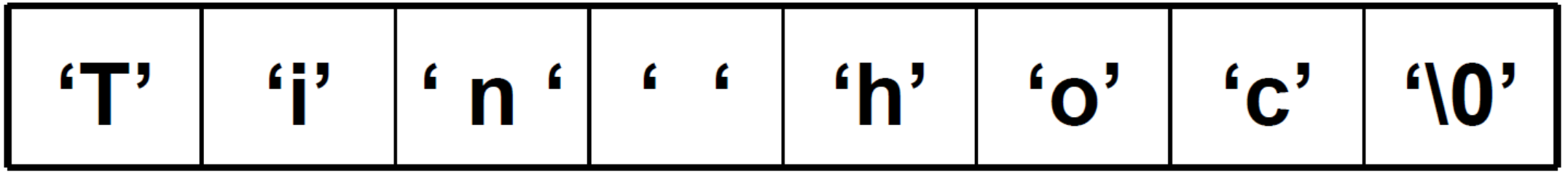
CHƯƠNG 5
CHUỖI KÝ TỰ

GIẢNG VIÊN

TS. Hà Ngọc Long

Khái niệm chuỗi ký tự

- **Xâu kí tự (string)** là một dãy các kí tự viết liên tiếp nhau
  - Độ dài xâu là số kí tự có trong xâu
  - Xâu không có kí tự nào: **Xâu rỗng**
- Ví dụ: **"Tin hoc", "String"**
- Lưu trữ: kết thúc xâu bằng kí tự **'\0'** hay **NULL** (mã ASCII là 0)



Lưu ý

- Xâu kí tự $\times$ mảng kí tự
  - Tập hợp các kí tự viết liên tiếp nhau
    - Truy nhập một phần tử của xâu ký tự (là một ký tự) giống như truy nhập vào một phần tử của mảng: **`Tên[Chỉ_số]`**
  - Xâu kí tự có kí tự kết thúc xâu, mảng kí tự không có kí tự kết thúc xâu
- Xâu kí tự độ dài 1 $\times$ kí tự (`"A" ='A' ?`)
  - `'A'` là 1 kí tự, được lưu trữ trong 1 byte
  - `"A"` là 1 xâu kí tự, ngoài kí tự **`'A'`** còn có kí tự **`'\0'`**

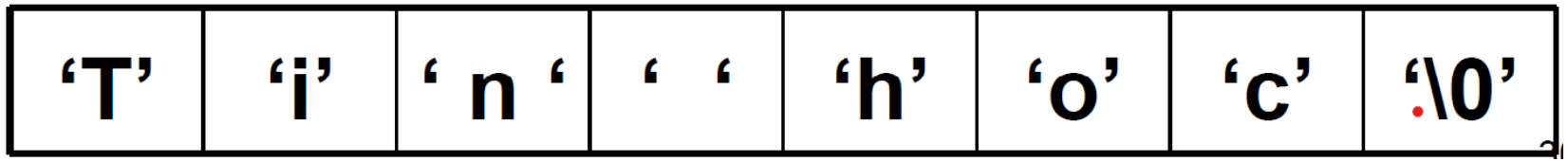
    => được lưu trữ trong 2 byte

Khai báo

```
char tên_xâu [số_kí_tự_tối_đa];
```

- Để lưu trữ một xâu có **n** kí tự chúng ta cần một mảng có kích thước **n+1**
  - Phần tử cuối cùng chứa ký tự NULL
- ***Ví dụ***
- • Để lưu trữ xâu “Tin hoc” chúng ta phải khai báo xâu có số phần tử tối đa ít nhất là 8

```
char str[8] = “Tin hoc”;
```



Truy cập phần tử của xâu

- Giống như truy nhập tới một phần tử của mảng ký tự.

tên_xâu [chỉ_số_của_kí_tự]

- ***Ví dụ:*** char Str[10] = "Tin hoc"

T	i	n	-	h	o	c	\0	?	\0

```
Str[0] → 'T'        Str[3] = '-';
Str[3] → ' '        Str[7] = ' ';
Str[7] → '\0 '      Str[8] = '1 ' ;
Str[8] → ?          Str[9] = '\0';
```

```
Str: Tin-hoc 1
```

Ví dụ: Nhập xâu và đếm số ký tự '*'

```
#include <stdio.h>
void main(){
    char Str[100];
    int d=0, i=0;
    printf("Nhap xau ky tu: "); gets(Str);
    while(Str[i] != '\0'){
        if(Str[i]=='*')
            d++;
        i++;
    }
    printf("Ket qua : %d",d);
}
```

```
Tính chiều dài của xâu
d=0;
while(Str[d] != '\0') d++;
```

```
Nhap xau ky tu: &&*****&&&&&**&*&
Ket qua : 8
```

TỆP TIÊU ĐỀ ctype.h

```
#include<ctype.h>
```

Các hàm xử lý ký tự: Chuyển đổi chữ hoa/thường

toupper(char ch)

- Chuyển kí tự thường thành kí tự hoa `toupper('a') => 'A'`

int tolower(char ch)

- Chuyển kí tự hoa thành kí tự thường `tolower('B') => 'b'`

Ví dụ

```
do{
        .....
        printf("Tiep tuc <C/K>? :"); fflush(stdin);
}while(toupper(getche()) !='K');
```

Các hàm xử lý ký tự: Kiểm tra chữ hoa/thường

`int islower(char ch)`

- Kiểm tra chữ thường:
  - Hàm trả về giá trị khác 0 nếu ch là chữ thường, ngược lại trả về 0
  - **Ví dụ:** `printf("%d ",islower('A'));` ⇒ `0`

`int isupper(char ch)`

- Kiểm tra chữ hoa:
  - Hàm trả về giá trị khác 0 nếu ch là chữ hoa, ngược lại trả về 0
  - Ví dụ: `printf("%d ",isupper('A'));` ⇒ ≠ `0 (1 !?)`

Các hàm xử lý ký tự: Kiểm tra chữ cái/chữ số

`int isalpha(char ch)`

– Kiểm tra kí tự trong tham số có phải chữ cái không (‘a’…’z’,’A’,..’Z’). Hàm trả về khác 0 nếu đúng, ngược lại trả về giá trị bằng 0

– Ví dụ: `printf("%d ",isalpha('A'));` ⇒ ≠ 0 (1 !?)

`int isdigit(char ch)`

– Kiểm tra kí tự trong tham số có phải chữ số (‘0‘,‘1‘,..‘9‘) không. Hàm trả về khác 0 nếu đúng, ngược lại trả về giá trị bằng 0

– Ví dụ: `printf("%d ",isdigit('A'));` ⇒ 0

Khái niệm chuỗi ký tự: Kiểm tra ký tự đặc biệt

`int iscntrl(char ch)`

– Kiểm tra kí tự điều khiển (0-31).

– Hàm trả về khác 0 nếu đúng, ngược lại trả về giá trị bằng 0

`int isspace(char ch)`

– Kiểm tra kí tự dấu cách (mã 32), xuống dòng **`('\n' 10)`**, đầu dòng **`('\r' 13)`**, tab ngang **`('\t' 9)`**, tab dọc **`('\v' 11)`**.

