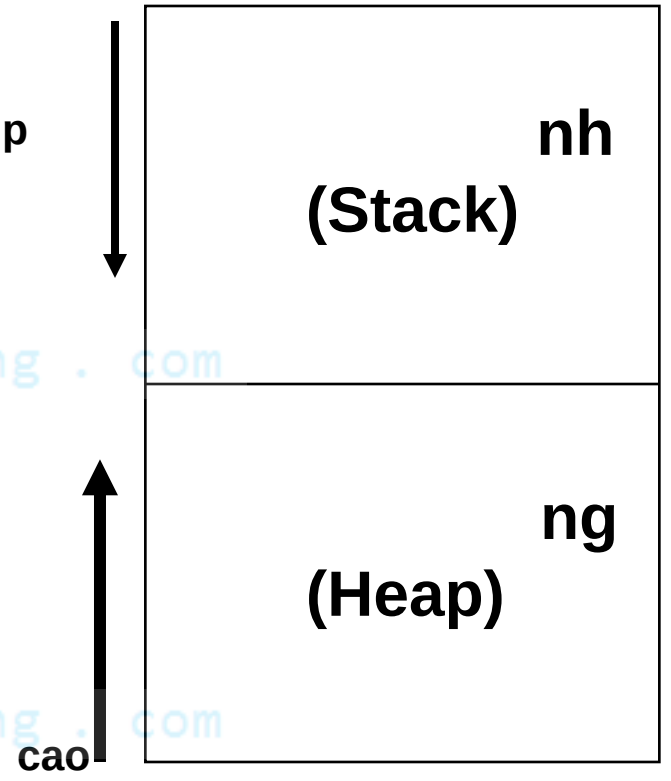
n

- c). Đ c

  n lên.

- c).

- Stack.



nh

-

  như khai báo mảng, biến và các đối tượng 1 cách tường minh.
-

  nhiều đối tượng có kích thước thay đổi linh hoạt.
-

  n.

  o trong CT).

t động

- t.
- riêng:
  - Trong C ta dùng các hàm **malloc**, **calloc**, **realloc** và **free** để xin cấp phát, tái cấp phát và giải phóng bộ nhớ. Trong C++ là **new** và **delete**.
  - u).
  - o heap overflow.

sung trong C/C++

delete

- Để xin cấp phát bộ nhớ ta dùng :

<biên trỏ> = new <kiểu dữ liệu>;

hoặc <biến trỏ> = new <kiểu dữ liệu>[số ftử];

dòng trên xin cấp phát một vùng nhớ cho một biến đơn, còn dòng dưới : cho một mảng các phần tử có cùng kiểu với kiểu dữ liệu.

- Bộ nhớ động được quản lý bởi hệ điều hành, và với môi trường đa nhiệm (multitask interface) thì bộ nhớ này sẽ 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 *pds;

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

Giải phóng bộ nhớ

- delete ptr; // xóa 1 biến đơn
- delete [] ptr; // xóa 1 biến mảng
- ví dụ : #include <iotream>

```
int main() {
    int i,n; long total=100,x,*l;
    cout << "Vao so ptu ";          cin >> n;
    l = new long [n];
    if (l==NULL) exit(1);
    for (i=0;i<n;i++){
        cout <<"\n Vao so thu "<< i+1 <<" :";    cin >> l[i]  }
    Cout << "Danh sach cac so : \n"
    for (i=0;i<n;i++)   cout << l[i] << ",";
    delete []l;
    return 0;
}
```

p 1

- n.
- nh
  - t
  -
  -
  - n
-). c C++.

p 2

- ch.
-).
- c

cuu duong than cong . com

c nâng cao trong C/ C++

1. Mảng

- Là một dãy hữu hạn các phần tử liên tiếp có cùng kiểu và tên
- Có thể là 1 hay nhiều chiều, C không giới hạn số chiều của mảng
- Khai báo theo cú pháp sau :

