



# BIẾN CON TRỎ

KỸ THUẬT LẬP TRÌNH

GVHD: Trương Toàn Thịnh

# NỘI DUNG

- Sử dụng kiểu vector<T>
- Biến con trỏ
- Kiểu dữ liệu chuỗi
- Các hàm khai thác bộ nhớ động
- Phép toán trên con trỏ
- Các kỹ thuật áp dụng con trỏ
- Bài tập

# SỬ DỤNG KIỂU VECTOR<T>

- Nằm trong thư viện chuẩn Standard Template Library (STL)
- Được cài đặt sẵn cho C++ chuẩn
- Có khả năng thay đổi kích thước mảng
- Có thể sử dụng với nhiều kiểu dữ liệu khác nhau
- Có thể khai báo kiểu lồng cho các dạng mảng nhiều chiều

# SỬ DỤNG KIỂU VECTOR<T>

- Xét ví dụ 1

Dòng		Dòng	
1	<code>#include &lt;iostream&gt;</code>	12	<code>if(n &lt; 1) return;</code>
2	<code>#include &lt;vector&gt;</code>	13	<code>a.resize(n);</code>
3	<code>using namespace std;</code>	14	<code>for(int i = 0; i &lt; a.size(); i++)</code>
4		15	<code>cin &gt;&gt; a[i];</code>
5	<code>void arrayOutput(vector&lt;float&gt; &amp;a){</code>	16	<code>}</code>
6	<code>for(int i = 0; i &lt; a.size(); i++)</code>	17	
7	<code>cout &lt;&lt; a[i] &lt;&lt; “ ”;</code>	18	<code>void main(){</code>
8	<code>}</code>	19	<code>vector&lt;float&gt; a;</code>
9		20	<code>arrayInput(a);</code>
10	<code>void arrayInput(vector&lt;float&gt; &amp;a){</code>	21	<code>arrayOutput(a);</code>
11	<code>int n; cin &gt;&gt; n;</code>	22	<code>}</code>

# SỬ DỤNG KIỂU VECTOR<T>

- Ví dụ 1 cần biết số lượng phần tử sử dụng
- Xét ví dụ 2

Dòng		Dòng	
1	<code>#include &lt;iostream&gt;</code>	10	<code>float x;</code>
2	<code>#include &lt;vector&gt;</code>	11	<code>while(cin &gt;&gt; x)</code>
3	<code>using namespace std;</code>	12	<code>  a.push_back(x);</code>
4		13	<code>  } // <b>cin.clear();</b></code>
5	<code>void arrayOutput(const vector&lt;float&gt; &amp;a){</code>	14	<code>void main(){</code>
6	<code>  for(int i = 0; i &lt; a.size(); i++)</code>	15	<code>  vector&lt;float&gt; a;</code>
7	<code>    cout &lt;&lt; a[i] &lt;&lt; “ ”;</code>	16	<code>  arrayInput(a);</code>
8	<code>  }</code>	17	<code>  arrayOutput(a);</code>
9	<code>void arrayInput(vector&lt;float&gt; &amp;a){</code>	18	<code>}</code>

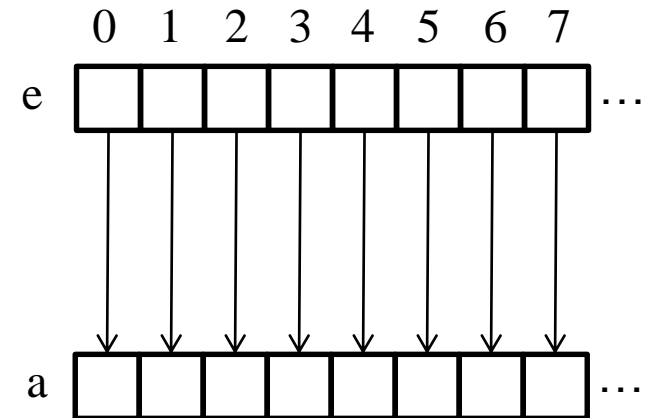
# SỬ DỤNG KIỂU VECTOR<T>

- Một số phương thức cơ bản của kiểu `vector<T>`
  - `size()`: trả về số phần tử hiện có của mảng
  - `resize(int)`: thay đổi kích thước mảng, tự động thêm/xóa bớt phần tử
  - `push_back(T)`: thêm một phần tử vào sau mảng
  - `pop_back()`: xóa phần tử cuối mảng, kích thước mảng giảm đi một

# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng một số hàm cơ bản
  - Hàm tạo vector từ một mảng số nguyên

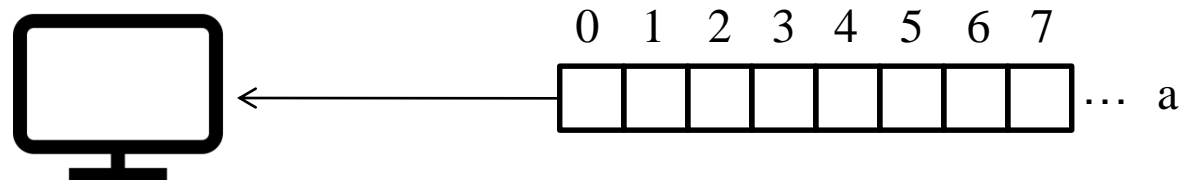
Dòng	
1	<code>void intArrayMake(vector&lt;int&gt; &amp;a, int e[], int n){</code>
2	<code>int i = 0;</code>
3	<code>a.resize(0);</code>
4	<code>while(i &lt; n){</code>
5	<code>  a.push_back(e[i]);</code>
6	<code>  i++;</code>
7	<code>}</code>
8	<code>}</code>



# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng một số hàm cơ bản
  - Hàm xuất vector ra thiết bị xuất

Dòng	
1	<code>void intArrayOut(vector&lt;int&gt; &amp;a, ostream&amp; outDev){</code>
2	<code>for(int i = 0; i &lt; a.size(); i++)</code>
3	<code>outDev &lt;&lt; a[i] &lt;&lt; “ ”;</code>
4	<code>outDev &lt;&lt; endl;</code>
5	<code>}</code>

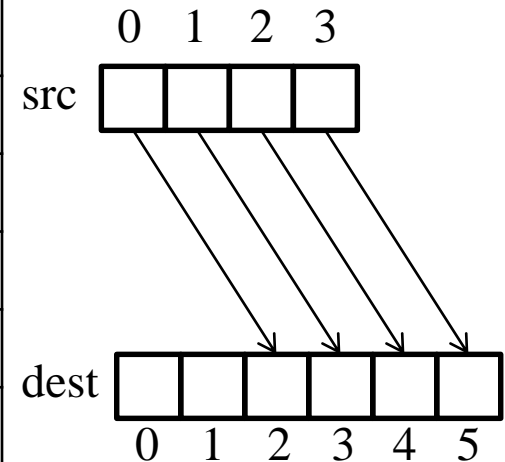




# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng một số hàm cơ bản
  - Hàm nối hai vector

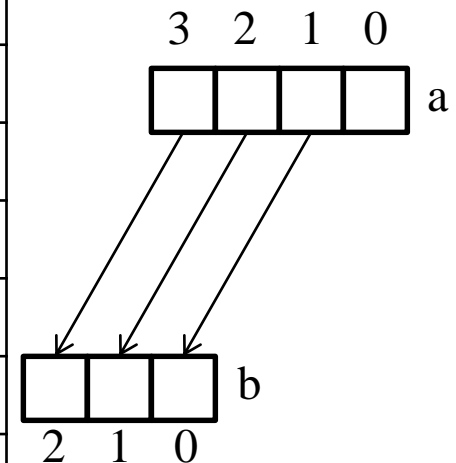
Dòng	
1	<code>void intArrayCat(vector&lt;int&gt; &amp;dest, vector&lt;int&gt; &amp;src){</code>
2	<code>int s1 = dest.size(), s2 = src.size();</code>
3	<code>int idx = s1, newsize = s1 + s2, i = 0;</code>
4	<code>dest.resize(newsize);</code>
5	<code>while(i &lt; s2){</code>
6	<code>dest[idx] = src[i];</code>
7	<code>idx++; i++;</code>
8	<code>}</code>
9	<code>}</code>



# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng một số hàm cơ bản
  - Hàm cắt vector theo chỉ số và chuyển sang vector khác

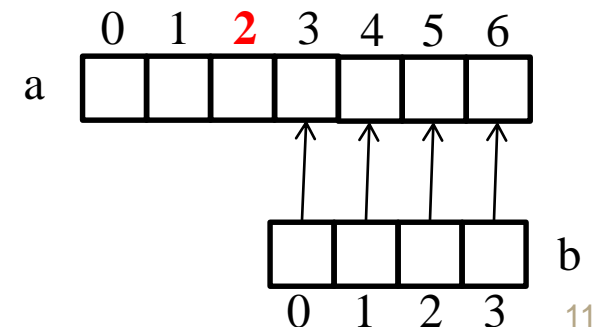
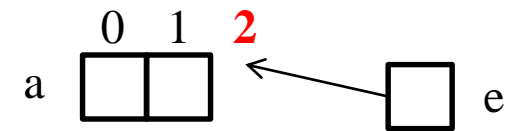
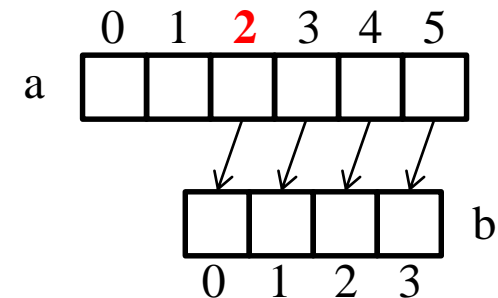
Dòng	
1	<code>void intArrayCut(vector&lt;int&gt; &amp;a, int pos, vector&lt;int&gt; &amp;b){</code>
2	<code>int size = a.size(), j = pos;</code>
3	<code>if(j &lt; 0    j &gt;= size) return;</code>
4	<code>b.resize(0);</code>
5	<code>while(j &lt; size){</code>
6	<code>  b.push_back(a[j]);</code>
7	<code>  j++;</code>
8	<code>}</code>
9	<code>a.resize(pos);}</code>



# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng một số hàm cơ bản
  - Hàm chèn một phần tử vào vector theo vị trí

Dòng	
1	<code>void intArrayInsert(vector&lt;int&gt; &amp;a, int pos, int e){</code>
2	<code>if(pos &lt; 0    pos &gt;= a.size()) return;</code>
3	<code>vector&lt;int&gt; b;</code>
4	<code>intArrayCut(a, pos, b);</code>
5	<code>a.push_back(e);</code>
6	<code>intArrayCat(a, b);</code>
7	<code>}</code>



# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng một số hàm cơ bản
  - Hàm main minh họa sử dụng

Dòng	
1	<code>void main(){</code>
2	<code>int x[] = {3, 5, 2, 4, 9, 7, 8, 6};</code>
3	<code>int n = sizeof(x)/sizeof(x[0]);</code>
4	<code>vector&lt;int&gt; a, b, c;</code>
5	<code>intArrayMake(a, x, n);</code>
6	<code>intArrayOut(a, cout);</code>
7	<code>intArrayCut(a, 3, b); intArrayOut(a, cout); intArrayOut(b,cout);</code>
8	<code>intArrayCat(b, a);</code>
9	<code>intArrayInsert(b, 3, 999); intArrayOut(b, cout);</code>
10	<code>}</code>

# SỬ DỤNG KIỂU VECTOR<T>

- Xây dựng với struct

```
struct pupil{  
    char name[31];  
    char code[6];  
    float grade;  
};  
typedef struct pupil PUPIL;
```

→ Toán tử '>>' để lại kí tự '↵' => ta cần *cin.ignore* để đọc lấy '↵' ra

```
void inputPupil(PUPIL &p){  
    cin>>p.grade;  
    cin.ignore();  
    cin.getline(p.code, 6);  
    cin.getline(p.name, 31);  
}
```

```
void inputPupil(PUPIL &p){  
    cin.getline(p.code, 6);  
    cin.getline(p.name, 31);  
    cin>>p.grade;  
}
```

Phương thức *getline* của *cin* đọc kí tự '↵' ra

# SỬ DỤNG KIỂU VECTOR<T>

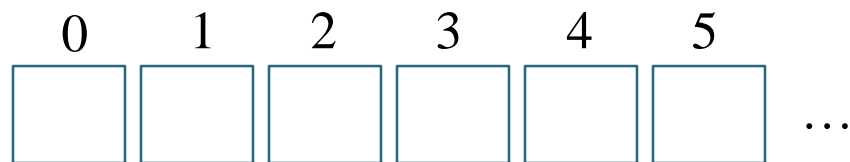
- Xây dựng với struct

Dòng			
1	<code>#include &lt;iostream&gt;</code>	12	<code>while(inputPupil(x)) a.push_back(x);</code>
2	<code>#include &lt;vector&gt;</code>	13	<code>cin.clear();</code>
3	<code>using namespace std;</code>	14	<code>}</code>
4		15	
5	<code>void arrayOutput(vector&lt;PUPIL&gt; a){</code>	16	<code>int inputPupil(PUPIL &amp;p){</code>
6	<code>for(int i = 0; i &lt; a.size(); i++)</code>	17	<code>cin&gt;&gt;p.grade;</code>
7	<code>outputPupil(a[i]);</code>	18	<code>cin.ignore();</code>
8	<code>}</code>	19	<code>cin.getline(p.code, 6);</code>
9		20	<code>cin.getline(p.name, 31);</code>
10	<code>void arrayInput(vector&lt;float&gt; &amp;a){</code>	21	<code>return cin.good();</code>
11	<code>PUPIL x;</code>	22	<code>}</code>

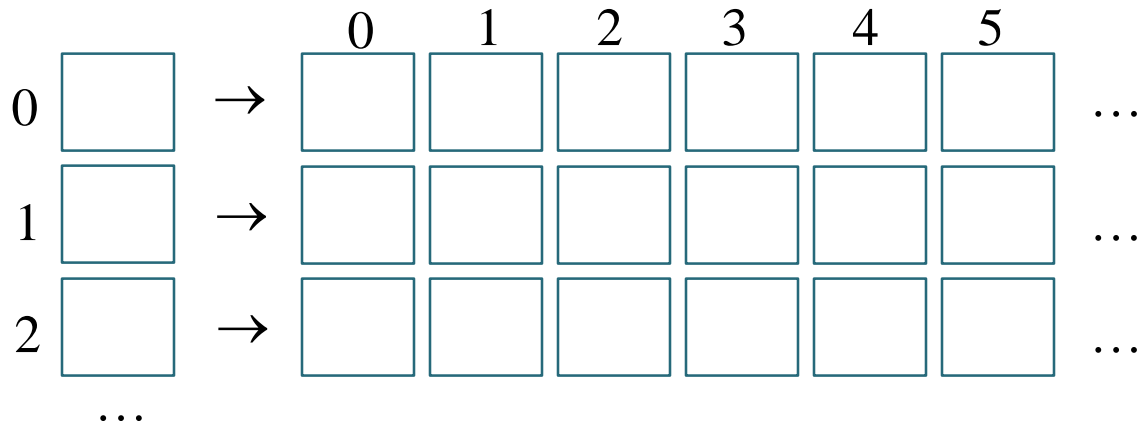
# SỬ DỤNG KIỂU VECTOR<T>

- Khai báo lòng kiểu `vector<T>` để tạo mảng hai chiều

`typedef vector<float> Floats;`



`typedef vector<Floats> float2D;`



# SỬ DỤNG KIỂU VECTOR<T>

- Khai báo lồng kiểu vector<T> để tạo mảng hai chiều

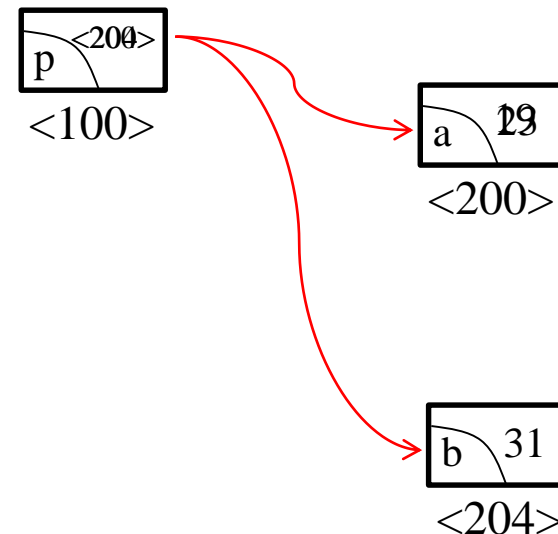
Dòng		Dòng	
1	<code>#include &lt;iostream&gt;</code>	10	<code>void float2DInput(float2D &amp;a){</code>
2	<code>#include &lt;vector&gt;</code>	11	<code>for(int i = 0; i &lt; a.size(); i++)</code>
3	<code>using namespace std;</code>	12	<code>for(int j = 0; j &lt; a[i].size(); j++)</code>
4	<code>typedef vector&lt;float&gt; Floats;</code>	13	<code>cin &gt;&gt; a[i][j];</code>
5	<code>typedef vector&lt;Floats&gt; float2D;</code>	14	<code>}</code>
6		15	<code>void float2DOutput(float2D &amp;a){</code>
7	<code>void float2DInit(float2D &amp;a, int n){</code>	16	<code>for(int i = 0; i &lt; a.size(); i++)</code>
8	<code>a.resize(n);</code>	17	<code>for(int j = 0; j &lt; a[i].size(); j++){</code>
9	<code>for(int i = 0; i &lt; n; i++) a[i].resize(n);</code>	18	<code>cout &lt;&lt; a[i][j] &lt;&lt; "\t";}</code>
10	<code>}</code>		<code>cout &lt;&lt; endl;}</code>



# BIẾN CON TRỎ

- Dùng để lưu địa chỉ vùng nhớ hợp lệ
- Cũng có kiểu dữ liệu như biến thông thường

