ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chương 2:
Quản lý bộ nhớ

Thứ tự ưu tiên các phép toán

Mức	Các toán tử	Trật tự kết hợp
1	() [] . -> ++ (hậu tố) -- hậu tố	------>
2	! ~ ++ (tiền tố) -- (tiền tố) - * & sizeof	<------
3	* / %	------>
4	+ -	------>
5	<< >>	------>
6	< <= > >=	------>
7	== !=	------>
8	&	------>
9	^	------>
10	|	------>
11	&&	------>
12	||	------>
13	?:	<------
14	= += -=	<------

Với x = 10				devc
Expression	Visual++	g++	Turbo C++	
b = x + x++	20	21	20	21
b = x++ + x	20	21	20	21
b = 2*x + x++	30	30	30	30
b = ++x + x++	22	23	22	22
b = x++ + ++x	22	22	22	22
b = x++	10	10	10	10
b = x++ + 3	13	13	13	13
b = x++ + x++	20	21	20	21
b = x++ + (++x + 2*x)	44	46	44	46
b = ++x + ++x	24	24	24	24
b = ++x + ++x + ++x	39	37	39	37
b = ++x + ++x * ++x	182	182	182	182
++b = ++b + b++	7	7	7	
++b = ++b + ++b + ++b	15	13	15	

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

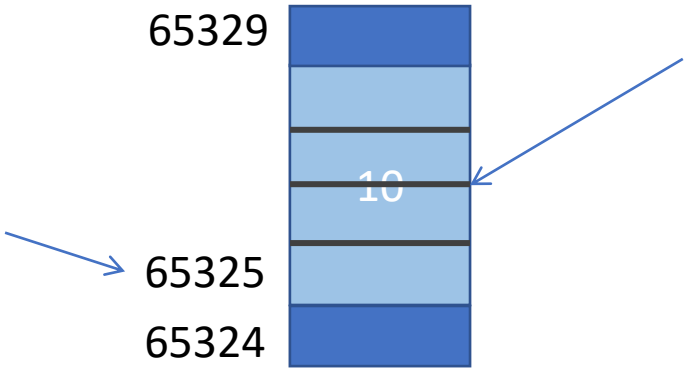
```
int x = 10;
```

- 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

Tên biến x
tham chiếu đến vị trí vùng nhớ

Địa chỉ của vị trí vùng nhớ



Giá trị được lưu trong vùng nhớ

- `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 `a`, ta nói `p` trỏ tới biến `a`

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

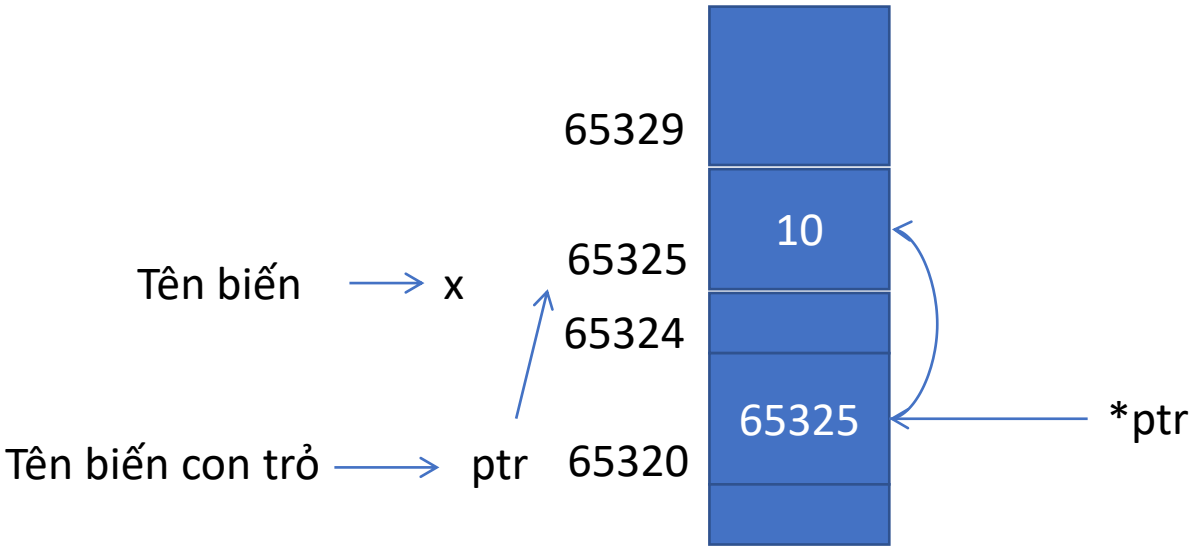
`*p` được gọi là "tham chiếu ngược" của con trỏ `p`