– Hàm trả về khác 0 nếu đúng, ngược lại trả về giá trị bằng 0

Ví dụ: Nhập chuỗi và đếm từ, phân cách bởi khoảng trắng

```c
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
int main(){
    char Str[100]; int d=0, i=0;
    printf("Nhap xau ky tu: "); gets(Str);
    if(Str[0] == '\0') printf(" Xau rong ");
    else{
        if( ! isspace(Str[0]) ) d=1;
        i=1;
        while(Str[i] != '\0'){
            if( isspace(Str[i-1] ) && (! isspace(Str[i])) ) d++;
            i++;
        }
        printf("Ket qua : %d",d);
    }
}
```

```
Nhap xau ky tu: NHAP MON LAP TRINH C
Ket qua : 5
```

Các hàm xử lý xâu ký tự

Vào/ra xâu (chuỗi) ký tự

- Tệp tiêu đề: `stdio.h`
- Nhập xâu kí tự
  - `gets(tên_xâu);`
  - `scanf("%s",&tên_xâu);`
- • Hiển thị xâu kí tự
  - `puts(tên_xâu);`
  - `printf("%s",tên_xâu);`

Sự khác nhau giữa `gets` và `scanf`?

TỆP TIÊU ĐỀ string.h

```
#include<string.h>
```

Các hàm xử lý chuỗi ký tự (1/6)

- ***Chú ý:***

```
char str[100] = “Hello world”;
char * p = str;
```

- `p` là con trỏ tới mảng các ký tự/ xâu ký tự
  - `p+6` cũng là xâu ký tự : world
- Xâu ký tự, có thể được khai báo `char *`

Các hàm xử lý chuỗi ký tự (2/6)

`size_t strlen(char * xâu)`

– Trả về độ dài xâu

```
printf("%d ",strlen("Hello world")); ⇒ 11
```

`char * strcpy(char * đích, char * nguồn)`

– Sao chép xâu, trả về giá trị xâu nguồn

```
char Str[20];
printf("%s ",strcpy(Str,"Hello")); ⇒ Hello
printf("%s", Str); ⇒ Hello
```

– **Chú ý:** Phép gán `Str = "Hello"` là không hợp lệ

Các hàm xử lý chuỗi ký tự (3/6)

`int strcmp(char * xâu_1, char * xâu_2)`

- So sánh hai xâu.
- Trả về giá trị 0 nếu hai xâu giống nhau;
- Giá trị < 0: xâu_1 < xâu_2
- Giá trị >0: xâu_1 > xâu_2

- ***Ví dụ***

```
char Str[20];
strcpy(Str,"hello");
printf("%d", strcmp(Str,"hello"));→0
printf("%d", strcmp(Str,"hello!"); →-1 (!?)
printf("%d", strcmp(Str,"Hello"); → 1 (!?)
```

Các hàm xử lý chuỗi ký tự (4/6)

char * strcat(char * xđích, char * nguồn)

– Ghép nối xâu nguồn vào ngay sau xâu đích, trả lại xâu kết quả

- Ví dụ

```
char Str[20];
strcpy(Str,"Hello ");
printf("%s ",strcat(Str,"world")); ⇒ Hello world
printf("\n%s",Str); ⇒ Hello world
```

Các hàm xử lý chuỗi ký tự (5/6)

char * strchr (char * s, int c)

– Trả về con trỏ trỏ tới vị trí xuất hiện đầu tiên của ký tự **c** trong **s**. Nếu không có trả về con trỏ null

```
strcpy(Str,"Hello world");
printf("%s ",strchr(Str,'o')); ⇒ o world
```

char* strstr(char * s1, char * s2)

– Trả về con trỏ trỏ tới vị trí xuất hiện đầu tiên của chuỗi **s2** trong **s1**. Nếu không tồn tại, trả về con trỏ null

```
printf("%s ",strstr(Str,"llo")); ⇒ llo world
```

Các hàm xử lý chuỗi ký tự (6/6)

Tệp tiêu đề: `stdlib.h`

`int atoi(char * str)`

– Chuyển một xâu kí tự thành một số nguyên tương ứng

– Ví dụ: `atoi("1234")` →`1234`

`int atol(char * str)`

– Chuyển xâu kí tự thành số `long int`

`float atof(char * str)`:

– Chuyển xâu kí tự thành số thực

– Ví dụ: `atof("123.456E-2")` →`1.23456`

Thất bại cả 3 hàm: trả về 0

Ví dụ 1: Đảo ngược chuỗi ký tự

```c
#include<stdio.h>
#include<string.h>
main(){
    char s[100],c;
    int i, n;
    printf("Nhap xau: "); gets(s);
    n =strlen(s);
    for(i=0;i <n/2;i++){
        c = s[i];
        s[i] = s[n-i-1];
        s[n-i-1]=c;
    }
    printf("%s",s);
}
```

```
Nhap xau: CONG NGHE THONG TIN
NIT GNOHT EHGN GNOC
```

Ví dụ 2: Kiểm tra chuỗi đối xứng

```c
#include<stdio.h>
#include<string.h>
main(){
    char s[20];
    int i,n;
    printf("Nhap vao xau ki tu: "); gets(s);
    n=strlen(s);
    for(i=0;i<n/2;i++)
        if(s[i]!=s[n-1-i])
            break;
    if(i==n/2)
        printf("Xau doi xung");
    else
        printf("Xau khong doi xung");
}
```

```
Nhap vao xau ki tu: ABCCBA
Xau doi xung
```

```
Nhap vao xau ki tu: ABCDDCAA
Xau khong doi xung
```

Ví dụ 3: Đếm số lần xuất hiện chữ cái trong chuỗi

```c
#include<stdio.h>
#include<ctype.h>
#include<string.h>
main(){
    char s[20];
    int dem[26] = {};
    int i,n;
    puts("Nhap vao xau ki tu:");gets(s);
    n=strlen(s);
    for(i=0;i<n;i++)
        if(isalpha(s[i]))
            dem[ tolower(s[i ]) - 'a' ]++;
    for(i=0;i<26;i++)
        if(dem[i]!=0)
            printf("Ki tu %c xuat hien %d lan\n",'a'+i,dem[i]);
}
```

```
Nhap vao xau ki tu:
NHAP MON LAP TRINH
Ki tu a xuat hien 2 lan
Ki tu h xuat hien 2 lan
Ki tu i xuat hien 1 lan
Ki tu l xuat hien 1 lan
Ki tu m xuat hien 1 lan
Ki tu n xuat hien 3 lan
Ki tu o xuat hien 1 lan
Ki tu p xuat hien 2 lan
Ki tu r xuat hien 1 lan
Ki tu t xuat hien 1 lan
```

Mảng chuỗi ký tự

- Xâu (Chuỗi) ký tự **có thể là kiểu phần tử của mảng**
- **Khai báo**

```
char DS[100][30];
```