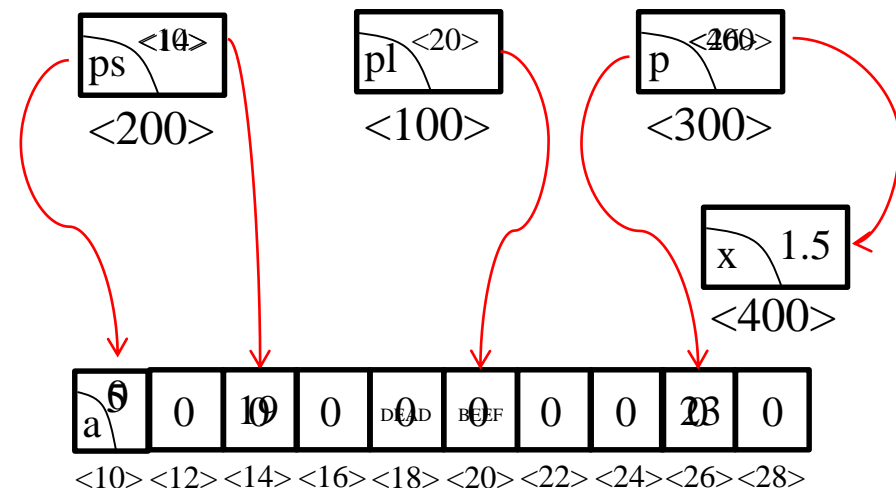
Dòng	
1	<code>void main(){</code>
2	<code>int *p;</code>
3	<code>int a = 19, b;</code>
4	<code>p = &amp;a;</code>
5	<code>*p = 23;</code>
6	<code>p = &amp;b;</code>
7	<code>*p = 31;</code>
8	<code>}</code>



# BIẾN CON TRỞ

- Lưu địa chỉ vùng nhớ của phần tử mảng
- Có thể khai báo con trỏ vô kiểu (**void\***)

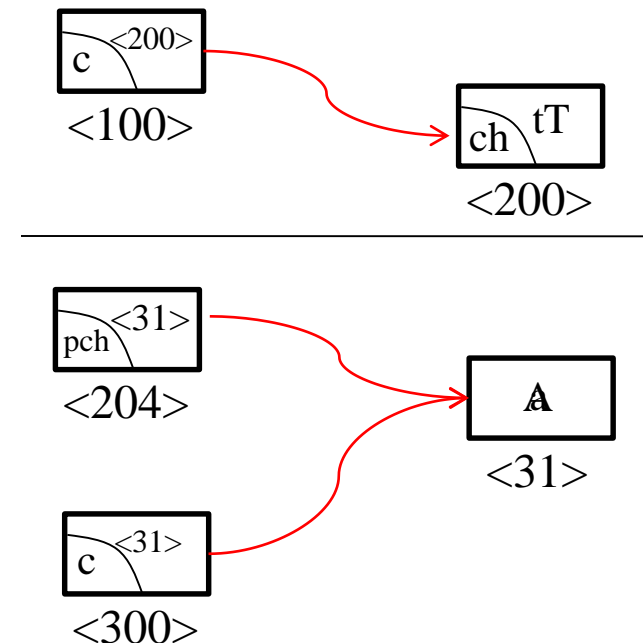
Dòng	
1	<b>void</b> main(){
2	float x; <b>unsigned short</b> a[10] = {0};
3	<b>unsigned short</b> *pshort;
4	<b>unsigned long</b> *plong; <b>void*</b> p
5	pshort = a; *pshort = 5;
6	pshort = &a[2]; *pshort = 19;
7	p = &x; *( <b>float*</b> )p = 1.5F;
8	p = &a[8]; *( <b>unsigned short*</b> )p = 23;
9	plong = ( <b>unsigned long*</b> )&a[5];
10	*plong = 0xDEADBEEF;}



# BIẾN CON TRỞ

- Hàm với tham số con trỏ có hai loại
  - Con trỏ tham trị
  - Con trỏ tham chiếu

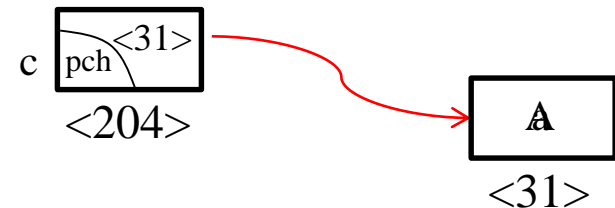
Dòng	
1	<code>void upperCase(unsigned char* c){</code>
2	<code>if(*c &gt;= 'a' &amp;&amp; *c &lt;= 'z')</code>
3	<code>    *c = (*c) - 32;</code>
4	<code>}</code>
5	<code>void main(){</code>
6	<code>    unsigned char ch, *pch = new unsigned char;</code>
7	<code>    scanf("%c", &amp;ch); scanf("%c", pch);</code>
8	<code>    upperCase(&amp;ch); upperCase(pch);</code>
9	<code>    printf("%c", ch); printf("%c", pch)}</code>



# BIẾN CON TRỎ

- Hàm với tham số con trỏ có hai loại
  - Con trỏ tham trị
  - Con trỏ tham chiếu

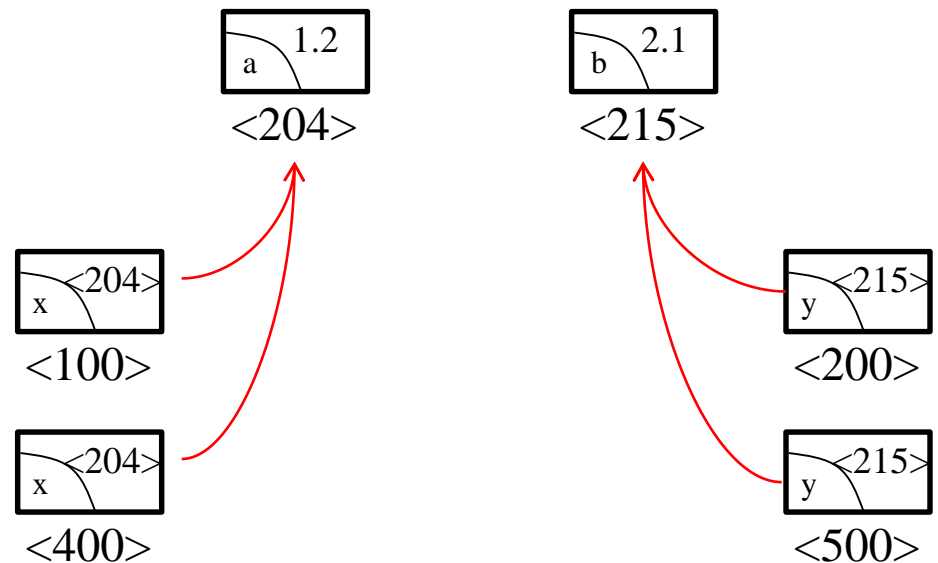
Dòng	
1	<code>void upperCase(unsigned char*&amp; c){</code>
2	<code>if(*c &gt;= 'a' &amp;&amp; *c &lt;= 'z')</code>
3	<code>    *c = (*c) - 32;</code>
4	<code>}</code>
5	<code>void main(){</code>
6	<code>    unsigned char *pch = new unsigned char;</code>
7	<code>    scanf("%c", pch);</code>
8	<code>    upperCase(pch);</code>
9	<code>    printf("%c", pch)}</code>



# BIẾN CON TRỞ

- Truyền tham số con trỏ qua nhiều hàm

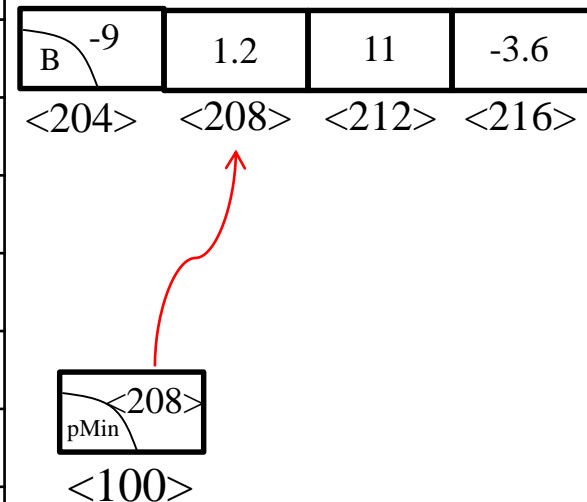
Dòng	
1	<code>void swap (float* x, float* y){</code>
2	<code>float u = *x; *x = *y; *y = u;</code>
3	<code>}</code>
4	<code>void adjust(float* x, float* y){</code>
5	<code>if(fabs(*x) &gt; fabs(*y))</code>
6	<code>swap(&amp;(*x), &amp;(*y));</code>
7	<code>}</code>
8	<code>void main(){</code>
9	<code>float a = 1.2F, b = 2.1F;</code>
10	<code>adjust(&amp;a, &amp;b);</code>
11	<code>cout&lt;&lt;*a &lt;&lt; *b &lt;&lt; endl;</code>
12	<code>}</code>