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

- int a,b,c; => a,b,c đều là int
- int * p,q,r; => p là con trỏ int, còn q,r là biến int
- Để cả p,q va r là con trỏ int :int *p, *q, * r;

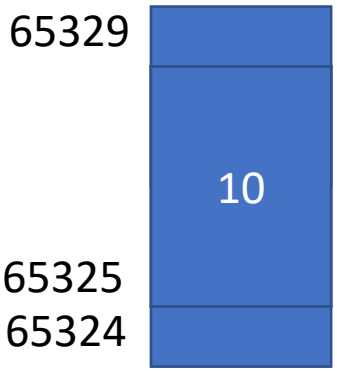
Con trỏ

```
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 = &a; p giữ giá trị địa chỉ a, i.e.65325
*p biểu diễn giá trị lưu tại p, ie.10
int b = *p; b có giá trị 10.
*p = 20; a có giá trị 20.

```
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
```

```
int x=10,*p;
p = &x;
*p+=5; => x=15
```

p+=5; => p không trỏ tới x nữa, mà trỏ tới 1 biến khác năm cách x 5 biến, x không thay đổi

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
  - p+/- n ==(giá tri p) +/- n*sizeof(*p)
- Trừ 2 con trỏ cùng kiểu cho ta khoảng cách (số phần tử) giữa 2 con trỏ
  - p-q == ((giá trị p) – (gia trị q))/sizeof(*p)
- 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 304 tương ứng

Nếu viết tiếp

```
plong += 5; => plong = 324
pchar -=10; => pchar = 91
pshort +=5; => pshort = 212
```

Chú ý

++ và – hậu tố có độ ưu tiên cao hơn * nên *p++ tương đương với *(p++) tức là tăng địa chỉ mà nó trỏ tới chứ không phải tăng giá trị mà nó chứa.

int x=100, *p=&x; // giả sử p=63244

++*p =5;

??? error

*p++ = *q++ sẽ tương đương với

*p = *q;

p=p+1;

q=q+1;

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
/* 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 một hằng địa chỉ bằng địa chỉ của phần tử đầu tiên của mảng

```
a == &a[0];
a+i == &a[i];
*(a+i) == a[i]; Khi gặp a[i] thì trình dịch đổi thành *(a+i)
```

```
{
   int i,m[10];
   for(i=1;i<12;i++) i[m]= i*5;
    for(i=1;i<12;i++) printf("\n m[%d] = %d",i,m[i]);
}
```

CT nay có vẻ có những vd gì :

- m khai bao kt = 10 ; 0-9, nhung trong ct chay tu 1 den 11
- i[m] ?
- i[m] => *(i+m) == *(m+i) == m[i]
- C không kiểm soat kích thước mảng

CÓ CẦN KHAI BÁO TƯỜNG MINH KÍCH THƯỚC MẢNG ?

```
{
   int i,n;
   int m[n];
   printf(“\n Vao kt mang “); scanf(“%d”,&n);
    for(i=0;i<n;i++) i[m]= i*5;
    for(i=0;i<n;i++) printf(“\n m[%d] = %d”,i,m[i]);
}
```

C quá hay ?

???

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]`

- Int M[10] ;

M == &M[0]

M+i== &M[i]

*(M+i) == M[i] : M[i] ➔ *(M+i)

M[i][j] ➔ *(*(M+i)+j)

Con trỏ và xâu

- Ta có `char tp[30] =“Da Lat”;`
- Tương đương :

```
char *ptp;
ptp=“Da lat”;
```

Hoặc: `char *ptp =“Da lat”;`

- Ngoài ra các thao tác trên xâu cũng tương tự như trên mảng

```
*(ptp+3) == “l”== ptp[3]
 == *(tp+3)==tp[3]
```

- Chú ý : với xâu thường thì không thể gán trực tiếp như dòng thứ 3

Tuy nhiên cần lưu ý con trỏ được cấp phát bộ nhớ hay chưa và trỏ tới đâu ?