  Mảng có tối đa 100 phần tử, các phần tử là xâu có độ dài tối đa 30

- **Sử dụng**
  - Như một mảng bình thường
  - Mỗi phần tử mảng được sử dụng như một xâu ký tự

Ví dụ 1: Nhập vào DSSV tới khi tên rỗng & in DS

```c
#include <stdio.h>
void main(){
    int i, n;
    char DS[100][30];
    printf("Nhap DSSV (<100), go Enter de thoat..\n");
    n =0;
    do{
        printf("Ten sinh vien[%d]: ",n+1); gets(DS[n]);
        if(DS[n][0] !='\x0') n++;
        else break;
        if(n==100) break;
    }while(1);
    printf("\n\nDS sinh vien vua nhap \n");
    for(i=0;i<n;i++) printf("%s\n",DS[i]);
}
```

```
Nhap DSSV (<100), go Enter de thoat..
Ten sinh vien[1]: NGUYEN VAN A
Ten sinh vien[2]: TRAN THI B
Ten sinh vien[3]: LE VAN C
Ten sinh vien[4]: NGUYEN THI D
Ten sinh vien[5]:


DS sinh vien vua nhap
NGUYEN VAN A
TRAN THI B
LE VAN C
NGUYEN THI D
```

Ví dụ 2: Nhập vào DSSV & in DS sau sắp xếp

```c
#include <stdio.h>
#include <string.h>
void main(){
    int i, j, N;
    char DS[100][30], str[30];
    //Nhap DS lop
    printf("So sinh vien : ");
    scanf("%d",&N);
    fflush(stdin);
    for(i=0;i < N;i++){
        printf("Ten sinh vien[%d]: ",i);
        gets(DS[i]);
    }
```

```
So sinh vien : 4
Ten sinh vien[0]: NGUYEN VAN C
Ten sinh vien[1]: LE VAN A
Ten sinh vien[2]: TRAN THI D
Ten sinh vien[3]: LE THI F

DS sinh vien vua nhap
LE THI F
LE VAN A
NGUYEN VAN C
TRAN THI D
```

```c
//So sánh theo Họ+đệm+tên
    for(i = 0; i < N - 1; i ++)
        for(j = i +1; j < N; j ++)
            if(strcmp(DS[i],DS[j]) > 0){
                strcpy(str,DS[i]);
                strcpy(DS[i],DS[j]);
                strcpy(DS[j],str);
            }
    //In danh sach da sap xep
    printf("\nDS sinh vien vua nhap \n");
    for(i=0;i < N;i++)
        printf("%s\n",DS[i]);
}//main
```

Ví dụ 2: Sắp xếp theo tên

```
/Sap xep theo tên
    char ten_i[30],ten_j[30];
    for(i = 0; i < N - 1; i ++)
        for(j = i +1; j < N; j ++){
            strcpy(ten_i,strrchr(DS[i],32));
            strcpy(ten_j,strrchr(DS[j],32));
            if(strcmp(ten_i,ten_j) > 0){
                strcpy(str,DS[i]);
                strcpy(DS[i],DS[j]);
                strcpy(DS[j],str);
            }
        }
```

```
So sinh vien : 4
Ten sinh vien[0]: NGUYEN VAN C
Ten sinh vien[1]: LE VAN A
Ten sinh vien[2]: TRAN THI D
Ten sinh vien[3]: LE THI F

DS sinh vien vua nhap
LE VAN A
NGUYEN VAN C
TRAN THI D
LE THI F
```

CHƯƠNG 6
QUẢN LÝ BỘ NHỚ & CON TRỎ

GIẢNG VIÊN

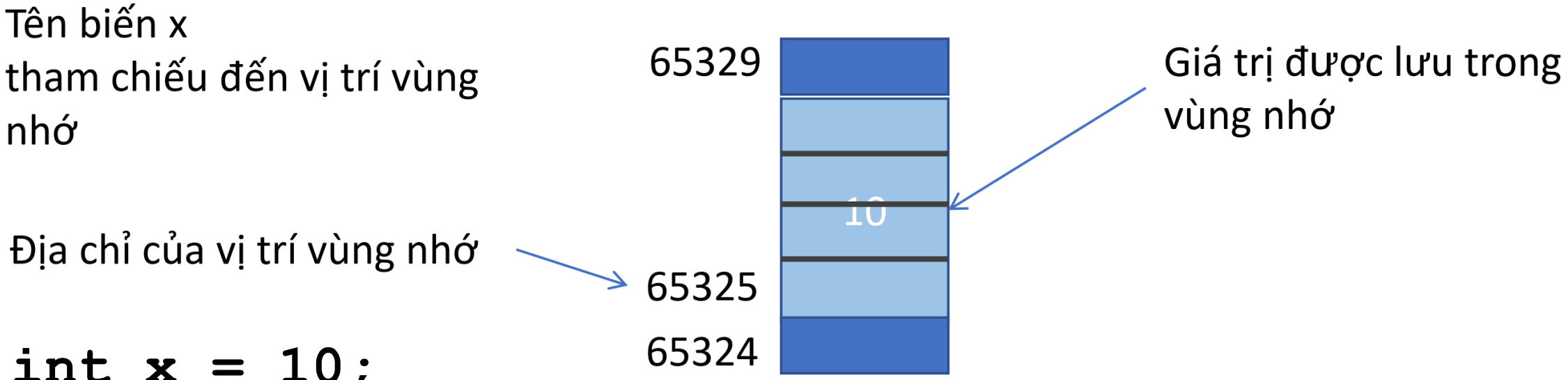
TS. Hà Ngọc Long

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
- (e.g. an assigned name for a memory location)
- **&x** biểu diễn địa chỉ ô nhớ đầu tiền thuộc vùng nhớ của **x**, tức là 65325
- ***(&x)** or **x** biểu diễn giá trị được lưu trong vùng nhớ của **x**, tức là 10

Ví dụ

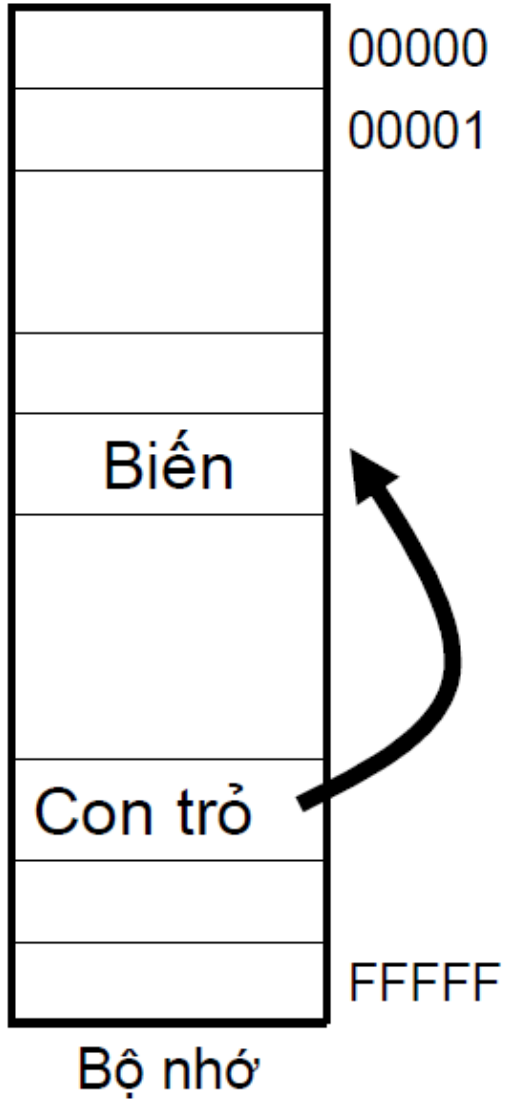
```c
#include<stdio.h>
int main() {
    int x = 10;
    printf("Value of x is %d\n",x);
    printf("Address of x in Hex is %p\n",&x);
    printf("Address of x in decimal is %lu\n",&x);
    printf("Value at address of x is %d\n",*(&x));
    return 0;
}
```

```
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
```