# BIẾN CON TRỎ

- Trả về giá trị con trỏ

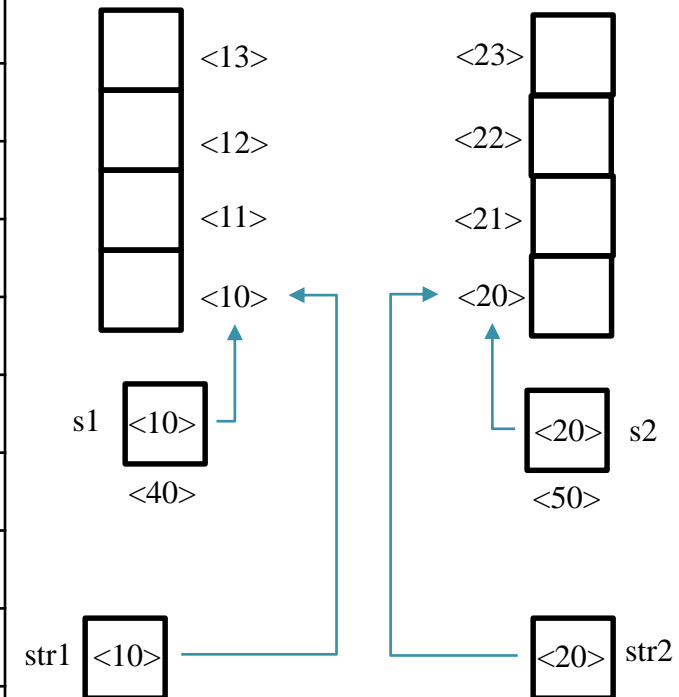
Dòng	
1	<code>float* pointerMin (float a[], int n){</code>
2	<code>int i = 1, idx = 0;</code>
3	<code>while(i &lt; n){</code>
4	<code>if(fabs(a[i]) &lt; fabs(a[idx])) idx = i;</code>
5	<code>i++;</code>
6	<code>}</code>
7	<code>return &amp;a[idx];</code>
8	<code>}</code>
9	<code>void main(){</code>
10	<code>float B[] = {-9, 1.2F, 11, -3.6F}; int n = sizeof(B)/sizeof(B[0]);</code>
11	<code>float* pMin = pointerMin(B, n);</code>
12	<code>cout &lt;&lt; *pMin &lt;&lt; endl;}</code>



# BIẾN CON TRỞ

- Trả về giá trị dạng con trỏ: có thể dùng giao tiếp với các hàm trong <string>

Dòng	
1	<code>char* strmax(char* str1, char* str2){</code>
2	<code>if(strcmp(str1, str2) &gt; 0) return str1;</code>
3	<code>return str2;</code>
4	<code>}</code>
5	
6	<code>void main{</code>
7	<code>char* s1 = new char[256], *s2 = new char[256];</code>
8	<code>cin.getline(s1, 256); cin.getline(s2, 256);</code>
9	<code>cout &lt;&lt; strupr(strmax(s1, s2));</code>
10	<code>}</code>



# BIẾN CON TRỎ

- Trả về giá trị dạng tham chiếu

Dòng	
1	<code>float&amp; refMin (float a[], int n){</code>
2	<code>int i = 1, idx = 0;</code>
3	<code>while(i &lt; n){</code>
4	<code>if(fabs(a[i]) &lt; fabs(a[idx])) idx = i;</code>
5	<code>i++;</code>
6	<code>}</code>
7	<code>return a[idx];</code>
8	<code>}</code>
9	<code>void main(){</code>
10	<code>float B[] = {-9, 1.2F, 11, -3.6F}; int n = sizeof(B)/sizeof(B[0]);</code>
11	<code>float&amp; rMin = refMin(B, n); // rMin = ...;</code>
12	<code>cout &lt;&lt; rMin &lt;&lt; endl;}</code>

B	-9	1.2	11	-3.6
<204>	<208>	<212>	<216>	
	rMin			



# Kiểu dữ liệu chuỗi

- Là mảng một chiều các phần tử kiểu **char**
- Có thể dùng con trỏ xin cấp phát chuỗi
- Kiểu chuỗi so sánh theo thứ tự từ điển
- Hàm có thể trả ra con trỏ (địa chỉ chuỗi)
- Nhiều hàm hỗ trợ các thao tác xử lý chuỗi
  - Sao chép chuỗi con của chuỗi cha ra ngoài
  - Nối hai chuỗi
  - Phát hiện chuỗi con
  - Đếm số lần xuất hiện của chuỗi con
  - ...

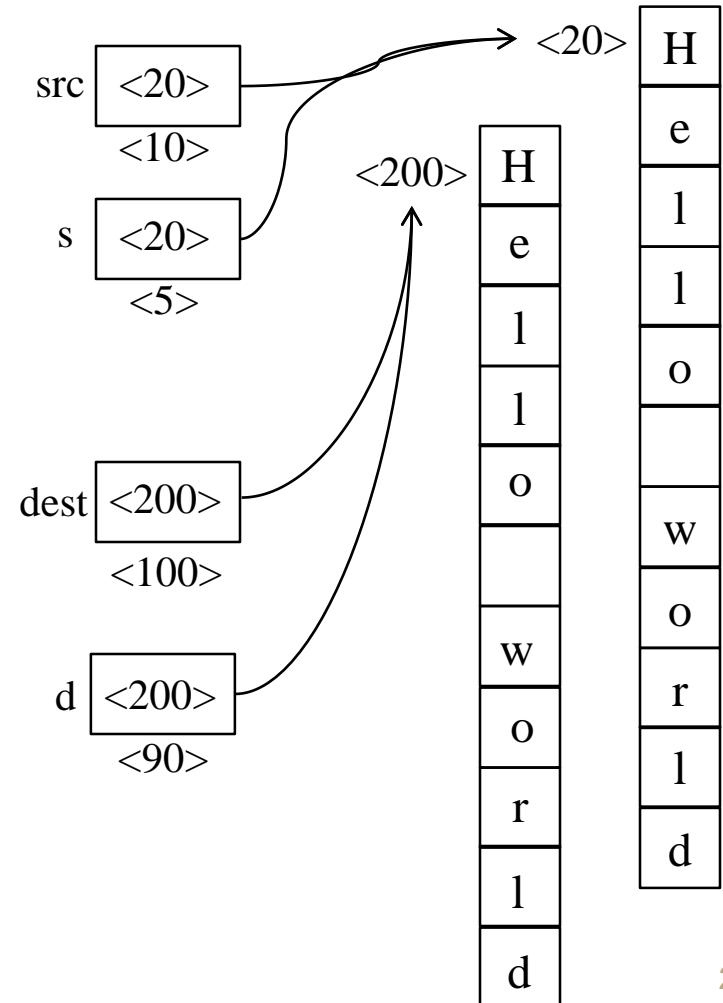
# KIỂU DỮ LIỆU CHUỖI

- Thao tác sao chép chuỗi: các tham số
  - `char*` d: địa chỉ chuỗi kết quả
  - `char*` s: địa chỉ chuỗi nguồn
  - `char*`: giá trị trả ra địa chỉ chuỗi kết quả
- Xét hai tình huống
  - Tình huống một: con trỏ d có địa chỉ trước khi vào hàm
  - Tình huống hai: con trỏ d chưa có địa chỉ trước khi vào hàm

# KIỂU DỮ LIỆU CHUỖI

- Tình huống một: con trỏ d có địa chỉ trước khi vào hàm

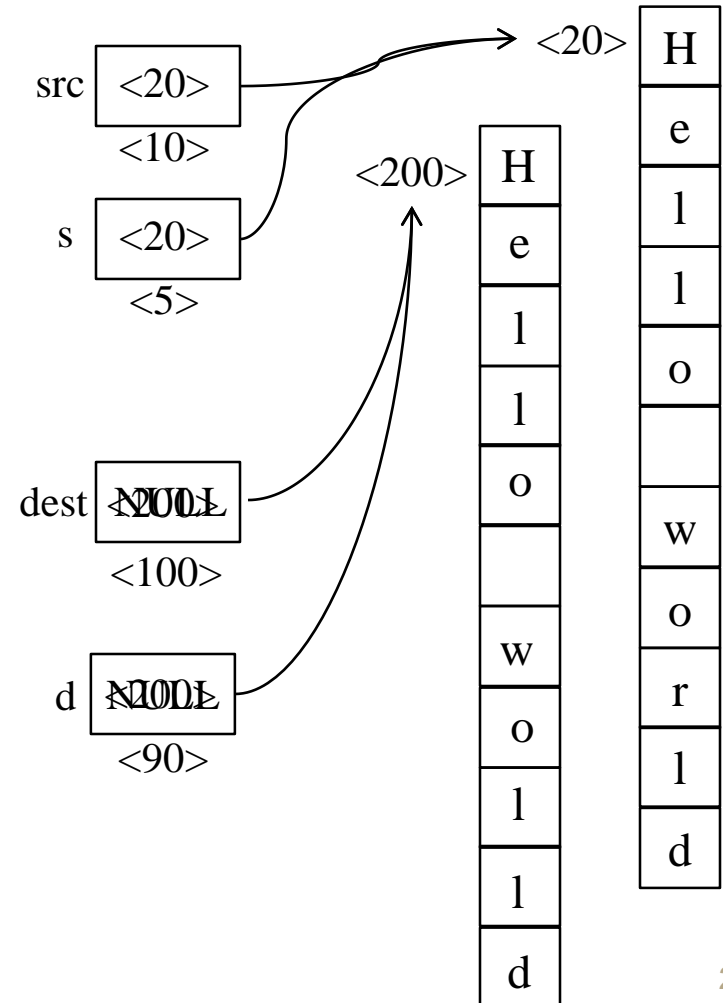
Dòng	Mô tả
1	<code>char* strCopyPB1(char* d, char* s){</code>
2	<code>int i, n = strlen(s);</code>
3	<code>for(i = 0; i &lt; n; i++) d[i] = s[i];</code>
4	<code>d[i] = '\0';</code>
5	<code>return d;</code>
6	<code>}</code>
7	<code>void main(){</code>
8	<code>char* src = "Hello world", *dest = new char[12];</code>
9	<code>strCopyPB1(dest, src);</code>
10	<code>cout &lt;&lt; dest &lt;&lt; endl;</code>
11	<code>delete[] dest;</code>
12	<code>}</code>