```
int main() {
    char s1[10]="ABC", *s2="ABC",*s3;
    s3="ABC";
    printf("S1= %s, s2= %s s3= %s",s1,s2,s3);
    printf("\n s1[1]= %c s2[1]= %c s3[1]=%c",s1[1],s2[2],s3[1]);
    s3=s1;
    s1[1]='D';
//  s2[1]='D';   Loi vi "ABC" la const,
    s3[2]='F';
    printf("\n s1[1]= %c s2[1]= %c s3[1]=%c",s1[1],s2[2],s3[2]);
    printf("\n S1= %s, s2= %s s3= %s",s1,s2,s3);
    gets(s1);
//  gets(s2);  loi vi s2 chua duoc cap phat bo nho
    gets(s3);
    printf("\n S1= %s, s2= %s s3= %s",s1,s2,s3);
}
```

```
int m[10] ={1};
int m[10]={1,,,2};
int m[][] ={{1,2,},,,{1,,1,1,1,,1}};   4,7
char s[6]={‘H’,’a’,’n’,’o’,’i’};
char s2[]=“Hanoi”;
s?s2   :
```

H	a	n	o	i	\0

H	a	n	o	i	??

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 trỏ tới 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"...};
```

char s1[10]=“hjkhj”, *s2=&s1;

s2=“abc”; // s1 khong doi, chi co noi dung cua s2 thay doi

Lam the nao de qua s2 ma lam thay doi duoc s1 =“abc” ?

*s2=“abc”; => gawpj chis pheof => bij anw chui : error

strcpy(s2,”abc”);

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

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

int main () {
    int n = 0;
    char ds[MAXHS][MAXLEN];
    char *ptr[MAXHS], *tmp;
    while (n < MAXHS) {
        printf("Enther the name of student %d: ", n + 1);
        gets(ds[n]);
        if (strlen(ds[n]) == 0) break;
        ptr[n] = ds+n;
        ++n;
    }
    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;
**p2=100 ~ x=100; ?
```

- 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 nơi chứa địa chỉ hàng thứ `i` của mảng

`*(M+i)` địa chỉ hàng thứ i

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

`*(*(M+i)+k)` là giá trị `M[i][k]`

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");
    }
    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"); }
}
```

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 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 bộ nhớ động

- 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

- 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
}
```

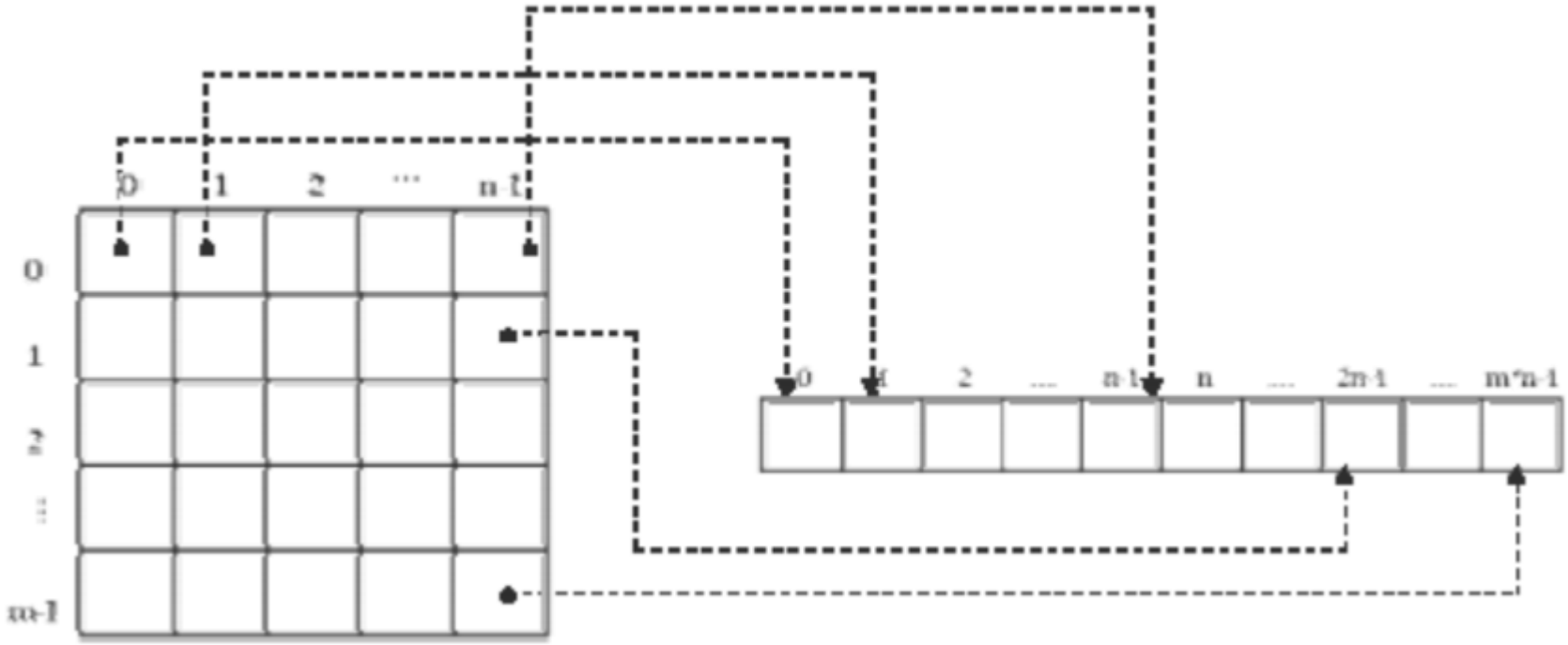
Ví dụ

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n;
    long total=100, x, *arr;
    printf("Nhap n = "); scanf("%d",&n);
    arr = new long [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]
```



