

CON TRỎ

KỸ THUẬT LẬP TRÌNH

GVHD: Trương Toàn Thịnh

NỘI DUNG

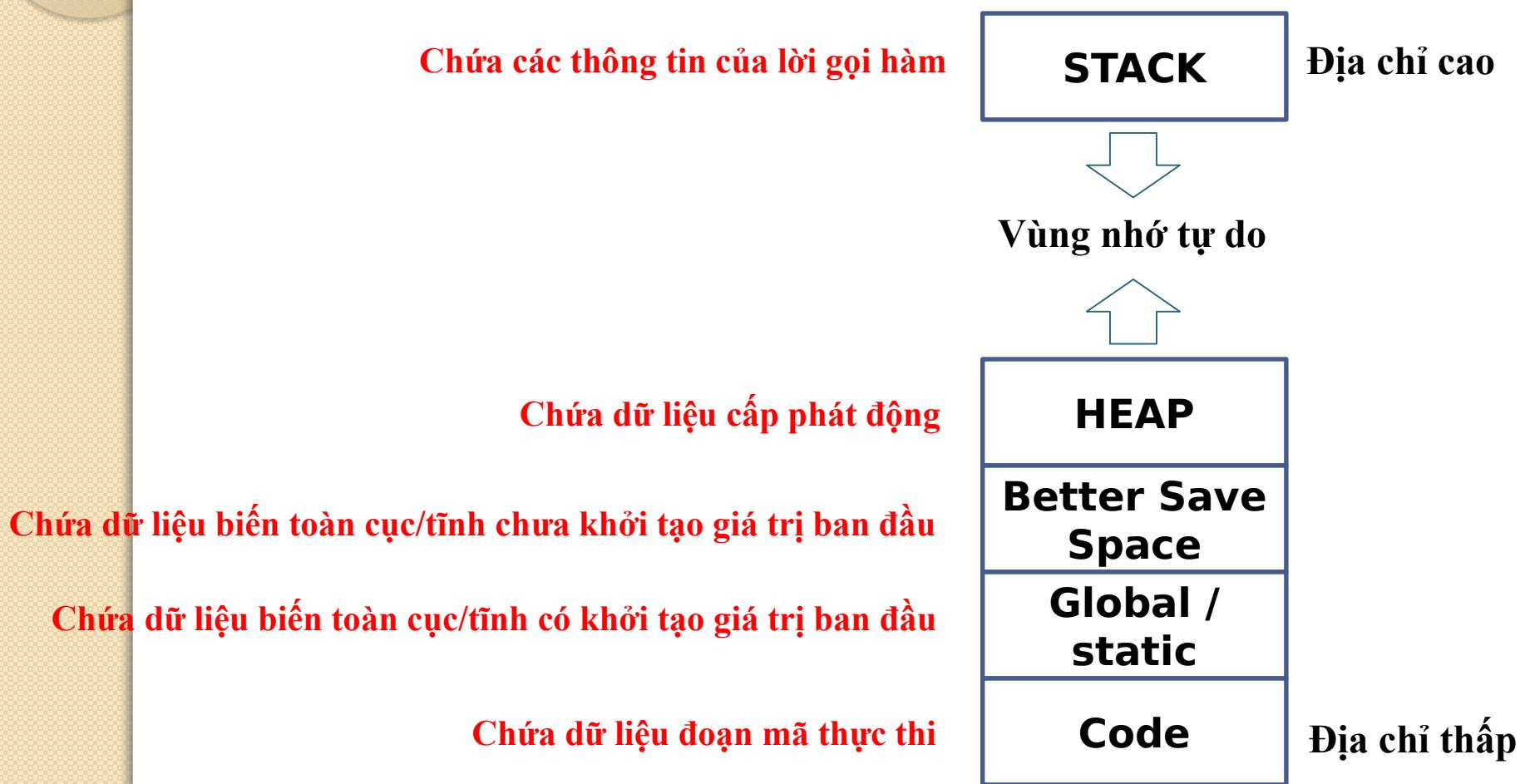
- Vùng nhớ heap & stack
- Các hàm hỗ trợ bộ nhớ
- Các hàm thao tác bộ nhớ
- Cấp phát & và dùng mảng động
- Kỹ thuật kiểm tra con trỏ an toàn
- Bài tập

VÙNG NHỚ HEAP & STACK

- Vùng nhớ stack được cấp riêng cho từng hàm khi được gọi
- Vùng nhớ stack chứa các tham số (đầu vào / đầu ra) và biến cục bộ của hàm
- Khi hàm kết thúc thì vùng nhớ stack được trả về hệ điều hành
- Thuật ngữ ‘stack’ ám chỉ vùng nhớ hoạt động theo quy tắc ‘vào sau – ra trước’
- Vùng nhớ heap mang tính toàn cục (không phụ thuộc vào lời gọi hàm)
- Phải giải phóng vùng nhớ heap tường minh

VÙNG NHỚ HEAP & STACK

- Mô hình bộ nhớ chương trình C/C++



CÁC HÀM HỖ TRỢ BỘ NHỚ

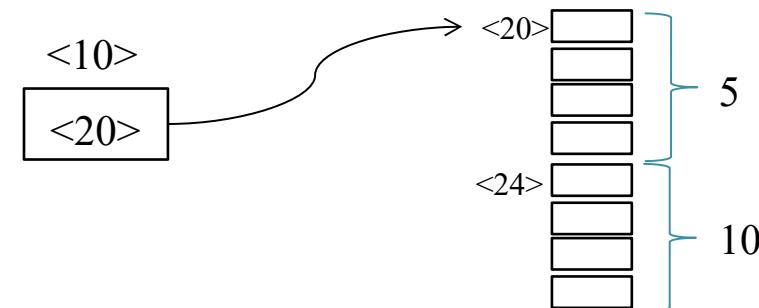
- Các hàm làm việc với vùng nhớ heap
- Cần khai báo “#include <malloc.h>” để sử dụng
- Hằng số NULL (zero) trong chỉ thị “<stdio.h>”
- Hàm **void*** malloc(**int** nSz)
 - Xin cấp phát vùng nhớ có kích thước nSz byte
 - Hàm trả ra địa chỉ vùng nhớ này
- Ví dụ hàm malloc:
 - **int*** a = (**int***)malloc(3*sizeof(**int**));
 - **for(int** i = 0; i < 3; i++){***(a + i)** = i;}
 - **for(int** i = 0; i < 3; i++){cout << ***(a + i)**;"}
 - free(**(int***)a)

CÁC HÀM HỖ TRỢ BỘ NHỚ

- Các hàm làm việc với vùng nhớ heap
- Hàm `void* realloc(void* p, int nSz)`
 - Xin cấp phát lại vùng nhớ cũ với kích thước nSz byte mới
 - Hàm **trả ra địa chỉ vùng nhớ cũ hoặc mới** tùy vào việc tối ưu sự cấp phát
- Ví dụ hàm realloc:
 - `int* a = (int*)malloc(3*sizeof(int));`
 - `cout << hex << a << endl;`
 - `for(int i = 0; i < 3; i++){* (a + i) = i;}`
 - `for(int i = 0; i < 3; i++){cout << *(a + i) << endl;}`
 - `realloc(a, 4xsizeof(int));`
 - `cout << hex << a << endl;`
 - `for(int i = 0; i < 4; i++){cout << *(a + i) << endl;}`
 - `free((int*)a)`