Con trỏ

Giới thiệu

- Là một khái niệm "mạnh" trong C
  - Cho phép tính toán trên con trỏ
  - Sử dụng con trỏ hàm
- Cho phép truy nhập gián tiếp tới một đối tượng có địa chỉ (*biến*, *hàm*)
  - Truy nhập trực tiếp→ thông qua tên

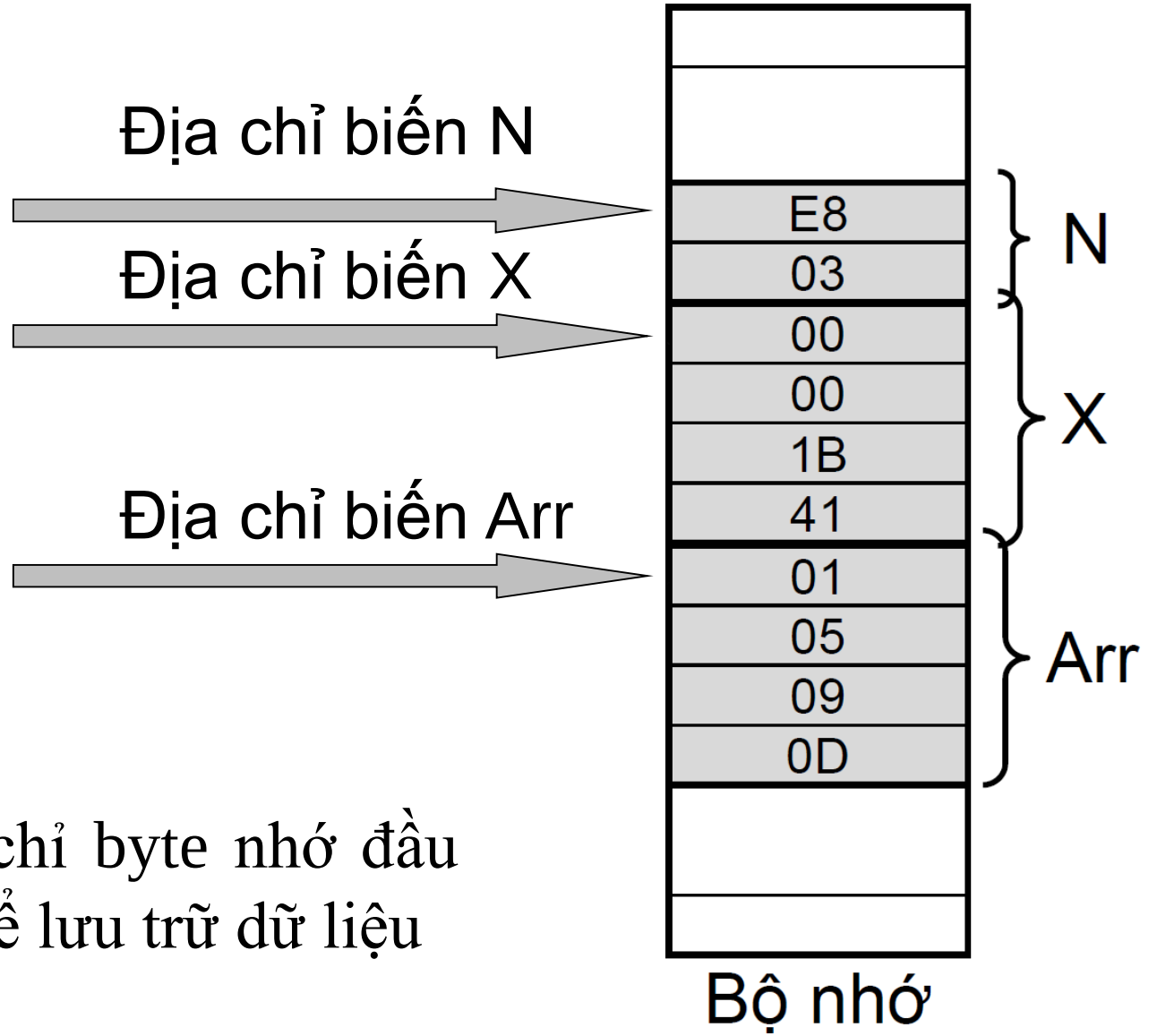


Địa chỉ

- Bộ nhớ gồm dãy các ô nhớ
  - Mỗi ô nhớ là một byte
  - Mỗi ô nhớ có một địa chỉ riêng
- Các biến trong chương trình được lưu tại vùng nhớ nào đó trong bộ nhớ
- Khi khai báo biến, tùy thuộc vào kiểu, biến sẽ được cấp một số ô nhớ liên tục nhau
  - Biến **`int`** được cấp 2 **`bytes`**, float được cấp 4 **`bytes`**,..
  - Địa chỉ của biến, là địa chỉ của byte đầu tiên trong số các byte được cấp
  - Khi gán giá trị cho biến, nội dung các byte cung cấp cho biến sẽ thay đổi