Bộ nhớ động cho mảng 2 chiều

- Cách 2: Dùng con trỏ của con trỏ - Cách cài đặt của C
- 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;
}
Dung mt[i][j] nhu 1 mang 2 chieu bt
```

- Để giải phóng:

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

Bộ nhớ động cho mảng 2 chiều

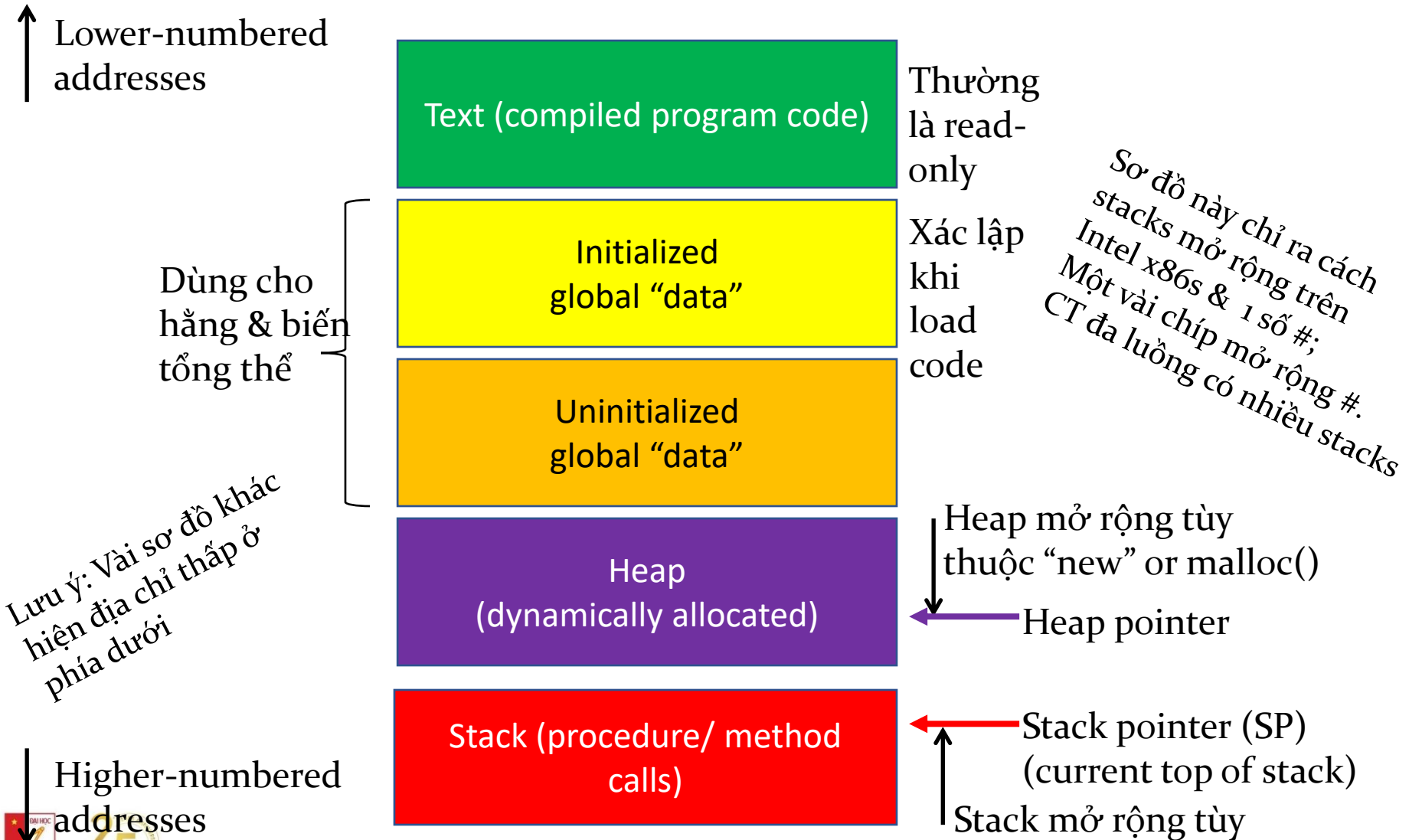
- Ví dụ khác để cấp phát động cho mảng hai chiều chứa các số thực
  float - Ma trận răng cưa ?