CÁC HÀM HỖ TRỢ BỘ NHỚ

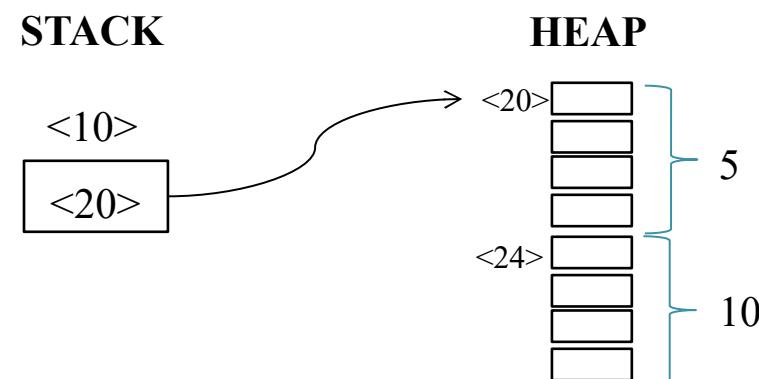
- Các hàm làm việc với vùng nhớ stack => tự hủy k cần free
- Có thể xin cấp phát trong vùng nhớ stack
- Không được giải phóng vùng nhớ khi sử dụng xong
- Các dữ liệu trong vùng nhớ này sẽ tự hủy khi kết thúc việc gọi hàm
- Hàm `void* _alloca(int nSz)`: tương tự như hàm malloc, nhưng vùng nhớ cấp phát trong stack STACK
 - Ví dụ:
 - `int* a = (int*)_alloca(8);`
 - `*a = 5;`
 - `*(a+1) = 10;`
 - `cout << *a << endl;`
 - `cout << *(a+1) << endl;`



CÁC HÀM HỖ TRỢ BỘ NHỚ

- Các hàm làm việc với vùng nhớ stack
- Hàm `void* _expand(void* p, int nSz)`: tương tự như hàm `realloc`, nhưng hàm này chỉ được hỗ trợ trong visual C++
- Khi có nhu cầu muốn biết số lượng byte đã được cấp phát bao nhiêu, ta có thể dùng các hàm sau
 - `int _msize(void* p)`: trả ra số lượng byte do p quản lý, dùng trong Windows
 - `int malloc_size(void* p)`: tương tự nhưng dùng trong Mac OS
 - `int malloc_usable_size(void* p)`: tương tự nhưng trong Linux
- Lưu ý: `_msize` chỉ trả ra số lượng byte được cấp phát bằng các hàm `malloc`, `realloc` hay `calloc`
- Ví dụ hàm `int _msize(void*)`:
 - `int* a = (int*)malloc(8);`
 - `*a = 5;`
 - `*(a+1) = 10;`
 - `cout << *a << " " << *(a+1)<<endl;`
 - `cout << _msize(a) << endl;`

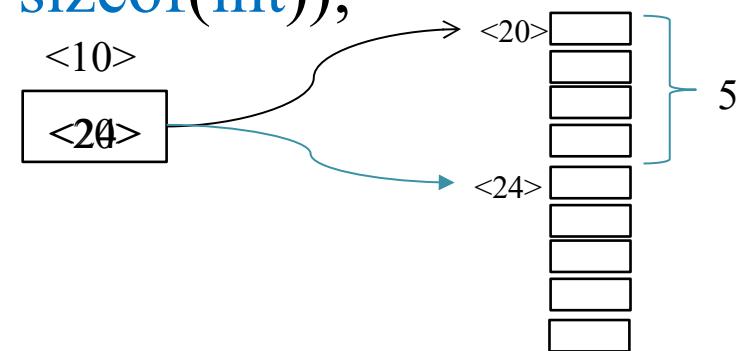
8 bytes



CÁC HÀM HỖ TRỢ BỘ NHỚ

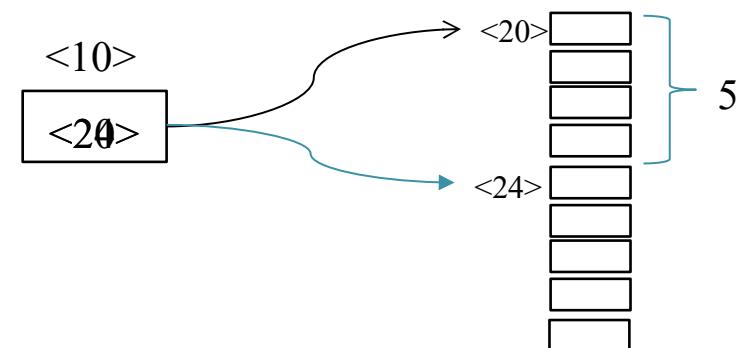
- Các hàm làm việc với vùng nhớ heap
- Xây dựng ba hàm hỗ trợ việc
 - Cấp phát bộ nhớ: tận dụng lại malloc
 - Giải phóng bộ nhớ: tận dụng lại free
 - Trả ra số lượng byte đã cấp phát
- `void* mAlloc(int size){ //ví dụ size là 5 byte`
 - `void* p = malloc(size + sizeof(int));`
 - `*(int*)p = size;`
 - `return ((int*)p) + 1;`
- `}`

ép kiểu void*



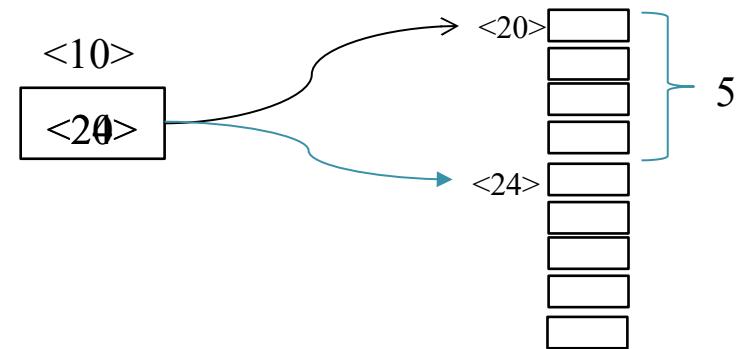
CÁC HÀM HỖ TRỢ BỘ NHỚ

- Các hàm làm việc với vùng nhớ heap
- `void* mAlloc(int size){ //ví dụ size là 5 byte`
 - `void* p = malloc(size + sizeof(int));`
 - `*(int*)p = size;`
 - `return ((int*)p) + 1;`
- `}` <24>
- `void mFree(void* p){` lùi lại <20>
 - `p = ((int*)p) - 1;`
 - `free(p);`
- `}`