Ví dụ

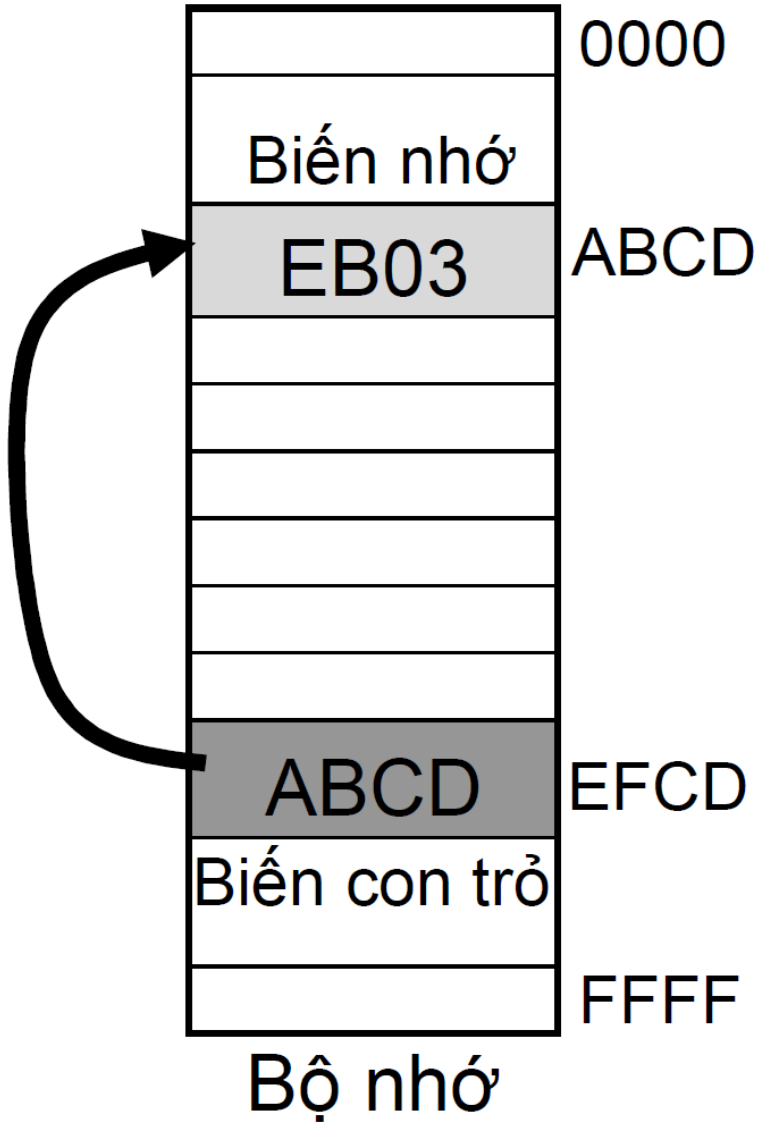
```
int N;
float x;
char Arr[4];
N = 1000;//03E8
X=9.6875;//411B0000
for(i=0;i<4;i++)
      Arr[i]= 4*i+1;
```



Địa chỉ của một biến là địa chỉ byte nhớ đầu tiên được cung cấp cho biến để lưu trữ dữ liệu

Con trỏ

- Con trỏ là một biến mà giá trị của nó là địa chỉ của một vùng nhớ
  - Vùng nhớ này có thể dung để chứa các biến có kiểu cơ bản (nguyên, thực, ký tự,…) hay có cấu trúc (mảng, bản ghi,..)
- • Con trỏ dùng "trỏ tới" một biến nhớ
  - Có thể trỏ tới một hàm
  - Có thể trỏ tới con trỏ khác



Khai báo con trỏ

```
Kiểu * Tên;
```

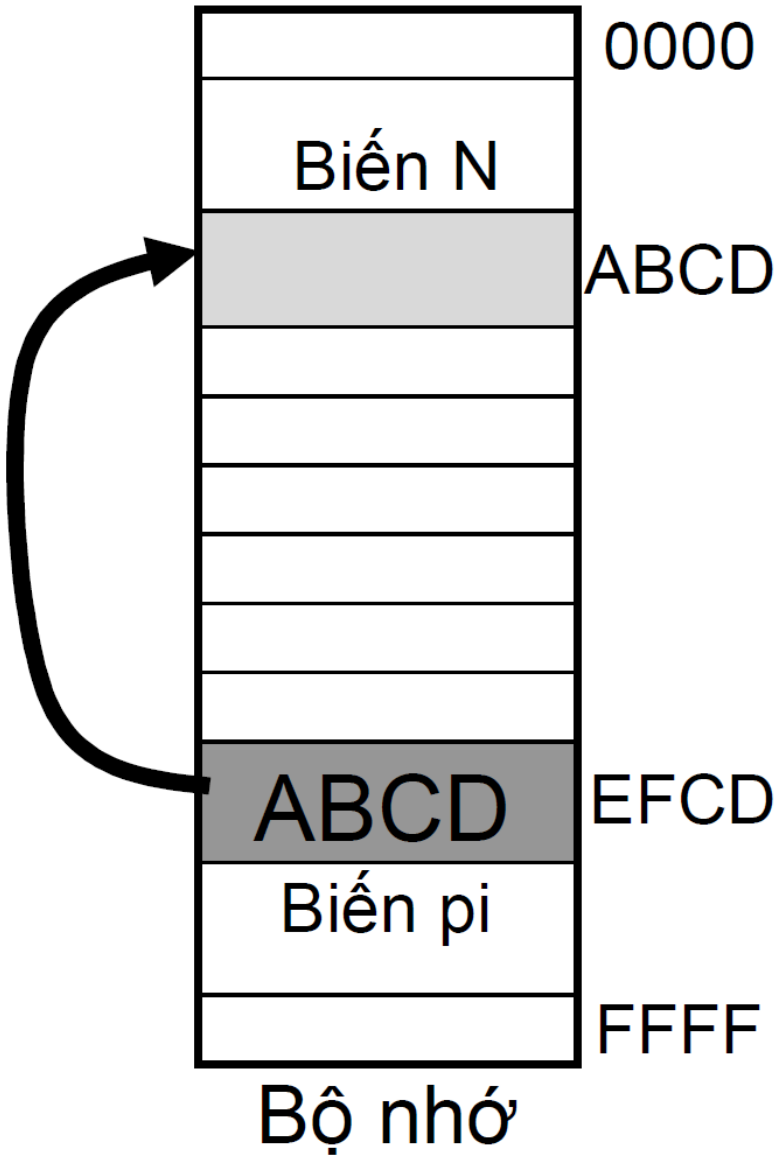
- **Tên**: Tên của một biến con trỏ.
- **Kiểu**: Kiểu của biến mà con trỏ "Tên" trỏ tới.
  - Giá trị của con trỏ có thể thay đổi được
    - Trỏ tới các biến khác nhau, có cùng kiểu
  - Kiểu biến mà con trỏ trỏ tới không thay đổi được
  - Muốn thay đổi phải thực hiện "ép kiểu"
- ***Ví dụ:***

```
int * pi; //Con trỏ, trỏ tới một biến kiểu nguyên
char * pc; //Con trỏ, trỏ tới một biến kiểu ký tự
```

Toán tử địa chỉ (&)

- **Ký hiệu: &**
- Là toán tử một ngôi, trả về địa chỉ của biến
  - Địa chỉ biến có thể được gán cho một con trỏ, trỏ tới đối tượng cùng kiểu
- Ví dụ