  DataType ArrayName [size];

  hooặc DataType ArrayN [Size1][Size2]...;

- Khởi tạo giá trị cho mảng theo 2 cách:
  - Khi khai báo :

    float y[5]={3.2, 1.2, 4.5 ,6.0, 3.6}

    int m[6][2] = {{1,1},{1,2},{2,1},{2,2},{3,1},{3,2}};

    char s1[6] ={'H','a','n','o','i','\0'}; hoặc

    char s1[6] = "Hanoi";

    char s1[] ="Dai hoc Bach Khoa Hanoi"; L=24

    int m[][] ={{1,2,3},{4,5,6}};
  - Khai báo rồi gán giá trị cho từng phần tử của mảng.

    Ví dụ : int m[4];

    m[0] = 1; m[1] = 2; m[2] = 3; m[3] = 4;

2. Con trỏ

- Khái niệm : Giá trị các biến được lưu trữ trong bộ nhớ MT, có thể truy cập tới các giá trị đó qua tên biến, đồng thời cũng có thể qua địa chỉ của chúng trong bộ nhớ (CTDL&GT).
- Con trỏ thực chất là 1 biến mà nội dung của nó là địa chỉ của 1 đối tượng khác (biến, hàm, nhưng không phải 1 hằng số).
- Có nhiều kiểu biến với các kích thước khác nhau, nên có nhiều kiểu con trỏ. Con trỏ **int** để trỏ tới biến hay hàm kiểu **int,…**
- Việc sử dụng con trỏ cho phép ta truy nhập tới 1 đối tượng gián tiếp qua địa chỉ của nó.
- Trong C, con trỏ là một công cụ rất mạnh, linh hoạt.

- Khai báo con trỏ :
- Cú pháp : dataType * PointerName;

  Chỉ rằng đây là con trỏ trỏ về kiểu dataType.
- Sau khi khai báo, ta được con trỏ NULL (chưa trỏ tới 1 đối tượng nào).
- Để sử dụng con trỏ, ta dùng toán tử lấy địa chỉ &

  PointerName = & VarName

  Ví dụ :

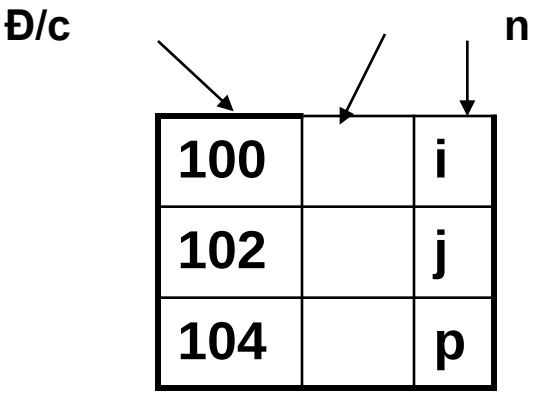
  int a; int *p; a=10;

  p= &a; => *p = 10
- Để lấy nội dung biến do con trỏ trỏ tới, ta dùng toán tử lấy nội dung *:

  * PointerName

Ví dụ :

```
int i,j, *p;
i= 5;          p= & i;
j= *p;         *p= j+2;
```



Gán i=5

100	5	i
102		j
104		p

gán p = & i

100	5	i
102		j
104	100	p

gán j = *p

100	5	i
102	5	j
104	100	p

***p = j+2**

100	7	i
102	5	j
104	100	p

Chú ý

- Một con trỏ chỉ có thể trỏ tới 1 đối tượng cùng kiểu.
- Toán tử 1 ngôi * và & có độ ưu tiên cao hơn các toán tử số học.
- Ta có thể viết *p mọi nơi có đối tượng mà nó trỏ tới xuất hiện.

```
int x = 5, *p; p = &x; =>
x=x+10;  ~ *p = *p+10;
```