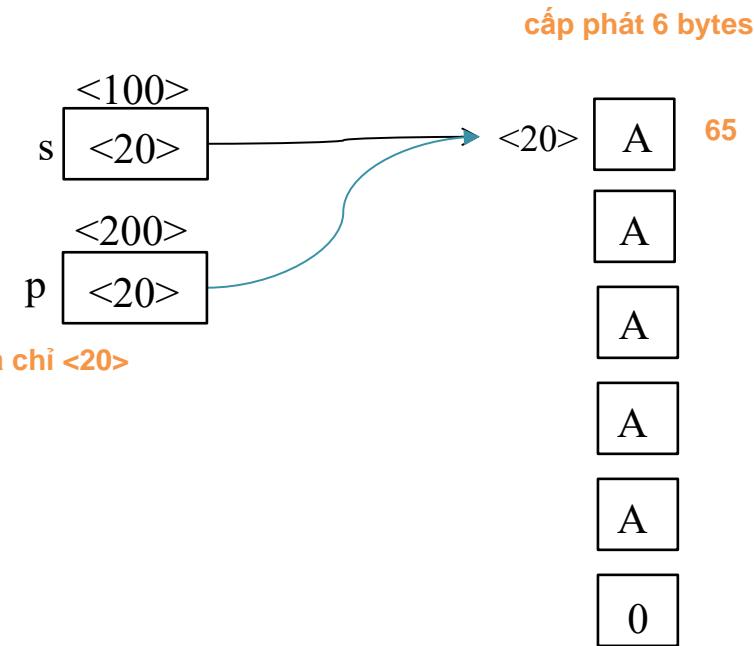
CÁC HÀM HỖ TRỢ BỘ NHỚ

- Các hàm làm việc với vùng nhớ heap
- `void* mAlloc(int size){ //ví dụ size là 5 byte`
 - `void* p = malloc(size + sizeof(int));`
 - `*(int*)p = size;`
 - `return ((int*)p) + 1;`
- }
- `int mSize(void* p){`
 - `return *(((int*)p) - 1); <20>, * lấy số 5`
- }
- `void main(){`
 - `char* p = (char*)mAlloc(5);`
 - `gets(p);`
 - `cout<<p<<endl;`
 - `cout<<mSize(p)<<endl;`
 - `mFree(p);`
- }



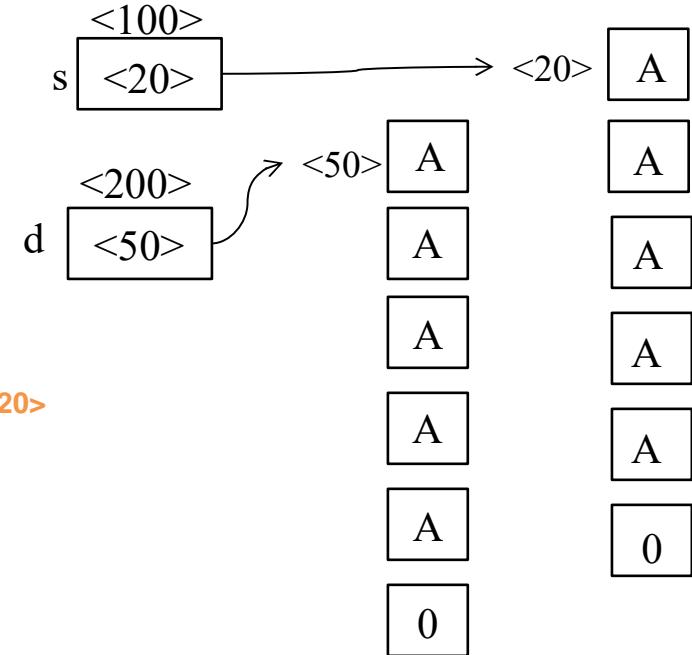
CÁC HÀM THAO TÁC BỘ NHỚ

- Thư viện <memory.h> chứa các hàm thao tác trên bộ nhớ
- void*** memset(**void*** des, **int** ch, **int** n): Khởi tạo n bytes bằng giá trị ch cho vùng nhớ des
- void** main(){
 - char*** s = (**char***)malloc(6);
 - s[5] = 0;
 - char*** p = **memset**(s, 65, 5);
dereference => in ra A thay vì <20>
 - cout << s << endl;
=> 5 chữ A
 - cout << (**int***)s << endl;
=> in ra địa chỉ <20>
 - cout << (**int***)p << endl;
 - free(s);
- }



CÁC HÀM THAO TÁC BỘ NHỚ

- Thư viện <memory.h> chứa các hàm thao tác trên bộ nhớ
- **void*** memmove(**void*** des, **const void*** src, **int** n): Sao chép n bytes từ vùng nhớ src vào vùng nhớ des. Hàm trả ra địa chỉ des đang trả tới
- **void** main(){
 - **char*** s = (**char***)malloc(6);
 - s[5] = 0;
 - memset(s, 65, 5);
 - **char*** d = (**char***)malloc(6);
 - d = (**char***)memmove(d, s, 6);
 - cout << d << endl; => in ra 5 chữ A
 - cout << (**int***)d << endl; => in ra <20>
 - free(s);
 - free(d);
- }



CÁC HÀM THAO TÁC BỘ NHỚ

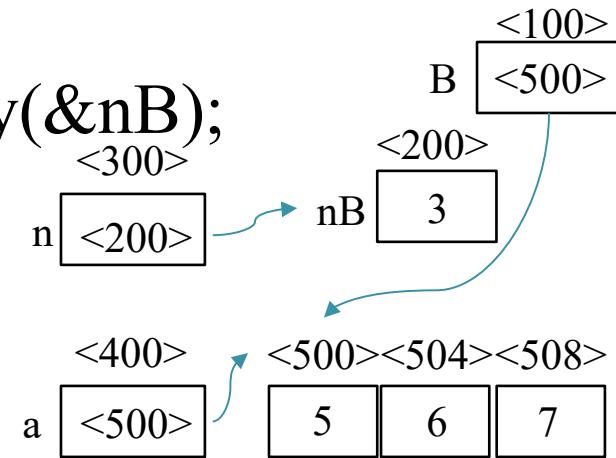
- Thư viện <memory.h> chứa các hàm thao tác trên bộ nhớ
- `int memcmp(const void* buf1, const void* buf2, int n)`: So sánh n bytes của hai vùng nhớ buf1 và buf2. Hàm trả ra 1 nếu `buf1 > buf2`, -1 nếu `buf1 < buf2` và 0 nếu bằng nhau. Lưu ý: hàm chỉ **sánh giá trị từng byte**
- `void main(){`
 - `char* s = (char*)malloc(6);`
 - `s[5] = 0;`
 - `memset(s, 65, 5);`
 - `char* d = (char*)malloc(6);`
 - `d = (char*)memmove(d, s, 6);`
 - `cout << memcmp(s, d, 6) << endl;`
 - `free(s);`
 - `free(d);`
- `}`

CÁC HÀM THAO TÁC BỘ NHỚ

- Thư viện <memory.h> chứa các hàm thao tác trên bộ nhớ
- `int _memicmp(const void* buf1, const void* buf2, usigned int n):`
Tương tự như memcmp nhưng không phân biệt ký tự hoa/thường
- `void main(){`
 - `char* s = (char*)malloc(6);`
 - `s[5] = 0;`
 - `memset(s, 65, 5);`
 - `char* d = (char*)malloc(6);`
 - `d = (char*)memmove(d, s, 6);`
 - `d[0] = 'a';`
 - `cout << memcmp(s, d, 6) << endl;`
 - `cout << _memicmp(d, s, 6) << endl;` => k so sánh phân số
 - `free(s);`
 - `free(d);`
- `}`

CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHO TRƯỚC

- Tách thành các hàm nhập/xuất riêng
- **void main()** <500>
 - int nB; float* B = inputArray(&nB);
 - if(B != NULL)
 - outputArray(B, nB); free(B);
- **void inputArray(int* n)**
 - cin>>(*n);
 - float* a = (float*)calloc((*n), sizeof(float));
 - if(a != NULL)
 - for(int i = 0; i < (*n); i++) {cin>>a[i];}
 - return a; <500>



CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHO TRƯỚC

- Xây dựng struct intArray

- Ví dụ

- **struct** intArray{ **int** n, ***arr**; };

- **void** main(){

- intArray B;

- nhap(&B);

- xuat(&B);

- huy(&B);