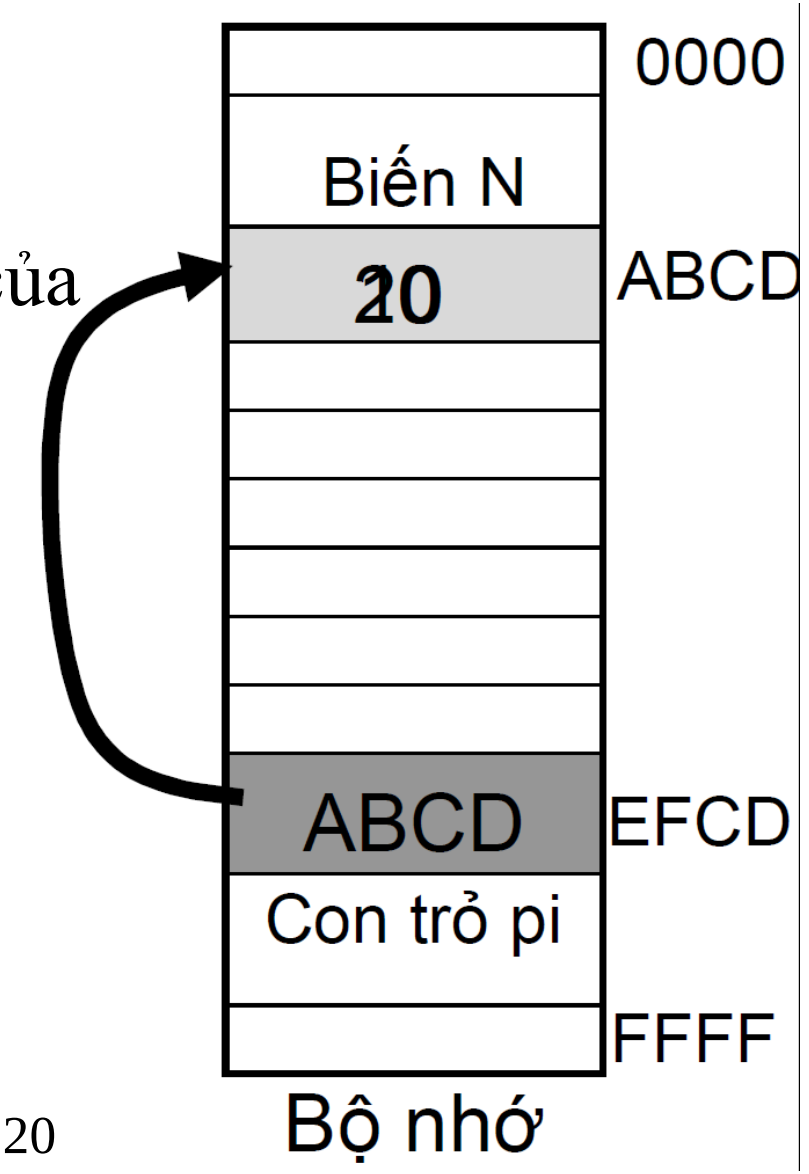
```
int N; // &N→ ABCD
int * pi;
pi = &N; // pi←ABCD
```



Toán tử nội dung (*)

- Ký hiệu: *****
- Là toán tử một ngôi, trả về giá trị (nội dung) của vùng nhớ mà con trỏ đang trỏ tới
- Ví dụ

```
int N;
int * pi;
pi = &N;
N = 10; //Vùng nhớ mà pi trỏ tới mang giá trị 10; Vậy *pi=10
*pi = 20; // Vùng nhớ pi trỏ tới được gán giá trị 20; Vậy N= 20
```



Gán giá trị cho con trỏ

- Con trỏ được gán địa chỉ của một biến
  - Biến cùng kiểu với kiểu mà con trỏ trỏ tới
    - Nếu không, cần phải ép kiểu
- Con trỏ được gán giá trị của con trỏ khác
  - Hai con trỏ sẽ trỏ tới cùng một biến(do cùng địa chỉ)
  - Hai con trỏ nên cùng kiểu trỏ đến
    - Nếu không, phải ép kiểu

- Con trỏ được gán giá trị NULL

Ví dụ:

```
int *p;
p = 0;
```

- Gán nội dung vùng nhớ 2 con trỏ trỏ tới.

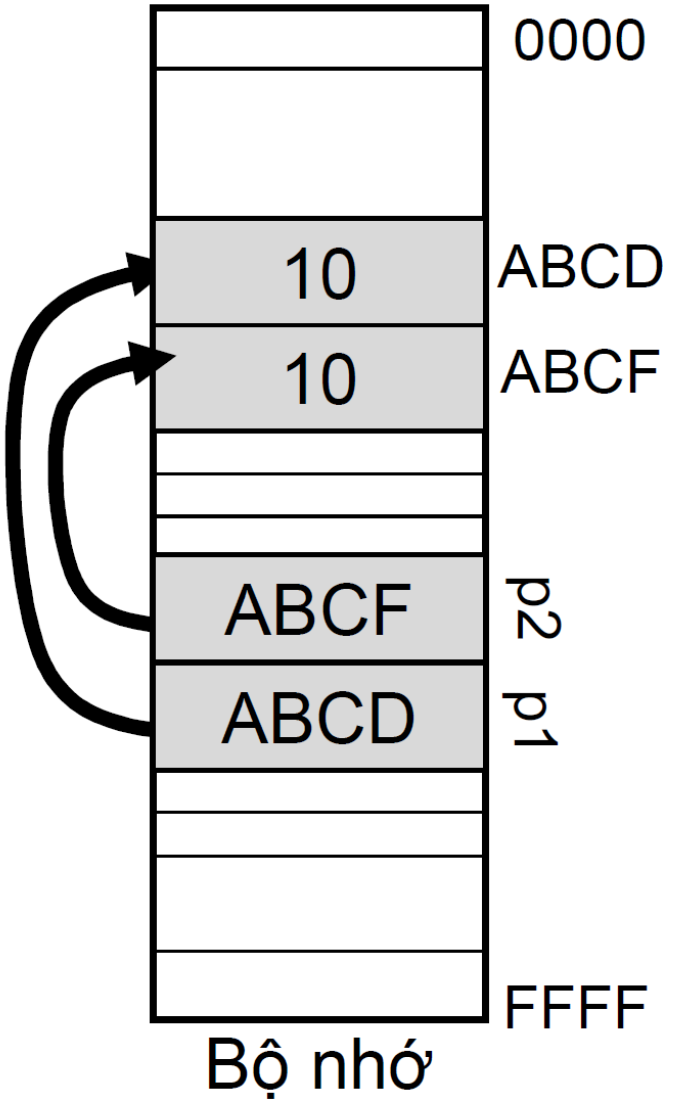
Ví dụ:

```
int *p1, *p2;
*p1 = *p2;
```