- Ta cũng có thể gán nội dung 2 con trỏ cho nhau : khi đó cả hai con trỏ cùng trỏ tới 1 đối tượng.

```
int x=10, *p, *q;
p = &x;          q = p;
```

=> p và q cùng trỏ tới x

Các phép toán trên con trỏ

- **Một biến trỏ có thể cộng hoặc trừ với 1 số nguyên n để cho kết quả là 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**
- **Phép trừ giữa 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, nhưng cần chú ý đến sự tương thích về kiểu.**

Ví dụ : char *pchar; short *pshort; long *plong;

⇒ sau khi xác lập địa chỉ cho các con trỏ, nếu :

pchar ++; pshort ++; plong ++; và các địa chỉ ban đầu tương ứng của 3 con trỏ là 100, 200 và 300, thì kết quả ta có các giá trị tương ứng là : 101, 202 và 304 tương ứng.

- Nếu viết tiếp :

  plong += 5; => plong = 324 (304 + 5 x 4)

  pchar -=10; => pchar = 91

  pshort +=5; => pshort = 212

- Chú ý : ++ và – có độ ưu tiên cao hơn * => *p++ ~ *(p++) tức là tăng địa chỉ mà nó trỏ tới chứ không phải tăng giá trị mà nó chứa.
- *p++ = *q++ sẽ tương đương :

  *p = *q;

  p=p+1;

  q=q+1;

Vì cả 2 phép tăng đều diễn ra sau khiphép gán được thực hiện

=> Cần dùng dấu () để tránh nhầm lẫn

Con trỏ void*

- Là con trỏ không định kiểu (void *). 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. => 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 con trỏ void về con trỏ có định kiểu của kiểu đối tượng.


```
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
p = &y;              // p chứa địa chỉ số nguyên y
*((int*)p) = 2;    // y = 2
```

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 1 hằng địa chỉ = địa chỉ của phần tử đầu tiên của mảng.

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

  a+i = &a[i];

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

  Xét con trỏ : int *pa;

  pa = & a[0];

=> pa trỏ vào ftử 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 xâu

- Ta có : char tinhthanh[30] =“Da lat”;
- Tương đương : char *tinhthanh;
- tinhthanh=“Da lat”;
- Hoặc : char *tinhthanh =“Da lat”;
- Ngoài ra các thao tác trên xâu cũng tương tự như trên mảng
- *(tinhthanh+3) = “l”
- Chú ý : với xâu thường thì không thể gán trực tiếp như dòng thứ 3

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>];

- vd : char *ds[10];

⇒

của 10 xâu ký tự nào đó.

- Cũng có thể khởi tạo trực tiếp các giá trị khi khai báo
- char * ma[10] = {“mot”,”hai”,”ba”...};
- Chú ý : cần phân biệt mảng con trỏ và mảng nhiều chiều. Mảng nhiều chiều là mảng thực sự được khai báo và có đủ vùng nhớ dành sẵn cho các ftử. Mảng con trỏ chỉ dành không gian nhớ cho các biến trỏ (chứa địa chỉ). Khi khởi tạo hay gán giá trị : cần thêm bộ nhớ cho các ftử sử dụng => tốn nhiều hơn.

- 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 chuyển các con trỏ
- Ưu điểm tiếp theo là việc truyền tham số trong hàm
- Ví dụ : Vào ds lớp theo họ và tên, sau đó sắp xếp để in ra theo thứ tự ABC.