- }

- **void** nhap(intArray* a){

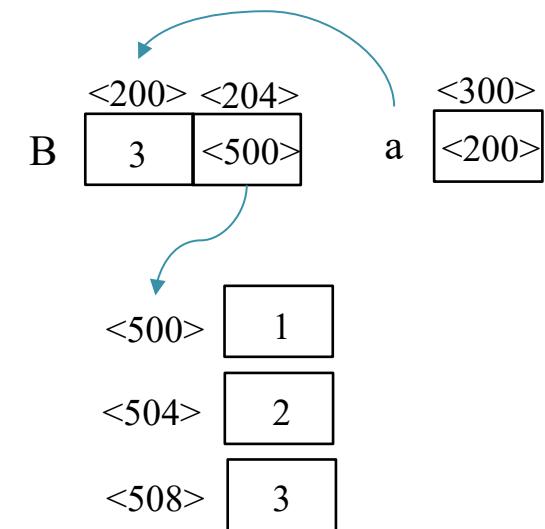
(truy xuất từng field của struct)

- cin >> a->n;

- a->**arr** = (**int***)malloc(**a->n*** sizeof(**int**));

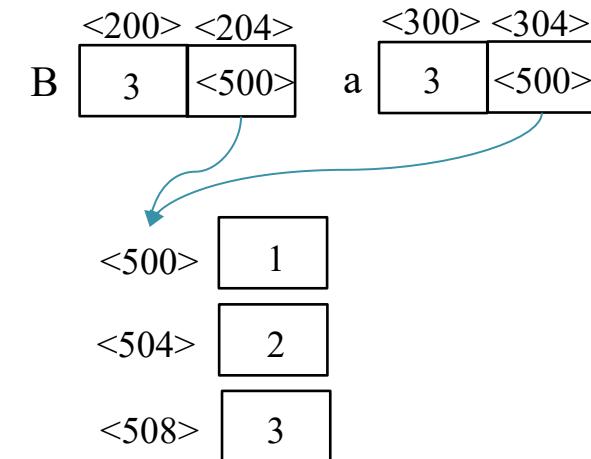
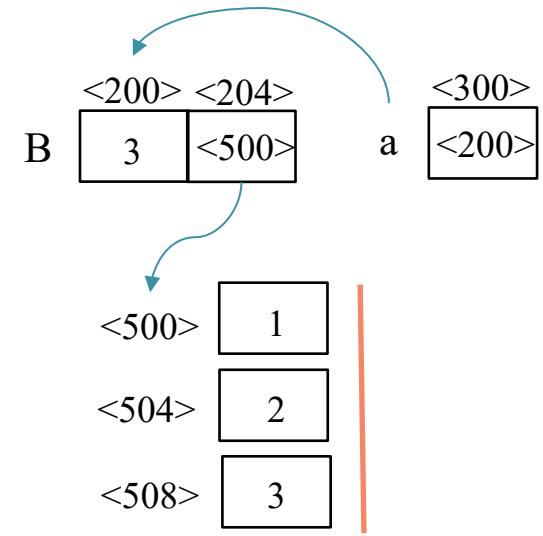
- **for**(**int** i = 0; i < a->n; i++) cin >> a->arr[i];

- }



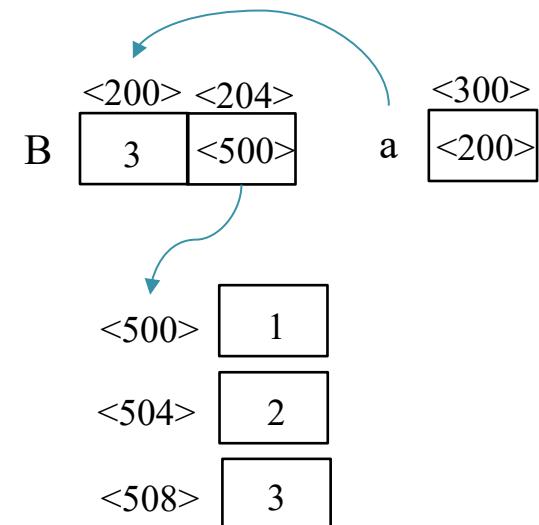
CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHỐNG TRƯỚC

- Xây dựng **struct intArray**
- Ví dụ
 - **struct intArray{ int n, *arr;};**
 - **void main(){**
 - intArray B;
 - nhap(&B);
 - xuat(&B); //xuat(B); 1, 2, 3
 - huy(&B);
 - }
 - **void xuat(const intArray* a){**
 - **for(int i = 0; i < a->n; i++) cout << a->arr[i];**
 - }
 - **void xuat(intArray a){**
 - **for(int i = 0; i < a.n; i++) cout << a.arr[i];**
 - }



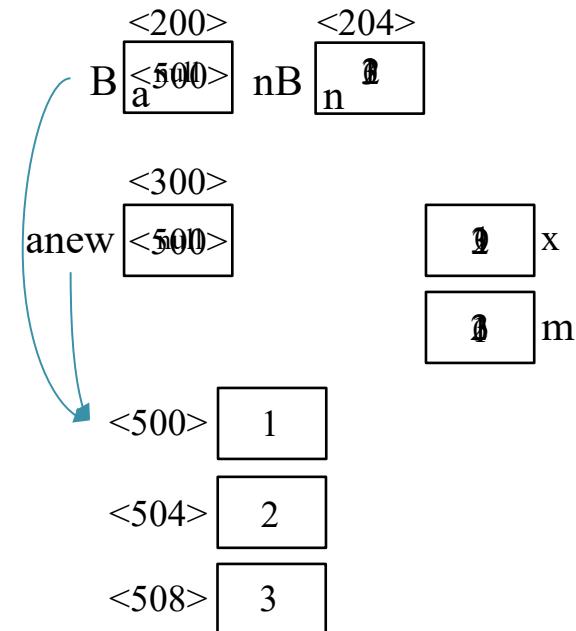
CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHÓI TRƯỚC

- Xây dựng **struct intArray**
- Ví dụ
 - **struct intArray { int n, *arr;};**
 - **void main() {**
 - **intArray B;**
 - **nhap(&B); xuat(&B); huy(&B);**
 - **}**
 - **void huy(intArray* a) {**
 - **if(a != NULL)**
 - **if(a->arr != NULL)**
 - **free(a->arr);**
 - **}**



CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHƯA BIẾT

- Người dùng quyết định khi nào dừng nhập
- Ví dụ: hàm nhập phiên bản **tham chiếu con trỏ**
 - `void main(){`
 - `int* B, nB;`
 - `nhap(B, nB); xuat(B, nB); free(B);`
 - `}`
 - `void nhap(int*& a, int &n){`
 - `int x, m, *anew;`
 - `anew = a = NULL;`
 - `m = x = n = 0;`
 - `while(cin >> x){`
 - `m = n + 1;`
 - `anew = (int*)realloc(a, m * sizeof(int));`
 - `if(anew != NULL){ anew[n++] = x; a = anew; }`
 - `}`
 - `cin.clear();`
 - `}`