# KIỂU DỮ LIỆU CHUỖI

- Tình huống hai: con trỏ d chưa có địa chỉ trước khi vào hàm

Dòng	Mô tả
1	<code>char* strCopyPB2(char* d, char* s){</code>
2	<code>int i, n = strlen(s); d = new char[n + 1];</code>
3	<code>for(i = 0; i &lt; n; i++) d[i] = s[i];</code>
4	<code>d[i] = '\0';</code>
5	<code>return d;</code>
6	<code>}</code>
7	<code>void main(){</code>
8	<code>char* src = "Hello world", *dest = NULL;</code>
9	<code>dest = strCopyPB2(dest, src);</code>
10	<code>cout &lt;&lt; dest &lt;&lt; endl;</code>
11	<code>delete[] dest;</code>
12	<code>}</code>



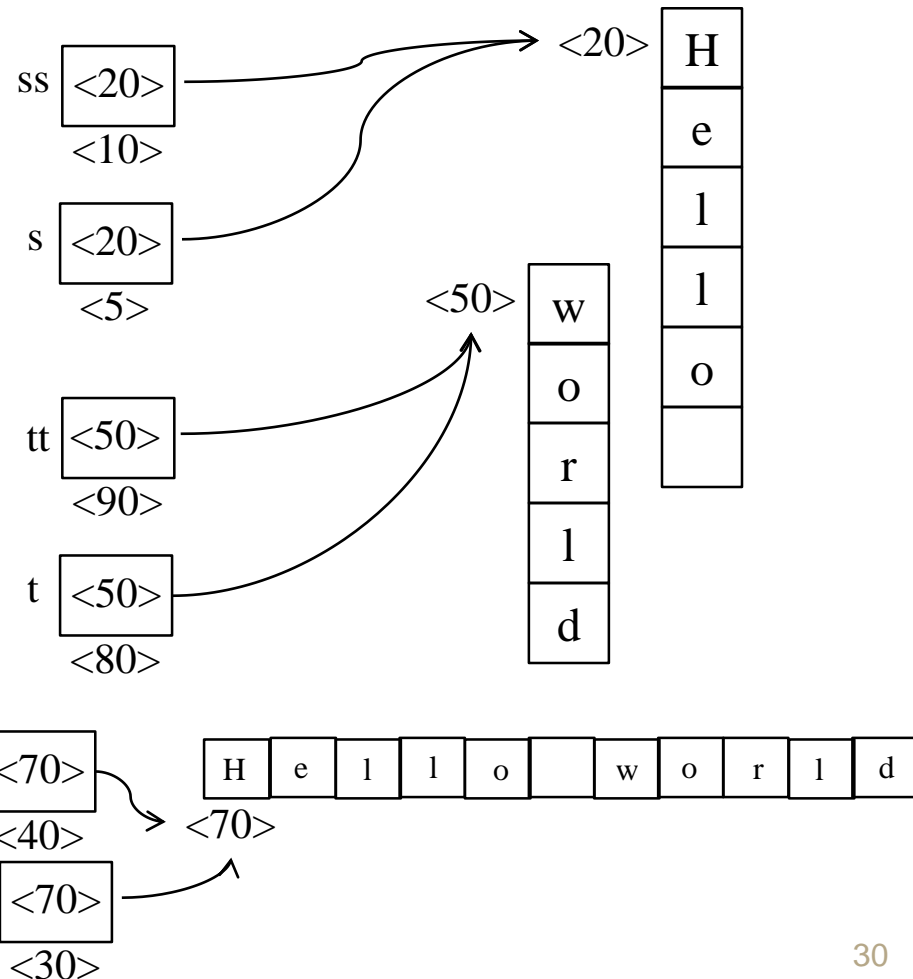
# KIỂU DỮ LIỆU CHUỖI

- Thao tác nối hai chuỗi thành một chuỗi
  - `char*` s: địa chỉ chuỗi thứ nhất
  - `char*` t: địa chỉ chuỗi thứ hai
  - `char*`: giá trị trả ra địa chỉ chuỗi kết quả
- Một số lưu ý
  - Độ dài chuỗi kết quả bằng tổng độ dài các chuỗi con
  - Thêm phân tử '\0' vào ô cuối cùng của chuỗi kết quả
  - Cần có biến con trỏ nhận địa chỉ trả về trong hàm main

# KIỂU DỮ LIỆU CHUỖI

- Đoạn mã minh họa

Dòng	Mô tả
1	<code>char* strCatenate(char* s, char* t){</code>
2	<code>int ns = strlen(s), nt = strlen(t), d = 0;</code>
3	<code>char* r = new char[ns + nt + 1];</code>
4	<code>for(int i = 0; i &lt; ns; i++) r[d++] = s[i];</code>
5	<code>for(int i = 0; i &lt; nt; i++) r[d++] = t[i];</code>
6	<code>r[d] = '\0';</code>
7	<code>return r;}</code>
8	<code>void main(){</code>
9	<code>char* ss = "Hello ", *tt = "world";</code>
10	<code>char* rr = strCatenate(ss, tt);</code>
11	<code>cout &lt;&lt; rr &lt;&lt; endl;</code>
12	<code>delete[] rr;</code>
13	<code>}</code>



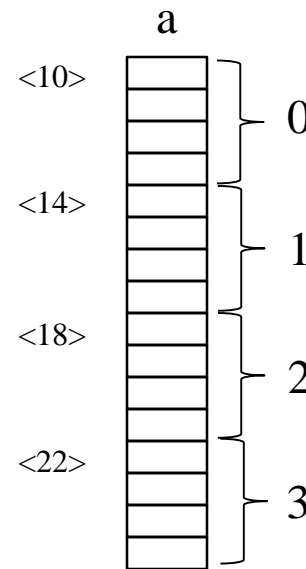
# CÁC HÀM KHAI THÁC BỘ NHỚ ĐỘNG

- C++ hỗ trợ một số hàm cấp phát bộ nhớ
  - `void*` `malloc(int n)`: xin cấp phát `n` bytes bộ nhớ
  - `void*` `calloc(int nItem, int n)`: xin cấp phát `nItem` phần tử liên tục, mỗi phần tử có kích thước `n` bytes
  - `void*` `realloc(void* pmem, int size)`: cấp phát lại vùng nhớ đã cấp phát trước đó, kích thước `size` có thể lớn hay nhỏ hơn kích thước cũ
  - `void` `free(void* pmem)`: Giải phóng vùng nhớ

# CÁC HÀM KHAI THÁC BỘ NHỚ ĐỘNG

- Cấp phát mảng một chiều

Dòng	
1	<code>#include &lt;stdio.h&gt;</code>
2	<code>#include &lt;stdlib.h&gt;</code>
3	<code>void main(){</code>
4	<code>int n; float* a = NULL;</code>
5	<code>scanf("%d", &amp;n);</code>
6	<code>a = (float*)malloc(n*sizeof(float));</code>
7	<code>if(a == NULL) return;</code>
8	<code>for(int i = 0; i &lt; n; i++)</code>
9	<code>scanf("%f", &amp;a[i]);</code>
10	<code>for(int i = 0; i &lt; n; i++)</code>
11	<code>printf("%d ", a[i]);</code>
12	<code>free(a);}</code>



Lưu ý:

$a = \&a[0]$  và  $(a + i) = \&a[i]$

$*a = a[0]$  và  $*(a + i) = a[i]$

Phép cộng/trừ địa chỉ:  $a + i \times \text{sizeof(float)}$

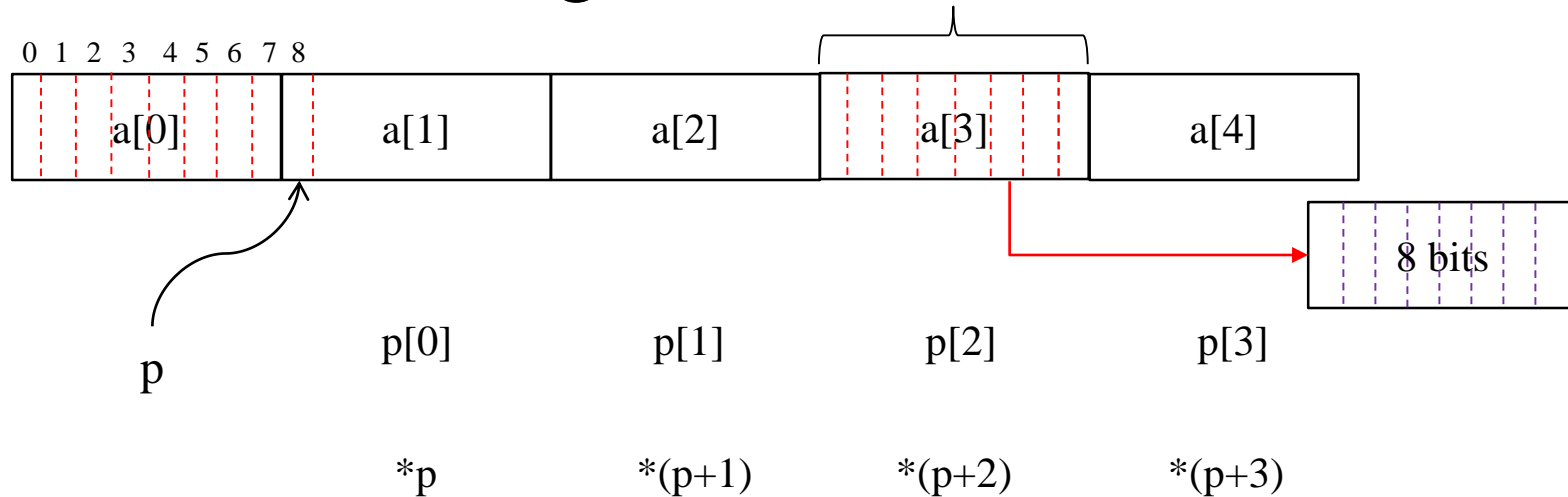


# PHÉP TOÁN TRÊN CON TRỎ

- Có thể dùng phép  $+$  và  $-$  trên địa chỉ
  - Phép cộng: địa chỉ  $\pm$  số nguyên sẽ tạo ra địa chỉ bộ nhớ mới có độ lệch so với địa chỉ cũ một số byte phụ thuộc vào kiểu con trỏ và số nguyên
  - Công thức: địa chỉ mới = địa chỉ cũ  $\pm$  số nguyên  $\times$  sizeof(kiểu con trỏ)
  - Phép trừ: con trỏ 1  $-$  con trỏ 2 trả ra độ lệch có đơn vị tùy thuộc vào kiểu con trỏ
  - Công thức: con trỏ 1  $-$  con trỏ 2 =  $\frac{address1 - address2}{sizeof(type)}$

# PHÉP TOÁN TRÊN CON TRỎ

- Xét ví dụ mảng **double** 8 bytes



- $p + 1 = \&a[2]$
- $(\text{float}^*)p + 2 = \&a[2]$ ,  $(\text{float}^*)p + 4 = \&a[3]$
- $*((\text{float}^*)p + 4) = \frac{1}{2}$  bytes đầu của  $a[3]$

# PHÉP TOÁN TRÊN CON TRỎ

- Xét đoạn mã

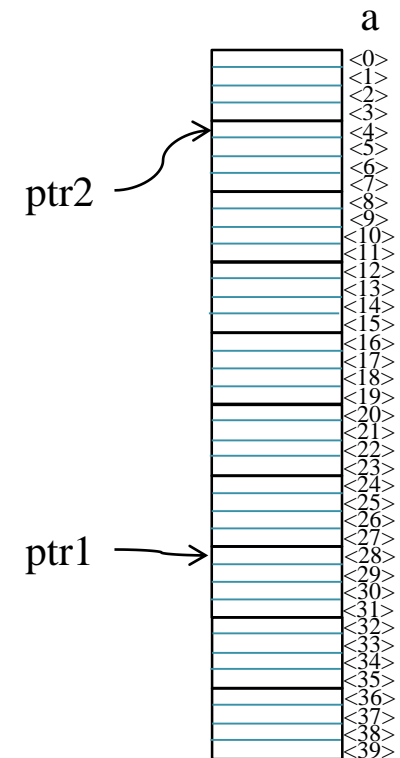
Dòng	Mô tả
1	<code>void main () {</code>
2	<code>long a[10];</code>
3	<code>void* ptr1 = &amp;a[7], *ptr2 = &amp;a[1];</code>
4	<code>long d1 = (long*)ptr1 - (long*)ptr2;</code>
5	<code>long d2 = (char*)ptr1 - (char*)ptr2;</code>
6	<code>long d3 = (short*)ptr2 - (short*)ptr1;</code>
7	<code>cout&lt;&lt;"d1 = " &lt;&lt; d1 &lt;&lt; endl;</code>
8	<code>cout&lt;&lt;"d2 = " &lt;&lt; d2 &lt;&lt; endl;</code>
9	<code>cout&lt;&lt;"d3 = " &lt;&lt; d3 &lt;&lt; endl;</code>
10	<code>cout&lt;&lt;"Distance = " &lt;&lt; (long)ptr1 - (long)ptr2 &lt;&lt; endl;</code>
11	<code>}</code>

$(\langle 28 \rangle - \langle 4 \rangle) / 4$

$(\langle 28 \rangle - \langle 4 \rangle) / 1$

$(\langle 4 \rangle - \langle 28 \rangle) / 2$

$28 - 4$



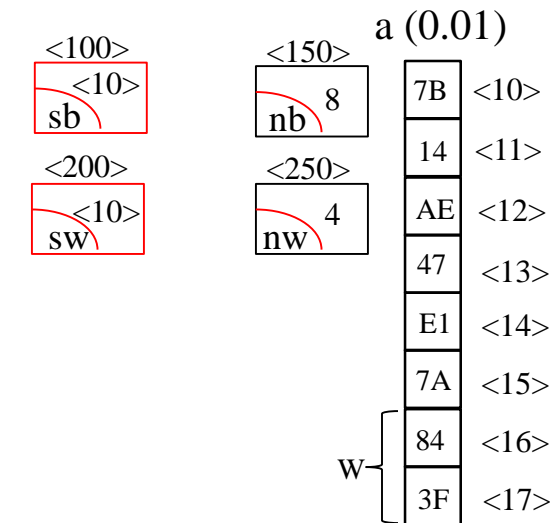
# CÁC KỸ THUẬT ỨNG DỤNG CON TRỎ

- Lấy ra dữ liệu lưu trữ vật lý
  - Lấy ra dữ liệu là các byte/word trong một biến
  - Có thể dùng kiểu `char*`/`short*` để truy xuất tới từng byte/word trong một kiểu dữ liệu cơ sở
  - Dùng cú pháp
    - `(char*)x` trong C hoặc `(char*)&x` trong C++
    - `(short*)x` trong C hoặc `(short*)&x` trong C++
  - Dùng “%X” trong C hay `hex` trong C++ để in ra số dạng thập lục
  - Kỹ thuật tương tự như với các byte động

# CÁC KỸ THUẬT ỨNG DỤNG CON TRỎ

- Lấy ra dữ liệu lưu trữ vật lý

Dòng	Mô tả
1	<code>char* getBytes(double* x, int* n){ *n = 8; return (char*)x; }</code>
2	<code>short* getWords(double* x, int* n){ *n = 4; return (short*)x; }</code>
3	
4	<code>void listBytes(char b[], int nb){</code>
5	<code>for(int i = 0; i &lt; nb; i++) cout &lt;&lt; hex &lt;&lt; (unsigned char)b[i];</code>
6	<code>cout &lt;&lt; endl;</code>
7	<code>}</code>
8	
9	<code>void listWords(short w[], int nw){</code>
10	<code>for(int i = 0; i &lt; nw; i++) cout &lt;&lt; hex &lt;&lt; (unsigned short)w[i];</code>
11	<code>cout &lt;&lt; endl;</code>
12	<code>}</code>



7B 14 AE 47 E1...  
`sb = getBytes(&a, &nb)`



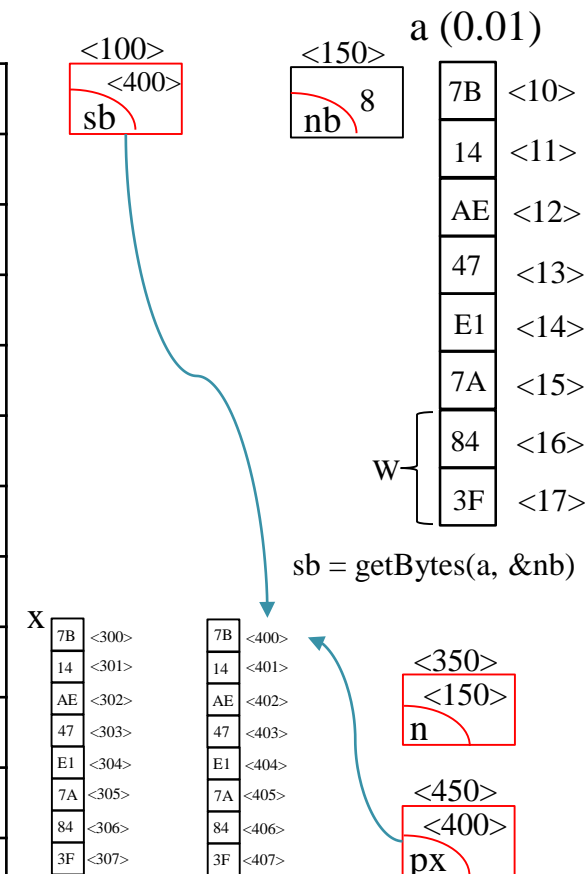
147B 47AE ...  
`sw = getWords(&a, &nw)`



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Lấy ra dữ liệu lưu trữ vật lý (hiệu chỉnh hàm `getBytes`)

Dòng	Mô tả
1	<code>char* getBytes(double x, int* n){</code>
2	<code>double* px = (double*)malloc(sizeof(double));</code>
3	<code>*n = 8;</code>
4	<code>if(px != NULL) *px = x;</code>
5	<code>return (char*)px;</code>
6	<code>}</code>
9	<code>void main(){</code>
10	<code>double a = 0.01; char* sb; int nb;</code>
11	<code>sb = getBytes(a, &amp;nb);</code>
12	<code>if(sb != NULL){ //...; free(sb);}</code>
	<code>}</code>