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

```
int main () {
  int  i, j, count = 0;   char ds[MAXHS][MAXLEN];
  char *ptr[MAXHS],  *tmp;
  while ( count < MAXHS) {
      printf(“ Hoc sinh thu : %d  “,count+1);
      gets(ds[count]);
      if (strlen(ds[count] == 0) break;
      ptr[count] = ds +count;
      ++count;
   }
   for ( i=0;i<count-1;i++)
      for ( j =i+1;j < count; j++)
         if (strcmp(ptr[i],ptr[j])>0) {
               tmp=ptr[i]; ptr[i] = ptr[j]; ptr[j] = tmp;
   }
   for (i=0;i<count; i++)
      printf(“\n %d :  %s”, i+1,ptr[i]);
}
```

Con trỏ trỏ tới con trỏ

- Bản thân con trỏ cũng là 1 biến, vì vậy nó cũng có địa chỉ và có thể dùng 1 con trỏ khác để trỏ tới địa chỉ đó.
- <Kiểu DL> **<Tên biến trỏ>;
- Ví dụ : int x = 12;

  int *ptr = &x;

  int **ptr_to_ptr = &ptr;
- Có thể mô tả 1 mảng 2 chiều qua con trỏ của con trỏ theo công thức :

ArrName[i][k] = *(*(ArrName+i)+k)

Với ArrName+i là địa chỉ của phần tử thứ i của mảng

*(ArrName+i) cho nội dung ftử trên

*(ArrName+i)+k là địa chỉ phần tử [i][k]

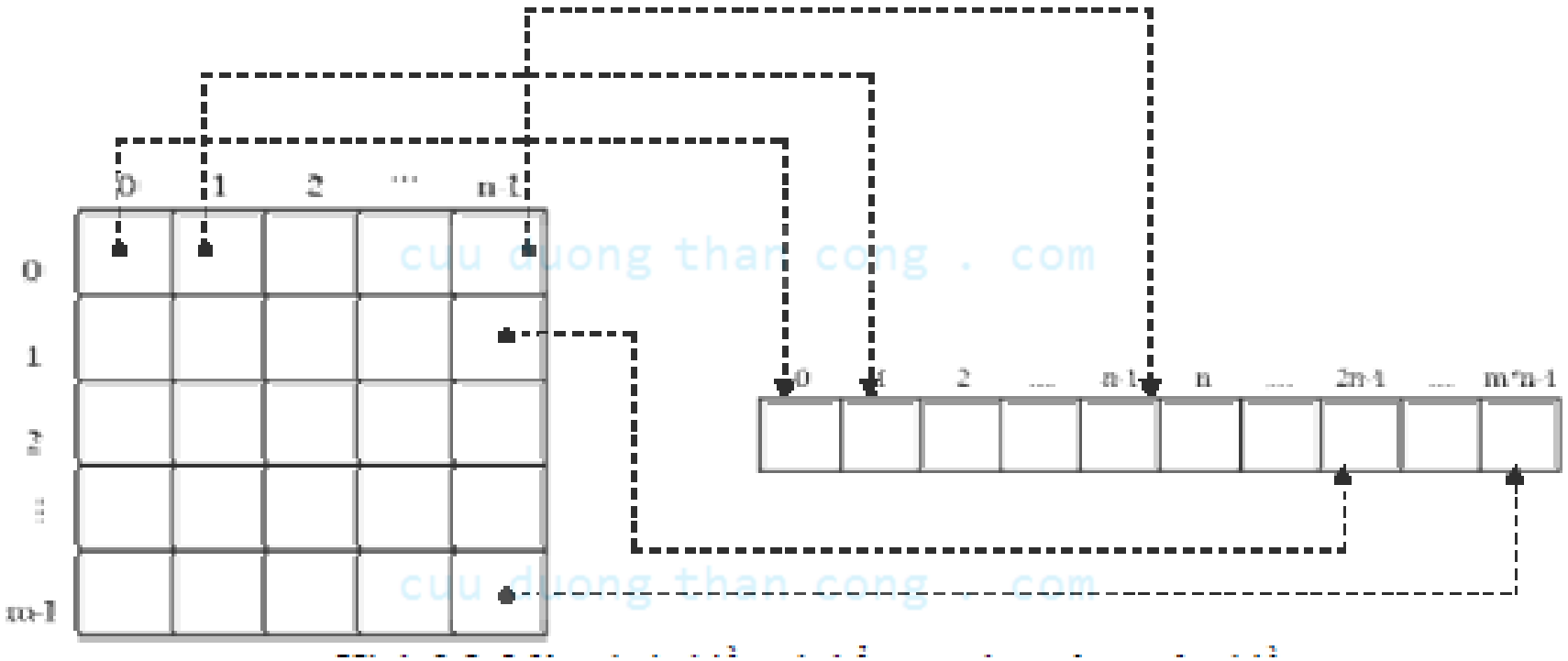
- **Ví dụ : in ra 1 ma tran vuông và cộng mỗi ptử của MT với 10**

```
#include <stdio.h>
#define hang 3
#define cot 3
int main() {
  int mt[hang][cot] = {{7,8,9},{10,13,15},{2,7,8}};
  int i,j;
  for (i=0; i<hang; i++) {
     for (j=0; j<cot; j++)   printf(" %d ",mt[i][j]);
     printf("\n");
   }
// cách dùng địa chỉ (con trỏ tương đương)
for (i=0; i<hang;i++) {
      for (j=0;j<cot;j++)  {

              printf(" %d ", *(*(mt+i)+j);    }
       printf("\n"); }
  }
```