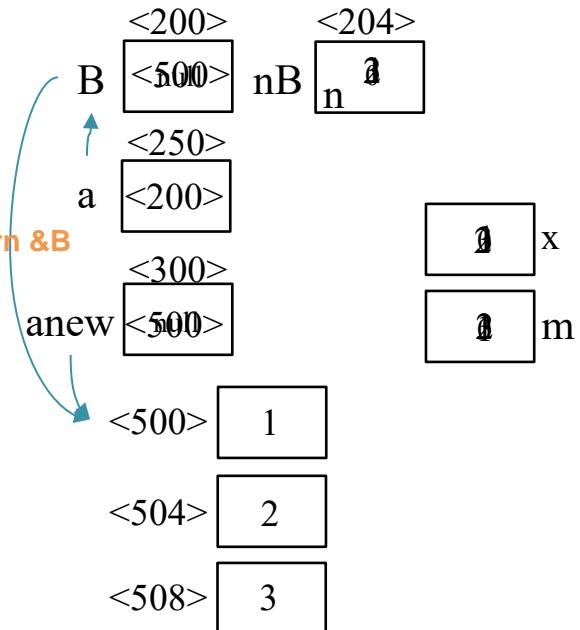


CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHƯA BIẾT

- Người dùng quyết định khi nào dừng nhập
- Ví dụ: hàm nhập phiên bản **tham con trả cấp 2**

```
◦ void main(){  
    • int* B, nB;  
    • nhap(B, nB); xuat(B, nB); free(B);  
◦ }  
◦ void nhap(int** a, int &n){  
    • int x, m, *anew;  
    • anew = *a = NULL;  
    • m = x = n = 0;  
    • while(cin >> x){  
        • m = n + 1;  
        • anew = (int*)realloc(*a, m * sizeof(int));  
        • if(anew != NULL){ anew[n++] = x; *a = anew; }  
        • }  
    • cin.clear();  
◦ }
```

lấy địa chỉ của con trả cấp 1 nên trn & B



CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHƯA BIẾT

- khuôn mẫu hàm cho phép dùng nhiều kiểu dữ liệu
- Ví dụ: dùng hàm nhập phiên bản **tham con trả cấp 2**

- **template <class T>**

cần tổng quát mảng => kiểu T, nhập xuất k cần dùng template

- **void nhap(T** a, int &n){**

- **T** x, *anew; **int** m;

- anew = *a = **NULL**;

- m = n = 0;

- **while**(cin >> x){

- m = n + 1;

- anew = (**T***)realloc(*a, m * **sizeof(T)**);

- **if**(anew != NULL){ anew[n++] = x; *a = anew; }

- }

- cin.clear();

- }

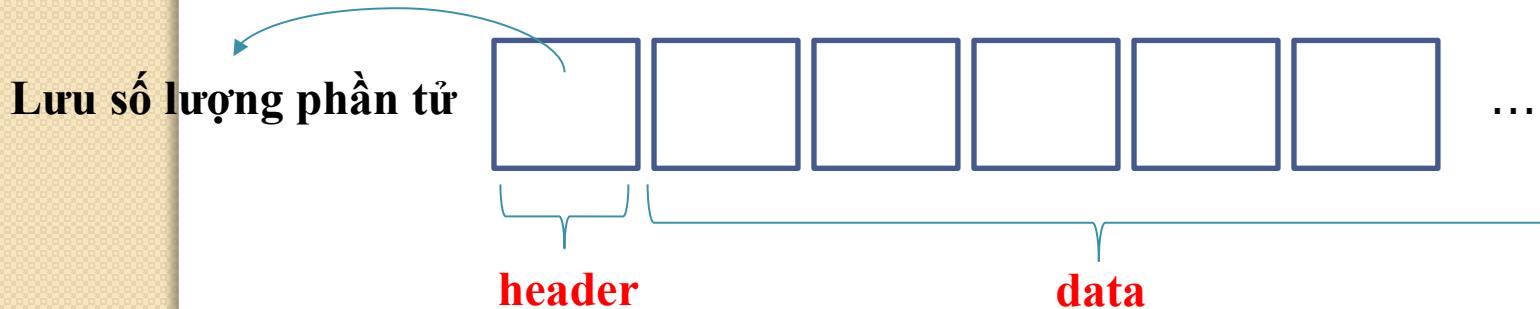
Cần quá tải toán tử ‘>>’ cho kiểu tự định nghĩa

CẤP PHÁT & DÙNG MẢNG ĐỘNG VỚI SỐ PHẦN TỬ CHƯA BIẾT

- Khuôn mẫu hàm cho phép dùng nhiều kiểu dữ liệu
- Ví dụ: dùng hàm nhập phiên bản **tham con trả cấp 2**
 - struct PhanSo{int tu, mau;};
 - void nhap(T** a, int &n){
 - //...
 - }
 - istream& operator>>(istream& inDev, PhanSo& p){
 - inDev >> p.tu >> p.mau;
 - return inDev;
 - }
 - void main(){
 - PhanSo* B; int nB;
 - nhap(B, nB); xuat(B, nB); free(B);
 - }

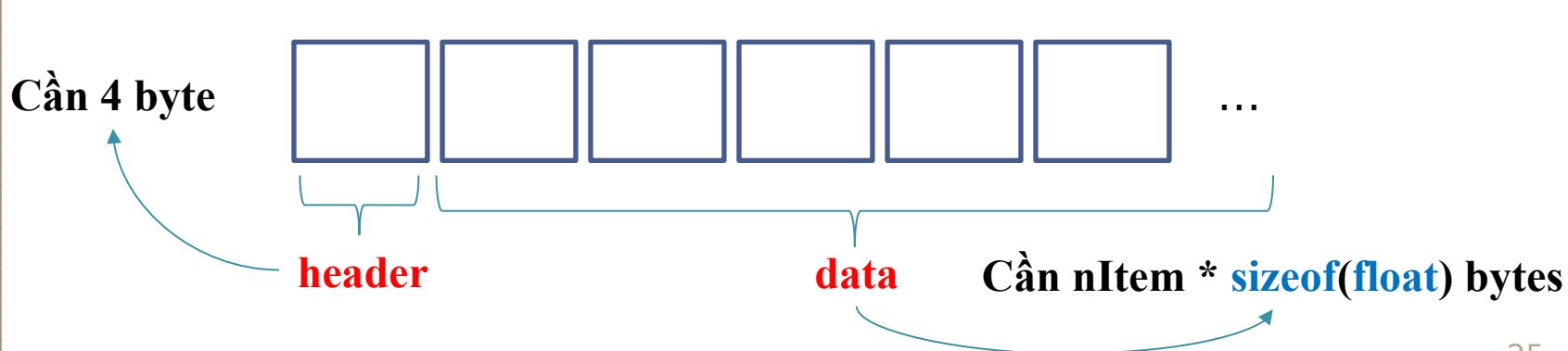
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều
- Các hàm này có ưu điểm:
 - Xử lý cho kiểu tổng quát
 - Không cần dùng kiểu vector<T>
- Ta cần thử nghiệm với mảng kiểu float



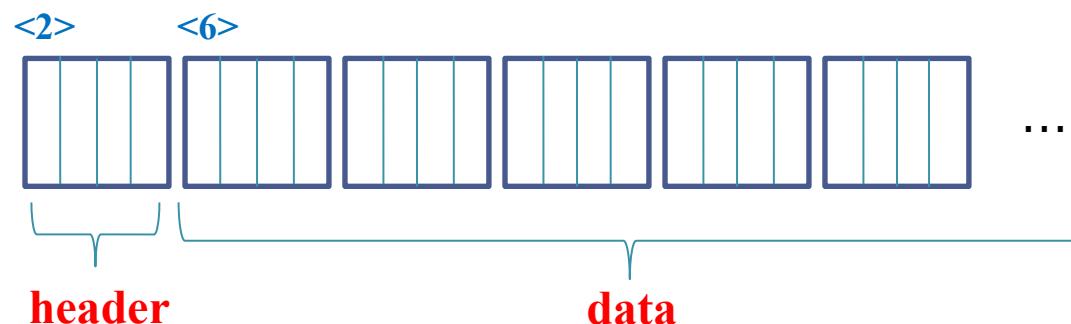
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Ta cần thử nghiệm với mảng kiểu **float**
 - Kích thước header: `int headSize = sizeof(int)`
 - Hàm `int memSize(int nItem)`: dùng để tính ra **kích thước cần dùng** (byte) để lưu `nItem`
 - `int memSize(int nItem){`
 - `return headSize + nItem * sizeof(float);`
 - }