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Lấy trực tiếp mảng con
  - Có thể dùng một phần của mảng
  - Ví dụ: `int a[15] = { ... }, *sub_a = &a[5]`
  - Dễ thấy `sub_a[0] = a[5], ..., sub_a[9] = a[14]`
- Xét ví dụ hàm sắp xếp đệ quy

```
void main(){  
    float B[] = {-9, 12, 2.3, 11, -10, -3.6}  
    int nB = sizeof(B)/sizeof(B[0]);  
    minmaxSort(B, nB);  
    for(int i = 0; i < nB; i++){ cout<<B[i]}  
}
```

# CÁC KỸ THUẬT ỨNG DỤNG CON TRỎ

- Xét ví dụ sắp xếp đệ quy (ý tưởng)

max			min						max			min					
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
-9	12.7	2.3	11	-10	-3.6	-10	-3.6	2.3	11	-9	12.7	-10	-3.6	2.3	11	-9	12.7
max			min						max			min					
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
-10	12.7	2.3	11	-9	-3.6	-10	-9	2.3	11	-3.6	12.7	-10	-9	2.3	11	-3.6	12.7
max			min						max			min					
0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
-10	-3.6	2.3	11	-9	12.7	-10	-9	2.3	-3.6	11	12.7	-10	-9	2.3	-3.6	11	12.7



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Xét ví dụ hàm sắp xếp đệ quy (ý tưởng)

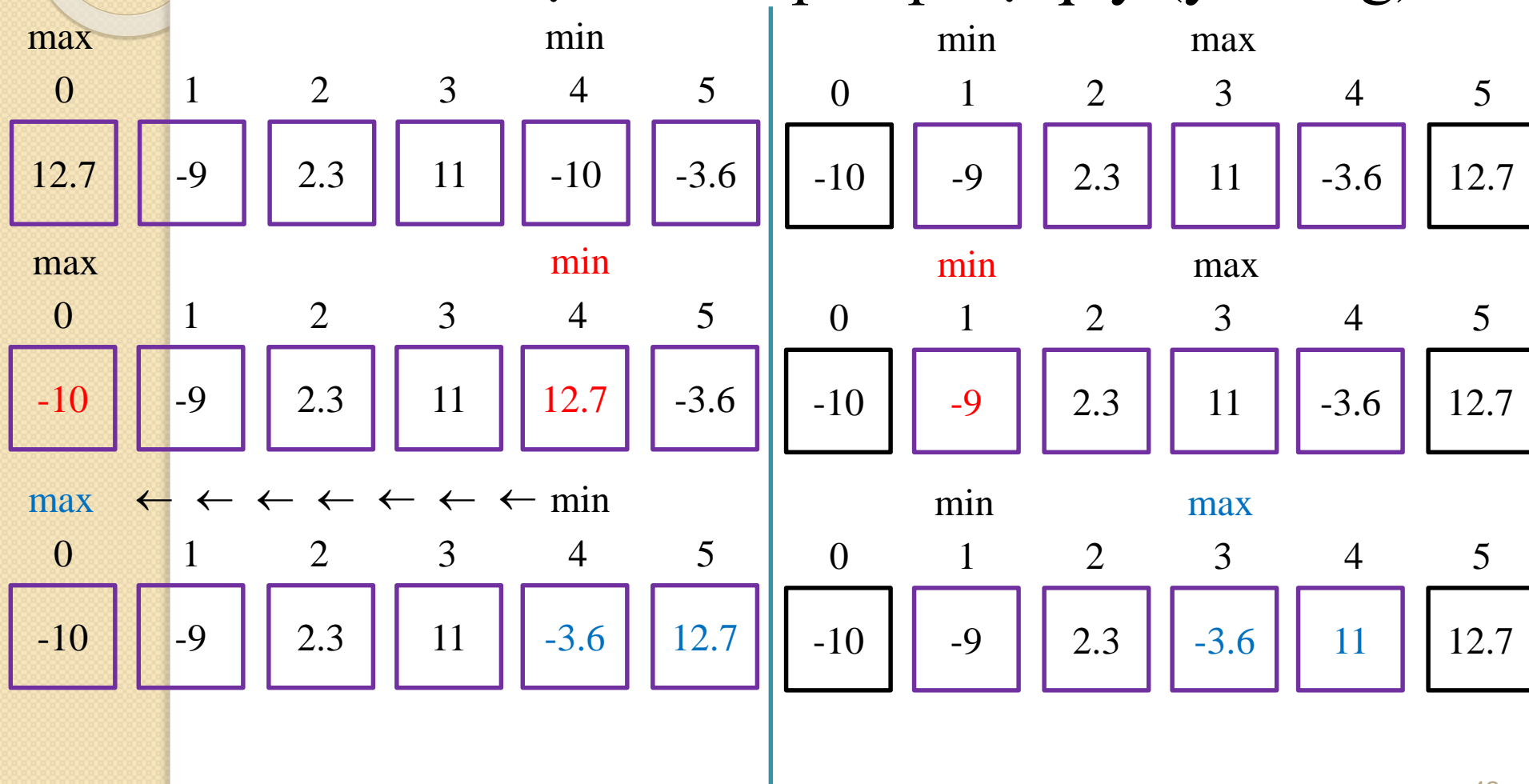
	0	1	max	min	4	5
	0	1	2	3	4	5
	-10	-9	2.3	-3.6	11	12.7

	0	1	max	min	4	5
	0	1	2	3	4	5
	-10	-9	-3.6	2.3	11	12.7

	0	1	max	← min	4	5
	0	1	2	3	4	5
	-10	-9	-3.6	2.3	11	12.7

# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Xét ví dụ khác sắp xếp đệ quy (ý tưởng)



# CÁC KỸ THUẬT ỨNG DỤNG CON TRỎ

- Xét ví dụ hàm sắp xếp đệ quy (ý tưởng)

	0	1	max	min	4	5
	0	1	2	3	4	5
	-10	-9	2.3	-3.6	11	12.7

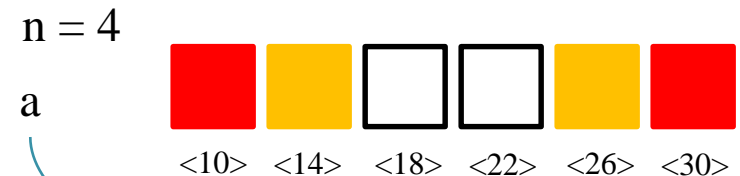
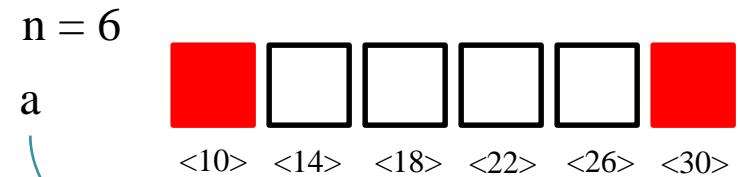
	0	1	max	min	4	5
	0	1	2	3	4	5
	-10	-9	-3.6	2.3	11	12.7

	0	1	max	← min	4	5
	0	1	2	3	4	5
	-10	-9	-3.6	2.3	11	12.7

# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Lấy trực tiếp mảng con

Dòng	Mô tả
1	<code>void minMaxSort(float a[], int n){</code>
2	<code>int idmin = 0, idmax = 0;</code>
3	<code>for(int i = 1; i &lt; n; i++){</code>
4	<code>if(a[idmin] &gt; a[i]) idmin = i;</code>
5	<code>if (a[idmax] &lt; a[i]) idmax = i;</code>
6	<code>}</code>
7	<code>swap(&amp;a[0], &amp;a[idmin]);</code>
8	<code>if(idmax == 0) idmax = idmin;</code>
9	<code>swap(&amp;a[n - 1], &amp;a[idmax]);</code>
10	<code>if(n &gt; 3) minMaxSort(&amp;a[1], n - 2);</code>
11	<code>}</code>



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật khởi tạo dữ liệu với biến cấu trúc

Dòng	
1	<code>#include &lt;stdlib.h&gt;</code>
2	<code>typedef struct {</code>
3	<code>char* Name, *FamilyName</code>
4	<code>long id; char BirthDate[11];</code>
5	<code>float AGP;</code>
6	<code>} StudentRec;</code>
7	<code>void Init(StudentRec* st){</code>
8	<code>st-&gt;Name = st-&gt;FamilyName = NULL;</code>
9	<code>st-&gt;id = st-&gt;AGP = 0;</code>
10	<code>for(int i = 0; i &lt; sizeof(st-&gt;BirthDate); i++)</code>
11	<code>st-&gt;BirthDate[i] = 0;</code>
12	<code>}</code>

Chú ý hàm main

```
void main(){
    StudentRec st1;
    Init(&st1);
    StudentRec* st2;
    Init(st2);
}
```

```
#include <memory.h>
memset(st, 0, sizeof(StudentRec))
```

# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật duyệt trên chuỗi ký tự
- Kiểu dữ liệu con trỏ phải là **char**

Dòng	
1	<b>int</b> strLen( <b>char</b> *s){
2	<b>int</b> len = 0;
3	<b>while</b> (s[len] != '\0') len++;
4	<b>return</b> len;
5	}