Dùng bộ nhớ động cho mảng

Ta có thể coi một mảng 2 chiều là 1 mảng 1 chiều như hình sau :



Gọi X là mảng hai chiều có kích thước m dòng và n cột. A là mảng một chiều tương ứng ,thì X[i][j] = A[i*n + j] *nếu phân bố theo hàng.*

Dùng bộ nhớ động cho mảng (cách 1)

- Với mảng số nguyên 2 chiều có kích thước là R * C ta khai báo như sau :

```
int **mt;
mt = new int *[R];
int temp = new  int [R*C];
for (i=0; i< R; ++i) {
    mt[i] = temp;
    temp += C;          ? ? ?
 }  / Khai bao xong.
```

Su dung : mt[i][j] như bình thường. cuối cùng để giải phóng:

```
delete [] mt[0];   // xoá ? Tại sao?
delete [] mt;
```

p)

```
 void main() {
int *A;
int row, col, i, j;
clrscr();
printf("\n so hang");
scanf("%d",&row);
printf("\n so cot");
scanf("%d",&col);
// cap phat vung nho cho mang A
A = (int *)  malloc (row * col *sizeof(int));
// nhap mang 1
for (i=0;i<row;i++)
for (j=0;j<col;j++)
  { printf("M1[%d][%d]=",i,j);
    scanf("%d",A +i*row+j);
    }
```

p)

```
            ng
printf("\n mang 1\n");
for (i=0;i<row;i++)
  { for (j=0;j<col;j++)    printf (" %p  %d   ",A +i *row+j, *(A +i
    *row+j));
   printf("\n");
   }
   getch();

}
```

```
so hang3

so cot3
M1[0][0]=1
M1[0][1]=2
M1[0][2]=3
M1[1][0]=4
M1[1][1]=5
M1[1][2]=6
M1[2][0]=7
M1[2][1]=8
M1[2][2]=9

mang 1
077C  1     077E  2     0780  3
0782  4     0784  5     0786  6
0788  7     078A  8     078C  9
```

ch 2)

```
// chuong trinh thu cap phat dong cho mang 2 chieu row x col phan tu.
// CT cap phat lan dau theo hang: Spt bang so hang voi con tro *A.
// CT cap phat tiep theo cot voi con tro *(A+i)
void main() {
int **A;
int row, col, i, j;
clrscr();
printf("\n so hang");
scanf("%d",&row);
printf("\n so cot");
scanf("%d",&col);
// cap phat vung nho cho mang 1, mang 2
*A = (int *)  malloc (row  *sizeof(int));
for (i=0;i <row;i++)
   *( A+i) = (int *) malloc (col *sizeof(int));
```

p)

```
 // nhap mang 1
for (i=0;i<row;i++)
for (j=0;j<col;j++)
  { printf("M1[%d][%d]=",i,j);
    scanf("%d",*(A +i)+j);
    }
// In  mang
printf("\n mang 1\n");
for (i=0;i<row;i++)
  {for (j=0;j<col;j++)    printf (" %p  %d   ", *(A +i)+j, *(*(A +i) +j));
   printf("\n");
   }
```