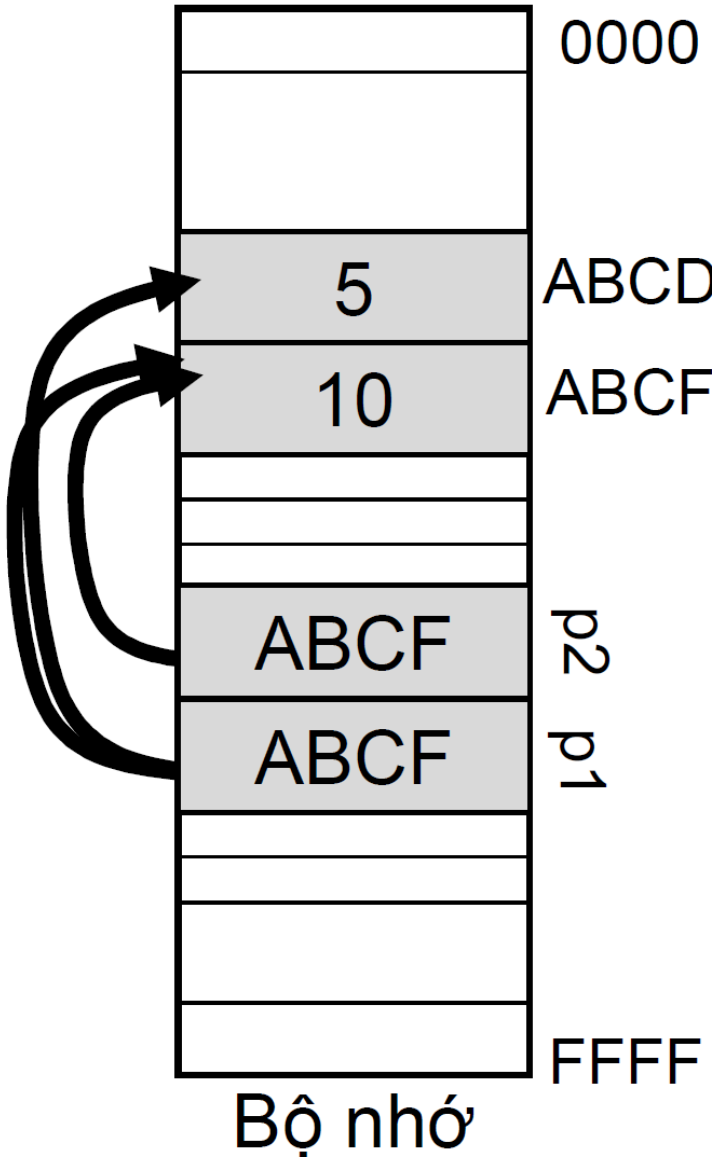
Ví dụ 1

```c
#include <stdio.h>
void main(){
    int N=5, M=10;
    int *p1 = &N;
    int *p2 = &M;
    *p1 = *p2;
    printf("%d %d",*p1,*p2);
}
```

```
10 10
```



Ví dụ 2




```c
#include <stdio.h>
void main(){
    int N=5, M=10;
    int *p1 = &N;
    int *p2 = &M;
    p1 = p2;
    printf("%d %d",*p1,*p2);
}
```

```
10 10
```

Các phép toán trên con trỏ

- **Cộng con trỏ** với một số nguyên
  - Kết quả: Con trỏ cùng kiểu
- **Trừ con trỏ** với một số nguyên
  - Kết quả: Con trỏ cùng kiểu
- **Trừ 2 con trỏ** cùng kiểu cho nhau
  - Kết quả: Một số nguyên
    - Khoảng cách giữa 2 con trỏ được đo bằng số phần tử thuộc kiểu dữ liệu mà con trỏ trỏ tới

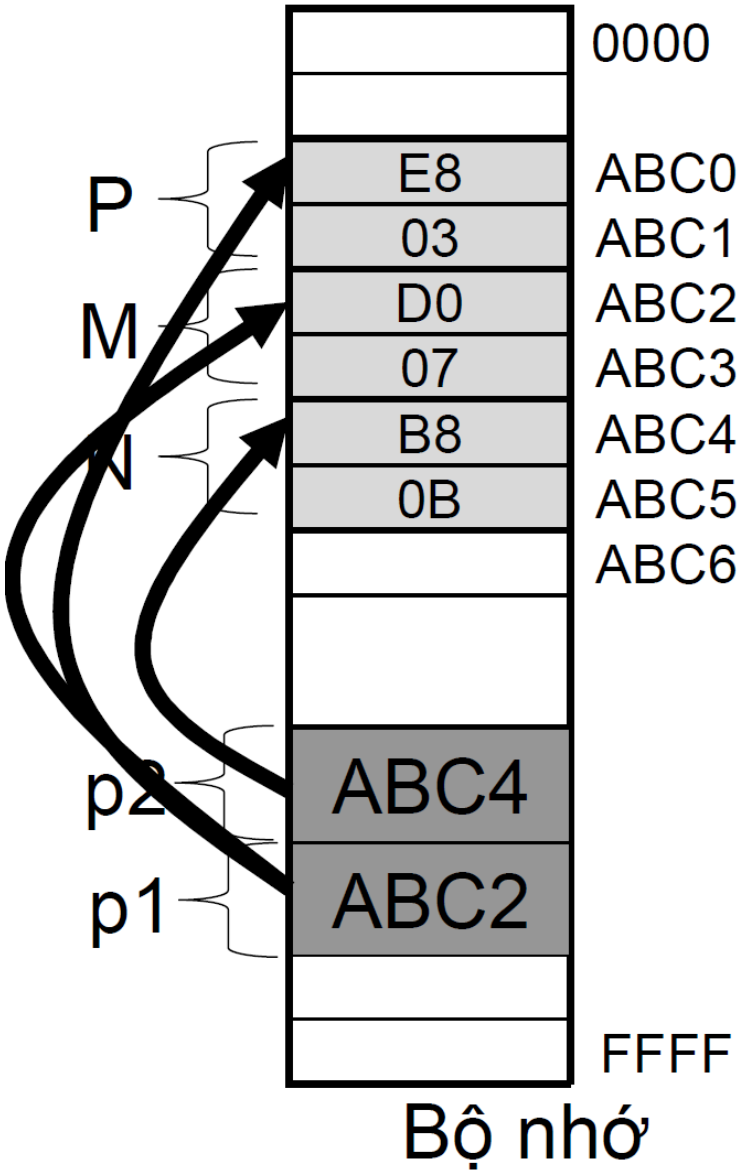
Ví dụ: Phép toán trên con trỏ

- int N=1000, M=2000,P=3000;
- int * p1 = &P, *p2 = &N;

```
p1 - p2 → -2
```

```
* (p2-1) → 2000
```

```
* ++ p1 → 2000
```



Ghi chú:

- Kiểu **int**, các phần tử cách nhau 2 bytes
- Kiểu **float**, các phần tử cách nhau 4 bytes

Mối quan hệ giữa con trỏ và mảng một chiều

- Nếu **`Tab`** là tên một mảng ⇒ **`Tab`** là một con trỏ hằng chứa địa chỉ của phần tử đầu tiên của mảng **`Tab (&Tab[0])`**
  - Không tồn tại phép tính trên tên mảng, hoặc gán giá trị cho tên mảng (VD: **`Tab=…; Tab++`**)
- Có thể sử dụng một con trỏ để duyệt mảng nếu nó được gán giá trị bằng địa chỉ của mảng (địa chỉ của phần tử đầu tiên)

Ví dụ

```
int Tab[10];

int * p = Tab; // int *p = &Tab[0]
```

```
for(i = 0; i < 10; i ++)
    printf("%d ", *(p + i) );
```

```
for(i = 0; i < 10; i ++)
    printf("%d ", p[i]);
```

```
for(i = 0; i < 10; i ++)
    printf("%d ", *(p++) );
```