CẤP PHÁT & DÙNG MẢNG ĐỘNG

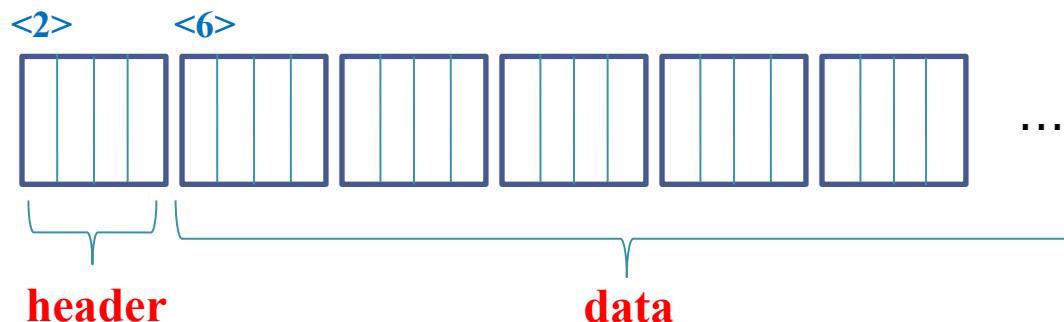
- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm `void* origin_addr(float* aData)`: dùng để lấy ra địa chỉ gốc (địa chỉ vùng head) từ địa vùng data
 - `void* origin_addr(void* aData){`
 - `return (char*)aData - headSize;`
 - `}`



CẤP PHÁT & DÙNG MẢNG ĐỘNG

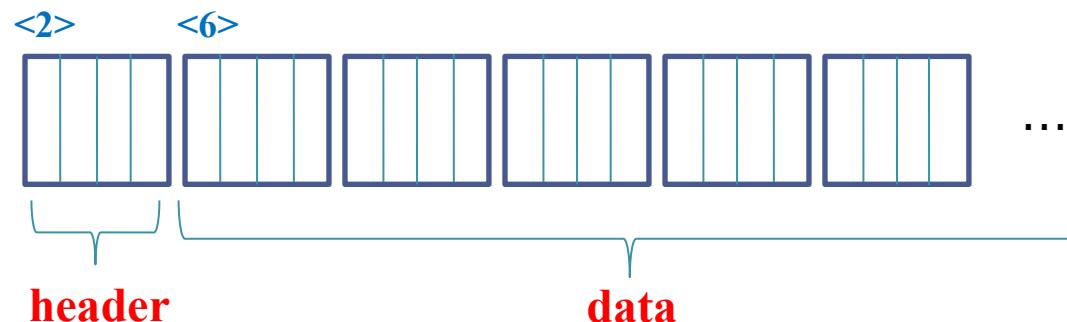
- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **float* data_addr(void* origin)**: dùng để lấy ra địa chỉ vùng data từ địa vùng head (con trỏ origin giữ địa chỉ vùng head)
<2> => return <6>

- **float* data_addr(void* origin){**
 - **return (float*)((char*)origin + headSize);**
 - **}**



CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **void set_nItem(float* aData, int nItem)**:
dùng để gán giá trị nItem vào vùng nhớ head
 - **void set_nItem(float* aData, int nItem){**
 - $*((int*)\text{origin_addr}(aData)) = \text{nItem};$
 - **}** { } $\rightarrow *(<2>)$



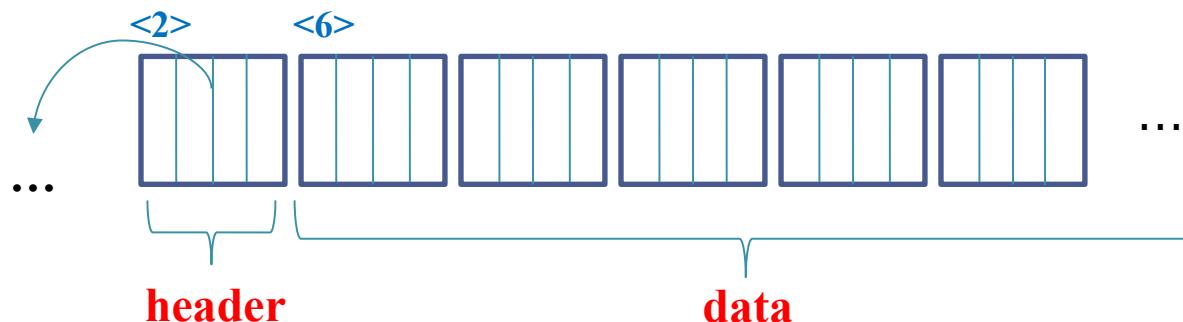
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Ta cần thử nghiệm với mảng kiểu **float**

static o Hàm **int get_nItem(float* aData)**: dùng để lấy giá trị nItem trong vùng nhớ head

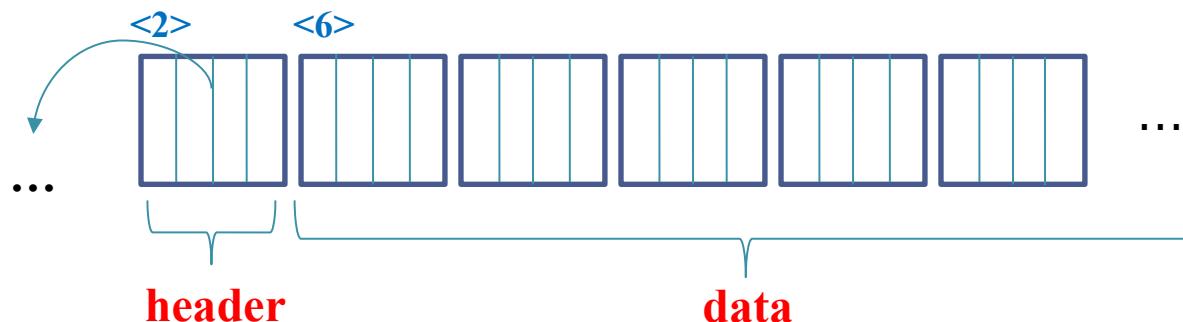
giấu,
hog gọi ra đc

- **int get_nItem(float* aData){**
 - **return *((int*)origin_addr(aData));**
 - **}**