CT cộng hai ma trận với mỗi ma trận được cấp phát động

```
#include <iostream.h>
#include <conio.h>
 int main()
 {
      int M,N;
      int *A = NULL,*B = NULL,*C = NULL;
      clrscr();
      cout<<"Nhap so dong cua ma tran:";      cin>>M;
      cout<<"Nhap so cot cua ma tran:";      cin>>N;
     //Cấp phát vùng nhớ cho ma trận A
      if (!AllocMatrix(&A,M,N))
      {
            cout<<"Khong con du bo nho!"<<endl;
            return 1;
      }
     //Cấp phát vùng nhớ cho ma trận B
      if (!AllocMatrix(&B,M,N))
      {
            cout<<"Khong con du bo nho!"<<endl;
            FreeMatrix(A);//Giải phóng vùng nhớ A
            return 1;
      }
```

```
    //Cấp phát vùng nhớ cho ma trận C
    if (!AllocMatrix(&C,M,N))
    {
        cout<<"Khong con du bo nho!"<<endl;
        FreeMatrix(A);//Giải phóng vùng nhớ A
        FreeMatrix(B);//Giải phóng vùng nhớ B
        return 1;
    }
   cout<<"Nhap ma tran thu 1"<<endl;
    InputMatrix(A,M,N,'A');
    cout<<"Nhap ma tran thu 2"<<endl;
    InputMatrix(B,M,N,'B');
    clrscr();
    cout<<"Ma tran thu 1"<<endl;
    DisplayMatrix(A,M,N);
    cout<<"Ma tran thu 2"<<endl;
    DisplayMatrix(B,M,N);
    AddMatrix(A,B,C,M,N);
    cout<<"Tong hai ma tran"<<endl;
    DisplayMatrix(C,M,N);
    FreeMatrix(A);//Giải phóng vùng nhớ A
    FreeMatrix(B);//Giải phóng vùng nhớ B
    FreeMatrix(C);//Giải phóng vùng nhớ C
    return 0;
}
```

```
//Cộng hai ma trận
void AddMatrix(int *A,int *B,int*C,int M,int N)
{
        for(int I=0;I<M*N;++I)
        C[I] = A[I] + B[I];
}
//Cấp phát vùng nhớ cho ma trận
int AllocMatrix(int **A,int M,int N)
{
        *A = new int [M*N];
        if (*A == NULL)
        return 0;
        return 1;
}
//Giải phóng vùng nhớ
void FreeMatrix(int *A)
{
        if (A!=NULL)
        delete [] A;
}
```

```
//Nhập các giá trị của ma trận
void InputMatrix(int *A,int M,int N,char Symbol)
{
        for(int I=0;I<M;++I)
        for(int J=0;J<N;++J)
        {
                cout<<Symbol<<"["<<I<<"]["<<J<<"]=";
                cin>>A[I*N+J];
        }
}
//Hiển thị ma trận
void DisplayMatrix(int *A,int M,int N)
{
        for(int I=0;I<M;++I)
        {
            for(int J=0;J<N;++J)
            {
                out.width(7);//canh le phai voi chieu dai 7 ky tu
                cout<<A[I*N+J];
            }
            cout<<endl;
        }
}
```

Phép tham chiếu

Trong C, hàm nhận tham số là con trỏ đòi hỏi chúng ta phải thận trọng khi gọi hàm. Chúng ta cần viết hàm hoán đổi giá trị giữa hai số như sau:

```
void Swap(int *X, int *Y);
{
    int Temp = *X;
    *X = *Y;
    *Y = *Temp;
}
```


Để hoán đổi giá trị hai biến *A* và *B* thì chúng ta gọi hàm:

```
Swap(&A, &B);
```

Rõ ràng cách viết này không được thuận tiện lắm.