```
// Khởi tạo ma trận với  R hàng và C cột
float ** M = new float *[R];
for (i=0; i < R; i++)
      M[i] = new float[C];
// Dùng M[i][j] cho các phần tử của ma trận

// Giải phóng
for(i=0; i<R; i++)
      // Giải phóng các hàng
      delete [] M[i];
delete [] M;
```

- Ưu nhược điểm của mỗi phương pháp ?

Sơ đồ bộ nhớ


Lower-numbered addresses
Text (compiled program code)
Thường là read-only
Initialized global "data"
Uninitialized global "data"
Dùng cho hằng & biến tổng thể
Xác lập khi load code
Sơ đồ này chỉ ra cách stacks mở rộng trên Intel x86s & 1 số #; Một vài chíp mở rộng #. CT đa luồng có nhiều stacks
Lưu ý: Vài sơ đồ khác hiện địa chỉ thấp ở phía dưới
Heap (dynamically allocated)
Heap mở rộng tùy thuộc "new" or malloc()
Heap pointer
Stack (procedure/ method calls)
Stack pointer (SP) (current top of stack)
Stack mở rộng tùy thuộc lời gọi hàm
Higher-numbered addresses

Ví dụ

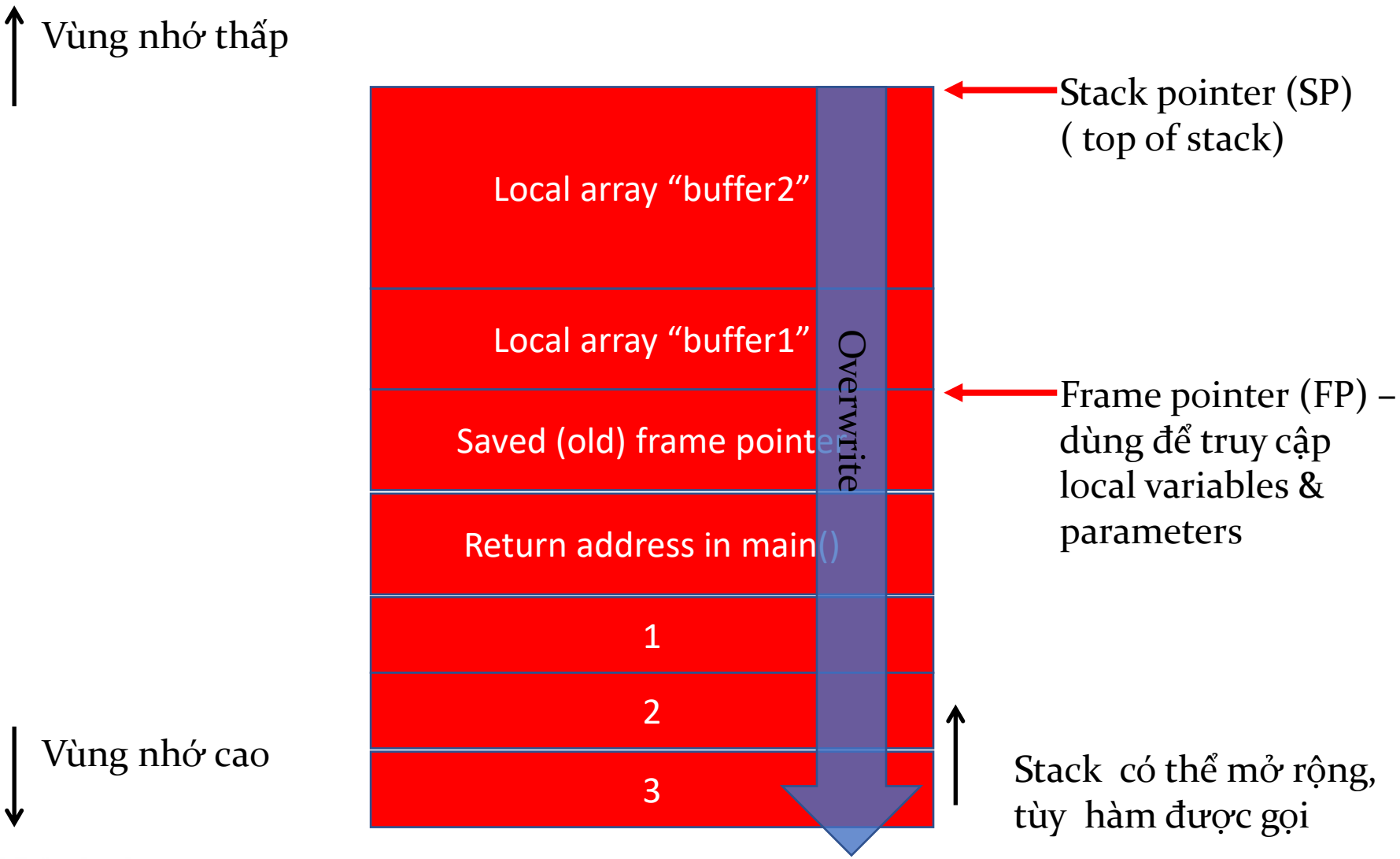
```
void main() {        f(1,2,3);            }
void f(int a, int b, int c) {
            char buffer1[5];
            char buffer2[10];
            strcpy(buffer2, "This is a very long string!!!!!!!");
}
```

- Lời gọi hàm f() sẽ thực hiện như sau:

```
pushl $3 ; đối số 3
pushl $2 ; Hầu hết C _Compiler đẩy vào stack theo thứ tự ngược
pushl $1
call f
```

- Lời gọi f() sẽ đẩy con trỏ lệnh -intruction pointer (IP) vào stack
  - Trong ví dụ trên, là vị trí trong “main()” ngay sau f(…)
  - Con trỏ lệnh đc lưu gọi là return address (RET)
  - CPU sau đó bắt đầu thực hiện hàm

Stack:


Vùng nhớ thấp
Local array “buffer2”
Local array “buffer1”
Saved (old) frame pointer
Return address in main()
1
2
3
Overwrite
Stack pointer (SP)
(top of stack)
Frame pointer (FP) – dùng để truy cập local variables & parameters
Vùng nhớ cao
Stack có thể mở rộng, tùy hàm được gọi

Bài tập

- Viết chương trình quản lý danh sách họ và tên SV đơn giản
- Gồm menu :
  - 1. Bổ sung danh sách
  - 2. Sắp xếp danh sách theo thứ tự ABC
  - 3. in danh sách
  - Bấm phím 1-3 để chọn , phím bất kỳ # để KT
- Biết rằng số SV tói đa là 100, hotenSV dài tối đa 30
- XD thành nhiều version, từ đơn giản đến phức tạp
  - Dùng mảng tĩnh
  - Dùng mảng động
  - Sắp xếp theo kiểu việt nam
  - Thay thế ds tên thành ds hồ sơ sv (struct)
  - …

Bt tiep…

- Xem ct + ma tran dung bo nho dong voi mang 1 chieu tren slide cu
- Van dung cach lam trong vi du, xd ct + ma tran voi bo nho dong dung mang 2 chieu (cach 2,3)
- Tuong tu : nhan ma tran
- XD ham alloc3DArray, 4DArray
- C++ chi co new va delete de cap phat va giai phong mang, neu muon tai cap phat lai bo nho cho cac mang 1,2,3… chieu ma van muon giu lai cac gia tri da co cua mang cu
- Khong the gan 2 mang, nhung co the gan 2 bien cau truc, du cac truong cua cau truc co the la 1 mang, xau, hay 1 struct khac. Nhung neu 1 truong cua cau truc la con tro va duoc cap phat dong thi sao ???

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Xin cảm ơn!