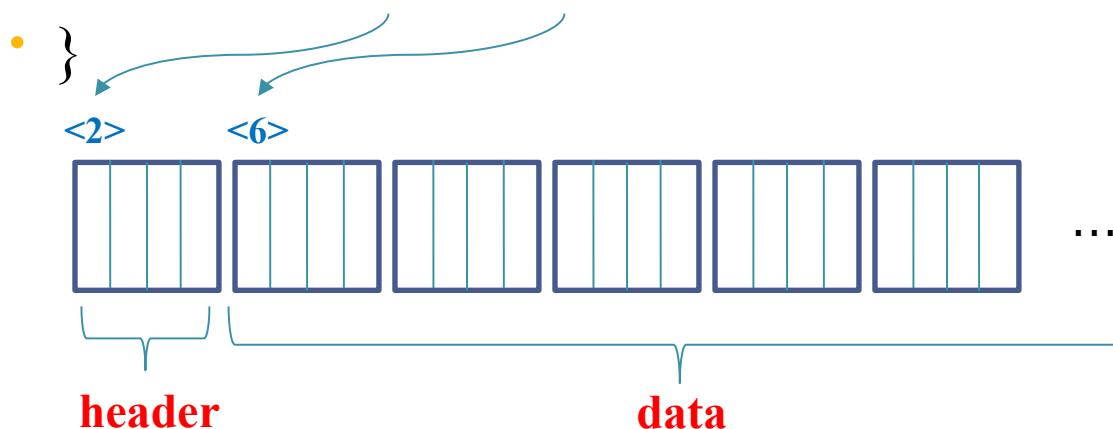
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **int floatArrSize (float* aData)**: dùng để lấy **số phần tử** của mảng aData
 - **int floatArrSize(float* aData){**
 - **if(aData != NULL) return get_nItem(aData);** => hàm giấu k cho developer dùng
 - **return 0;**
 - **}**



CẤP PHÁT & DÙNG MẢNG ĐỘNG

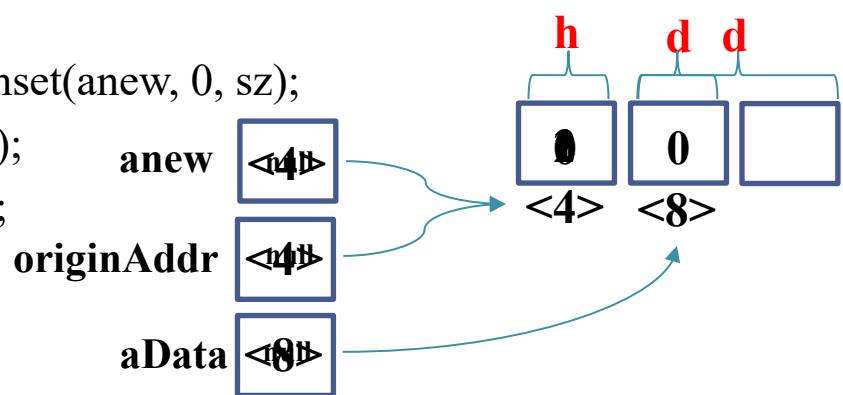
- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **void floatArrFree (float* aData)**: dùng để hủy toàn bộ vùng nhớ đã cấp phát (tính từ địa chỉ vùng head)
 - **void floatArrFree (void* aData){**
 - **if(aData != NULL)**
 ^{lùi về header}
 - **free(origin_addr(aData))**
 - **}**



CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **float*** floatArrResize(**float*** aData, **int** nItem): dùng để **thay đổi kích thước** cũ thành kích thước mới (nItem: số lượng phần tử mới)

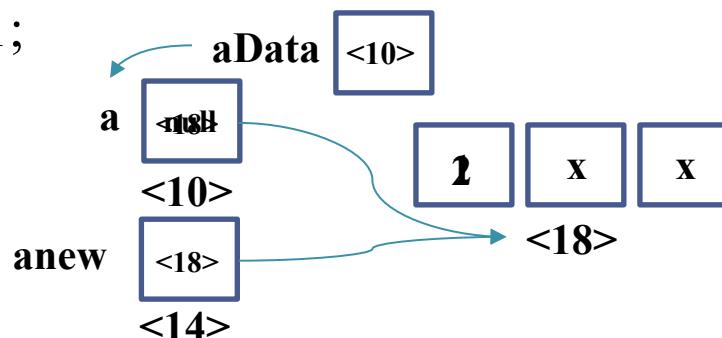
18 ← • **float*** floatArrResize (**float*** aData, **int** nItem){ → 1
• int sz = memSize(nItem);
• **float*** anew = NULL; **void*** originAddr = NULL;
• if(aData != NULL) originAddr = origin_addr(a);
• anew = (**float***)realloc(originAddr, sz);
• if(anew != NULL){
• if(aData == NULL) memset(anew, 0, sz);
• aData = data_addr(anew);
• set_nItem(aData, nItem);
• }
• return aData;
• }



CẤP PHÁT & DÙNG MẢNG ĐỘNG

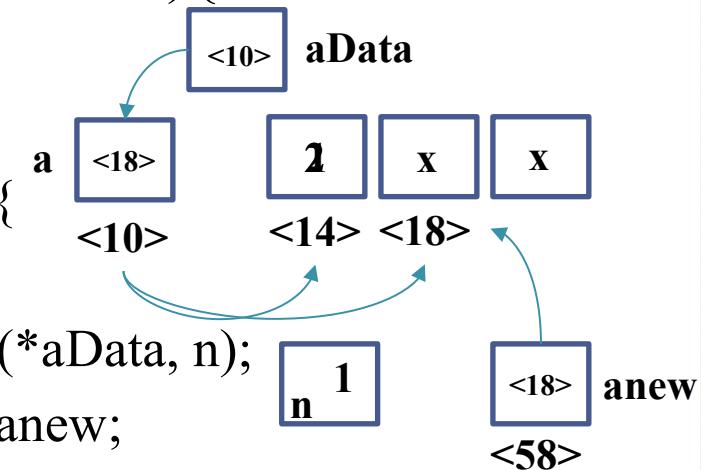
- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **int floatArrPushback(float** aData, float x)**: dùng để **thêm x vào** mảng **một chiều** aData

- **int floatArrPushback(float** aData, float x){**
 - **int n = floatArrSize(*aData);** 
 - **float* anew = floatArrResize(*aData, n + 1);**
 - **if(anew != NULL){**
 - **anew[n] = x;**
 - ***aData = anew;**
 - **return 1;**
 - **}**
 - **return 0;**



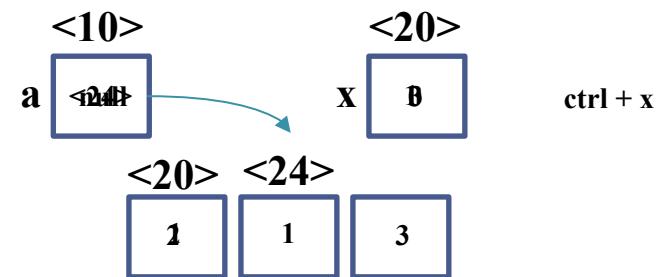
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **float** floatArrPopback(**float**** aData): dùng để **lấy phần tử cuối** ra khỏi mảng aData
 - **int floatArrPopback(**float**** aData){** *{78}*
 - **int** n = floatArrSize(*aData);
 - **float** x = 0;
 - **if(*aData != NULL && n > 0){**
 - **n--;** x = (*aData)[n];
 - **float*** anew = floatArrResize(*aData, n);
 - **if(anew != NULL) *aData = anew;**
 - **}**
 - **return** x;
 - **}**