Dùng tham chiếu với c++

```
void Swap(int &X, int &Y)
 {
   int Temp = X;
   X = Y;
   Y = Temp ;
}
```


- Chúng ta gọi hàm như sau :

  Swap(A, B);

- Với cách gọi hàm này, C++ tự gửi địa chỉ của *A* và *B* làm tham số cho hàm *Swap()*.

- **Khi một hàm trả về một tham chiếu, chúng ta có thể gọi hàm ở phía bên trái của một phép gán.**

```
#include <iostream.h>
int X = 4;
int & MyFunc()
{
    return X;
}

int main()
{
   cout<<"X="<<X<<endl;
   cout<<"X="<<MyFunc()<<endl;
   MyFunc() = 20; //Nghĩa là X = 20
   cout<<"X="<<X<<endl;
   return 0;
}
```

Phép đa năng hóa/chồng (Overloading)

- Với ngôn ngữ C++, chúng ta có thể *chồng* các hàm và các toán tử (operator). Chồng là phương pháp cung cấp nhiều hơn một định nghĩa cho tên hàm/toán tử đã cho trong cùng một phạm vi. Trình biên dịch sẽ lựa chọn phiên bản thích hợp của hàm hay toán tử dựa trên các tham số mà nó được gọi.
- Với C, tên hàm phải là duy nhất

Chồng hàm (Functions overloading)

- Trong c phải dùng 3 hàm để tính trị tuyệt đối ? :

  int abs(int i);

  long labs(long l);

  double fabs(double d);

- C++ cho phép chúng ta tạo ra các hàm khác nhau có cùng một tên.

  int abs(int i);

  long abs(long l);

  double abs(double d);

```
#include <iostream.h>
#include <math.h>
int MyAbs(int X)  {
    return abs(X);
}
long MyAbs(long X) {
    return labs(X);
}
double MyAbs(double X) {
    return fabs(X);
}
int main()  {
    int X = -7;
    long Y = 200000l;
    double Z = -35.678;
    cout<<"Tri tuyet doi cua so nguyen "<<X<<" la “
   <<MyAbs(X)<<endl;
    cout<<"Tri tuyet doi cua so nguyen "<<Y<<" la "
   <<MyAbs(Y)<<endl;
    cout<<"Tri tuyet doi cua so thuc "<<Z<<" la " <<MyAbs(Z)<<endl;
    return 0;
}
```

Chồng toán tử

- Trong ngôn ngữ C, khi chúng ta tự tạo ra một kiểu dữ liệu mới, chúng ta thực hiện các thao tác liên quan đến kiểu dữ liệu đó thường thông qua các hàm, điều này trở nên không thoải mái.
- Ví dụ : cài đặt các phép toán cộng và trừ số phức

```
#include <stdio.h> /* Dinh nghia so phuc */
struct  SP {
        double THUC;            double Image; }  ;
SP SetSP(double R,double I);
SP AddSP(SP C1,SP C2);
SP SubSP(SP C1,SP C2);
void DisplaySP(SP C);
int main(void) {
      SP C1,C2,C3,C4;
      C1 = SetSP(1.0,2.0);
        C2 = SetSP(-3.0,4.0);
      cout <<"\nSo phuc thu nhat:";          DisplaySP(C1);
      cout << "\nSo phuc thu hai:";          DisplaySP(C2);
      C3 = AddSP(C1,C2);
        C4 = SubSP(C1,C2);
      cout <<"\nTong hai so phuc nay:";     DisplaySP(C3);
      cout << "\nHieu hai so phuc nay:";      DisplaySP(C4);
    return 0;
}
```

```
SP SetSP(double R,double I) {
     SP Tmp;
     Tmp.THUC = R; Tmp.Image = I;
 return Tmp; }
SP AddSP(SP C1,SP C2) {
    SP Tmp;
    Tmp.THUC = C1.THUC+C2.THUC;
    Tmp.Image = C1.Image+C2.Image;
return Tmp; }
SP SubSP(SP C1,SP C2) {
       SP Tmp;
       Tmp.THUC = C1.THUC-C2.THUC;
       Tmp.Image = C1.Image-C2.Image;
return Tmp; }
void DisplaySP(SP C) { cout <<C.THUC <<‘ i ’ <<C.Image;
    }
```