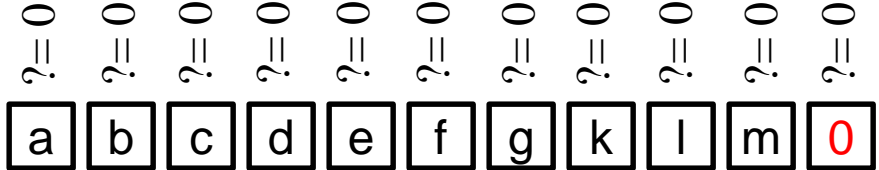
a b c d e f g k l m 0


len ← 10

# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật duyệt trên chuỗi ký tự
- Thuật toán tăng biến ‘len’ trực tiếp trong toán tử ‘[]’

Dòng	
1	<code>int strLen(char *s){</code>
2	<code>int len = 0;</code>
3	<code>while(s[len++] != '\0');</code>
4	<code>return len - 1;</code>
5	<code>}</code>



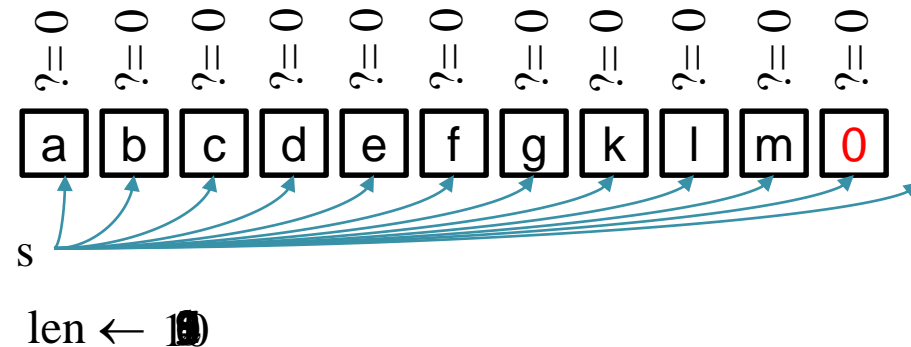
len ← 

- Lưu ý:
  - Dòng 3 lấy ra s[len], sau đó mới tăng biến len
  - Dòng 3 so sánh s[len] ?= ‘\0’ với “**len cũ**”

# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật duyệt trên chuỗi ký tự
- Dùng toán tử \* kết hợp toán tử ++

Dòng	
1	<code>int strLen(char *s){</code>
2	<code>int len = 0;</code>
3	<code>while(*s++ != '\0') len++;</code>
4	<code>return len;</code>
5	<code>}</code>



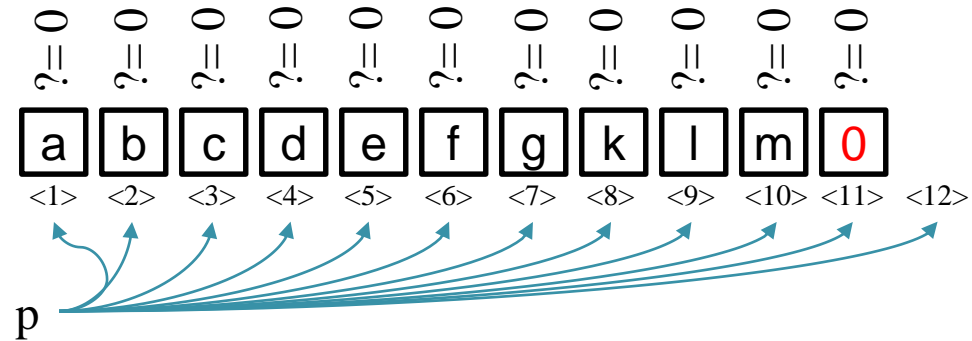
- Lưu ý:
  - Dòng 3 lấy ra \*s, sau đó mới tăng địa chỉ
  - Dòng 3 so sánh \*s != '\0' (\*s cũ)



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật duyệt trên chuỗi ký tự
- Tận dụng địa chỉ để tăng tốc tính toán

Dòng	Mô tả
1	<code>int strLen(char* s){</code>
2	<code>char* p = s;</code>
3	<code>while(*p++);</code>
4	<code>return (p - s - 1);</code>
5	<code>}</code>

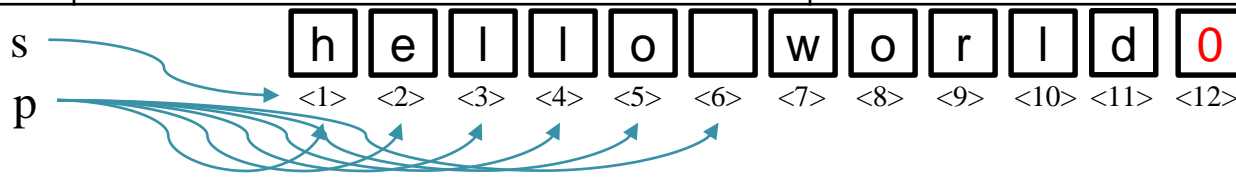


$$p - s - 1 = (<12> - <1>) / \text{sizeof}(\text{char}) - 1 \times \text{sizeof}(\text{char}) = 10$$

# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật tìm vị trí ký tự ngắt
- Các ký tự ngắt có thể là: khoảng trắng, dấu phẩy, ký tự tab...

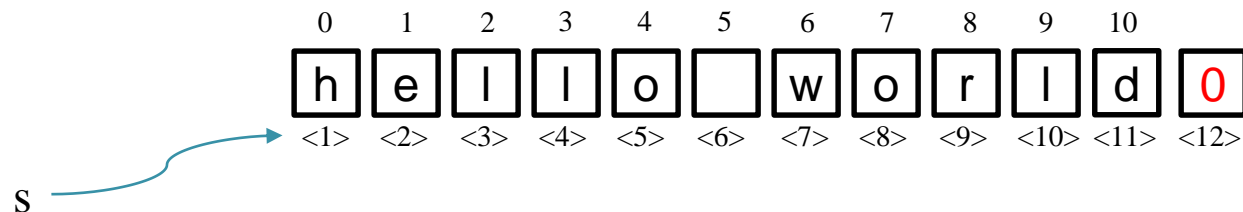
Dòng	
1	<code>int isDelimiter(char s){return (s == ' '    s == ','    s == '\t'    s == '\n');}</code>
2	<code>char* delim(char* s){</code>
3	<code>int i = 0, n = strlen(s);</code>
4	<code>while(i &lt; n &amp;&amp; !isDelimiter(s[i]))</code>
5	<code>{i++;}</code>
6	<code>return s + i;</code>
7	<code>}</code>



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Kỹ thuật tìm vị trí ký tự ngắt
- Cải tiến thuật toán dùng chỉ số

Dòng		
2	<code>char* delim(char* s){</code>	<code>char* delim(char* s){</code>
3	<code>int i = 0;</code>	<code>int i = 0; char c;</code>
4	<code>while(s[i] !=0 &amp;&amp; !isDelimiter(s[i]))</code>	<code>while((c = s[i]) != 0 &amp;&amp; !isDelimiter(c))</code>
5	<code>{i++;}</code>	<code>{i++;}</code>
6	<code>return s + i;</code>	<code>return s + i;</code>
7	<code>}</code>	<code>}</code>



# CÁC KỸ THUẬT ÁP DỤNG CON TRỎ

- Một số nguyên tắc khi dùng con trỏ
  - Khi khai báo con trỏ, giá trị của nó sẽ là rác (địa chỉ của chương trình trước đó)  $\Rightarrow$  nên gán NULL trực tiếp, ví dụ `int* p = NULL;`
  - Nên kiểm tra giá trị biến con trỏ trước khi dùng, ví dụ `if(p != NULL) { ... }`
  - Con trỏ ở vị trí tham số của hàm có nghĩa là nó đã được gán địa chỉ ở nơi gọi nó, ví dụ:
    - `int abc(int* p){ ... }; void main(){ abc(new int); }`
  - Khi dịch chuyển con trỏ nên trong phạm vi vùng nhớ mà nó đang quản lý
  - Khi dời địa chỉ con trỏ bằng phép toán ‘+’ hay ‘-’ thì nhớ phải dựa vào kiểu dữ liệu con trỏ, ví dụ: con trỏ `int` khi ‘+’ một đơn vị sẽ dời 4 byte, con trỏ `short` khi ‘+’ hai đơn vị sẽ dời 4 byte

# BÀI TẬP

- Bài 1: Xây dựng các hàm với nguyên mẫu:
  - Hàm `void InputArray_1D(int*& a, int& n)`: xin cấp phát mảng số nguyên một chiều n phần tử
  - Hàm `void FreeArray_1D(int*& a)`: xin hủy mảng vừa được cấp phát
  - Hàm `void OutputArray_1D(int* a, int n)`: in ra n phần tử từ mảng số nguyên một chiều
  - Hàm `void main()`: minh họa sử dụng hàm trên
- Lưu ý: có thể sử dụng các hàm `malloc`, `calloc`, `free`...hoặc toán tử `new/delete`

# BÀI TẬP

- Bài 2: Viết hàm main minh họa
  - Cách tạo ma trận vuông có kích thước  $n$
  - Cách xuất ma trận vuông có kích thước  $n$
  - Cách hủy ma trận vuông có kích thước  $n$
- Lưu ý: có thể sử dụng toán tử new/delete ngoài các hàm malloc, calloc hay delete