Con trỏ void

```
void * Tên_con_trỏ
```

- **Con trỏ đặc biệt:** con trỏ không có kiểu, có thể trỏ tới bất kỳ biến nào
- Có thể nhận giá trị là địa chỉ của một biến có kiểu dữ liệu bất kỳ
  - Thường dùng làm đối số trong lời gọi hàm để có thể nhận bất kỳ kiểu địa chỉ nào của tham số được truyền
- ***Ví dụ:***

```
void * p, *q;
int n; float x;
p = & n; q= & x; \\ ← Các câu lệnh hợp lệ
```

Ví dụ 1

```c
#include<stdio.h>
int main() {
    int a=3;
    void *p;
    p = &a;
    printf("%d\n", a * *p * a + *p);
    return 0;
}
```

30

Ví dụ 2

```c
#include<stdio.h>
int main(){
    int arr[2][2][2] = {10, 2, 3, 4, 5, 6, 7, 8};
    int *p, *q;
    p = &arr[1][1][1];
    q = (int *) arr;
    printf("%d, %d\n", *p, *(q+4) );
    return 0;
}
```

8, 5

Ví dụ 3

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

Ví dụ

- Cấp phát mảng 10 phần tử kiểu int

```
int *a = (int *) malloc(10 * sizeof(int));
```

```
int *a = (int *) calloc(10, sizeof(int));
```

Con trỏ và mảng (1/2)

- 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 (2/2)

- 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 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 (1/2)

- 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 thẻ khởi tạo trực tiếp các giá trị khi khai báo

```
char * ma[10] = {"mot","hai","ba"...};
```

Mảng các con trỏ (2/2)

- 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

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;
```

- Có thể mô tả một mảng 2 chiều qua con trỏ của con trỏ theo công thức :

```
M[i][k] = *(*(M+i)+k)
```

- Trong đó:

```
M+i //là địa chỉ của phần tử thứ i của mảng
*(M+i) //cho nội dung phần tử trên
*(M+i)+k //là địa chỉ phần tử [i][k]
```

Quản lý bộ nhớ - Bộ nhớ động (1/2)

- 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 (2/2)

- 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 ta dùng các hàm **`malloc`**, **`calloc`**, **`realloc`** và **`free`** để xin cấp phát, tái cấp phát và giải phóng bộ nhớ.
- Trong C++ ta dùng **`new`** và **`delete`**

Cấp phát và giải phóng vùng nhớ

- Hàm **`malloc`**: Cấp phát 1 vùng nhớ có kích thước là `size`
  - ***Cú pháp:*** `void* malloc(size_t size);`
- Hàm **`calloc`**: Cấp phát 1 vùng nhớ chứa đủ num phần tử, mỗi phần tử có kích thước size.
  - ***Cú pháp:*** `void* calloc(size_t num, size_t size);`

=> Con trỏ sẽ trỏ tới vùng nhớ vừa được cấp phát nếu thành công, con trỏ null nếu cấp phát thất bại.

- Hàm **`free`**: Giải phóng vùng nhớ trỏ bởi con trỏ ptr
  - ***Cú pháp:*** `void free(void* ptr)`

Cấp phát bộ nhớ động (1/2)

- Cú pháp xin cấp phát bộ nhớ:

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

- hoặc

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

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.

- Giải phóng bộ nhớ

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

Cấp phát bộ nhớ động (2/2)

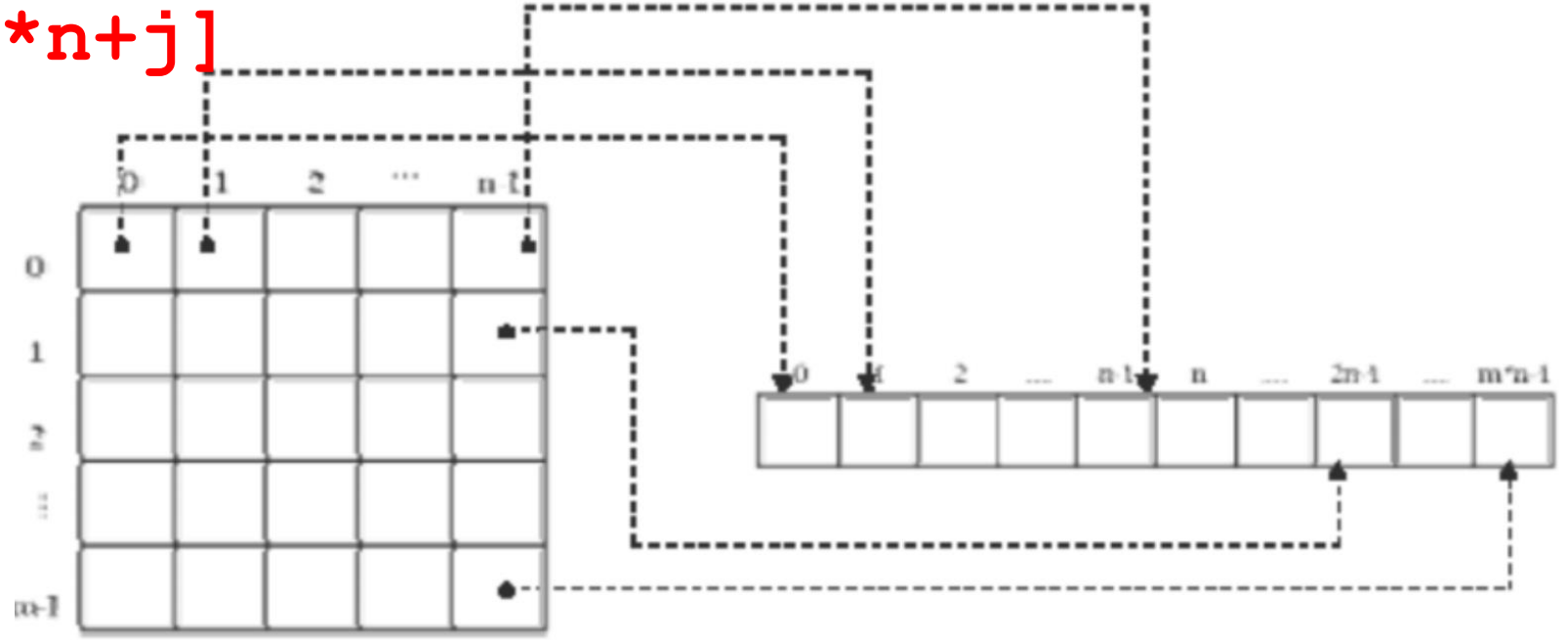
- Bộ nhớ động được quản lý bởi hệ điều hành, được chia sẻ giữa hàng loạt các ứng dụng, vì vậy có thể không đủ bộ nhớ. Khi đó toán tử new sẽ trả về con trỏ NULL.
- **Ví dụ:**

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

Bộ nhớ động cho mảng 2 chiều (1/2)

- **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]
```



Bộ nhớ động cho mảng 2 chiều (2/2)

- **Cách 2:** Dùng con trỏ của con trỏ
- Ví dụ: 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;}
```

- Để giải phóng:

```
delete [] mt[0];
delete [] mt;
```

Q&A