C++

- Trong ví dụ trên, ta dùng hàm để cài đặt các phép toán cộng và trừ hai số phức ; => phức tạp,không thoải mái khi sử dụng, vì thực chất thao tác cộng và trừ là các toán tử chứ không phải là hàm.
- C++ cho phép chúng ta có thể định nghĩa lại chức năng của các toán tử đã có sẵn một cách tiện lợi và tự nhiên hơn rất nhiều. Điều này gọi là chồng toán tử.
- Một hàm định nghĩa một toán tử có cú pháp sau:

data_type ***operator*** *operator_symbol* ***(*** *parameters* ***)***

{ *.....*

}

Trong đó:

- *data_type: Kiểu trả về.*
- *operator_symbol: Ký hiệu của toán tử.*
- *parameters: Các tham số (nếu có).*

```
#include <iostream.h> //Dinh nghia so phuc
struct { double THUC; double AO; }SP;
SP SetSP(double R,double I);
void DisplaySP(SP C);
SP operator + (SP C1,SP C2);
SP operator - (SP C1,SP C2);
 int main() {
     SP C1,C2,C3,C4;          C1 = SetSP(1.1,2.0);
     C2 = SetSP(-3.0,4.0);    cout<<"\nSo phuc thu nhat:";
    DisplaySP(C1);            cout<<"\nSo phuc thu hai:";
    DisplaySP(C2);            C3 = C1 + C2;
    C4 = C1 - C2;             cout<<"\nTong hai so phuc nay:";
    DisplaySP(C3);    cout<<"\nHieu hai so phuc nay:";
    DisplaySP(C4);
return 0; }
```

```
SetSP(double R,double I) {
 SP Tmp;
Tmp.THUC = R; Tmp.AO = I; return Tmp; }
   //Cong hai so phuc
SP operator + (SP C1,SP C2) { SP Tmp;
Tmp.THUC = C1.THUC+C2.THUC;
Tmp.AO = C1.AO+C2.AO; return Tmp; }
//Tru hai so phuc
 SP operator - (SP C1,SP C2) { SP Tmp;
Tmp.THUC = C1.THUC-C2.THUC;
Tmp.AO = C1.AO-C2.AO; return Tmp; }
 //Hien thi so phuc
void DisplaySP(SP C) {
   cout<<"("<<C.THUC<<","<<C.AO<<")"; }
```

Các giới hạn của chồng toán tử

- Không thể định nghĩa các toán tử mới.
- Hầu hết các toán tử của C++ đều có thể được chồng. Các toán tử sau không được chồng là :

  :: Toán tử định phạm vi.

  .* Truy cập đến con trỏ là trường của **struct** hay **class**.

  . Truy cập đến trường của **struct** hay **class**.

  ?: Toán tử điều kiện

  sizeof

  Các ký hiệu tiền xử lý .
- Không thể thay đổi thứ tự ưu tiên của một toán tử cũng như số các toán hạng của nó.
- Không thể thay đổi ý nghĩa của các toán tử khi áp dụng cho các kiểu có sẵn.
- Chồng các toán tử không thể có các tham số có giá trị mặc định.