CẤP PHÁT & DÙNG MẢNG ĐỘNG

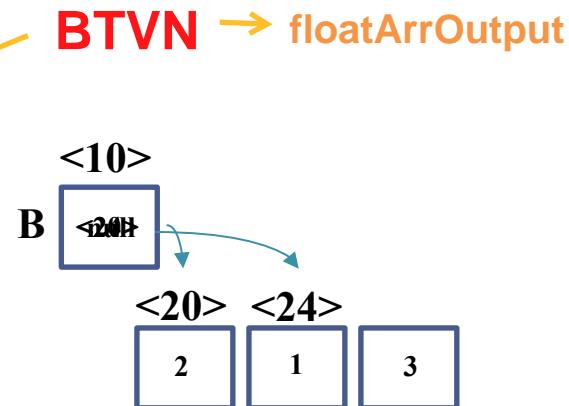
- Ta cần thử nghiệm với mảng kiểu **float**
 - Hàm **float*** floatArrInput(): dùng để trả ra mảng **một chiều các số thực**
 - **float*** floatArrInput(){
 - **float*** a = NULL, x = 0;
 - **while**(cin >> x) {
 - floatArrPushback(&a, x);
 - }
 - cin.clear();
 - **return** a;
 - }



CẤP PHÁT & DÙNG MẢNG ĐỘNG

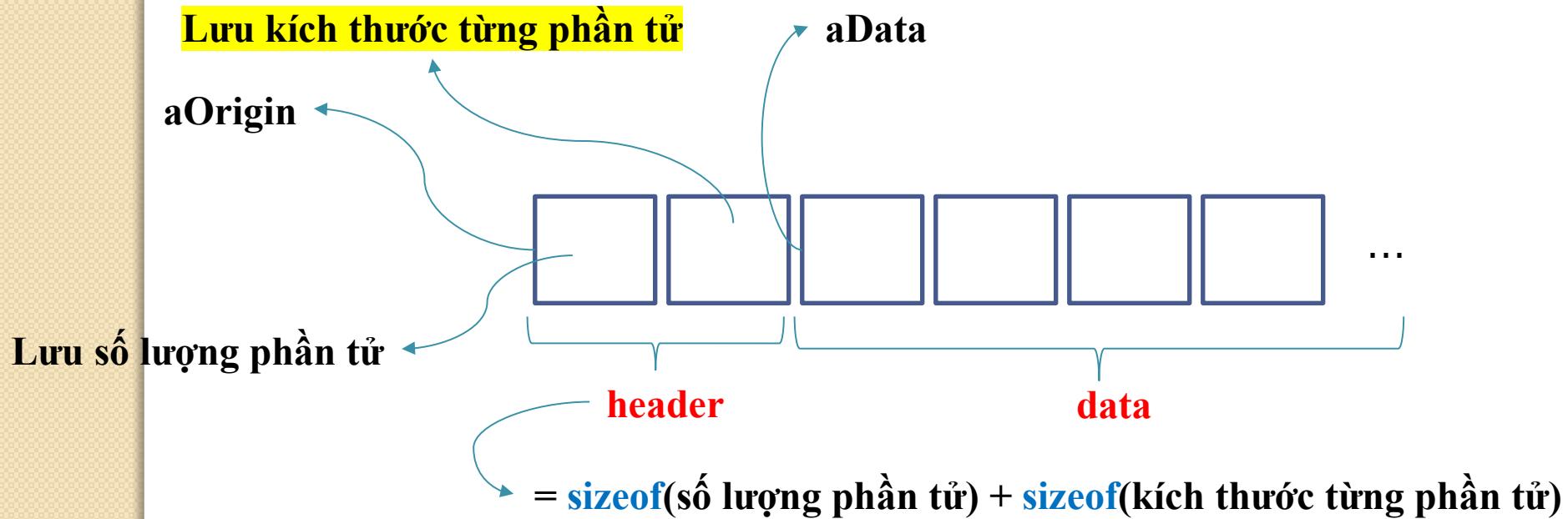
- Ta cần thử nghiệm với mảng kiểu **float**
 - Minh họa hàm main

```
void main(){  
    • float* B = NULL;  
    • B = floatArrInput();  
    • floatArrFree(B);  
    • }
```



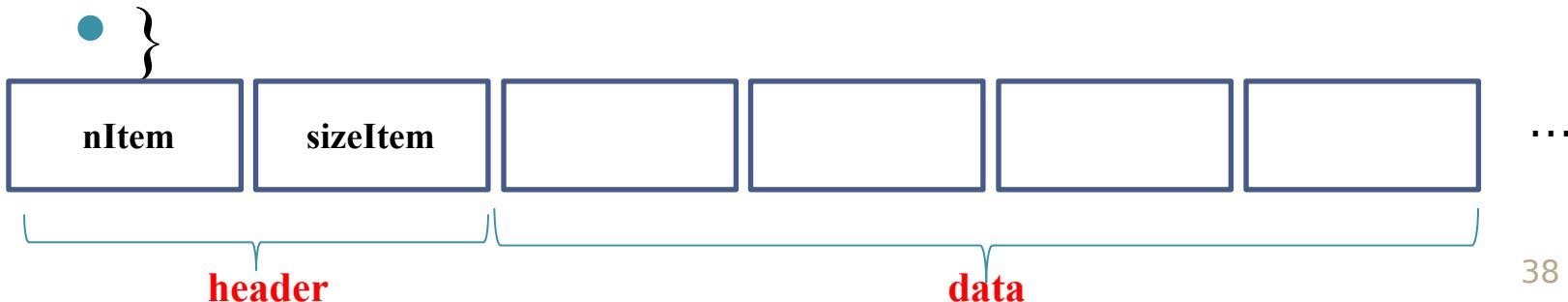
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu **TỔNG QUÁT**



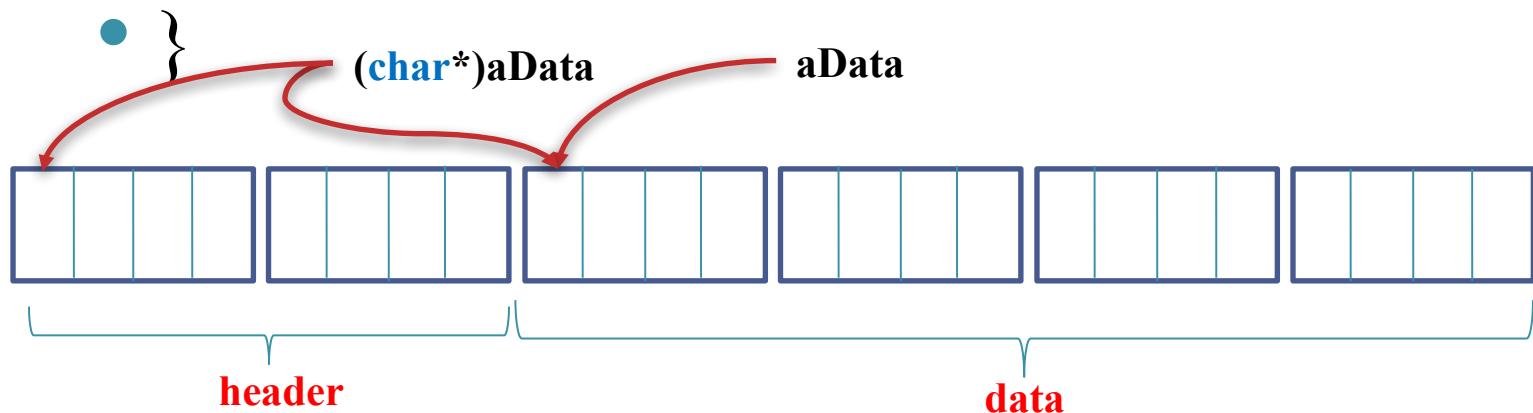
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Kích thước header: `int headSize = sizeof(int) + sizeof(int)`
- Hàm memSize: tính **tổng số byte** cần cấp
- Hàm `int memSize(int nItem, int szItem){`
 - `return headSize + nItem * szItem;`



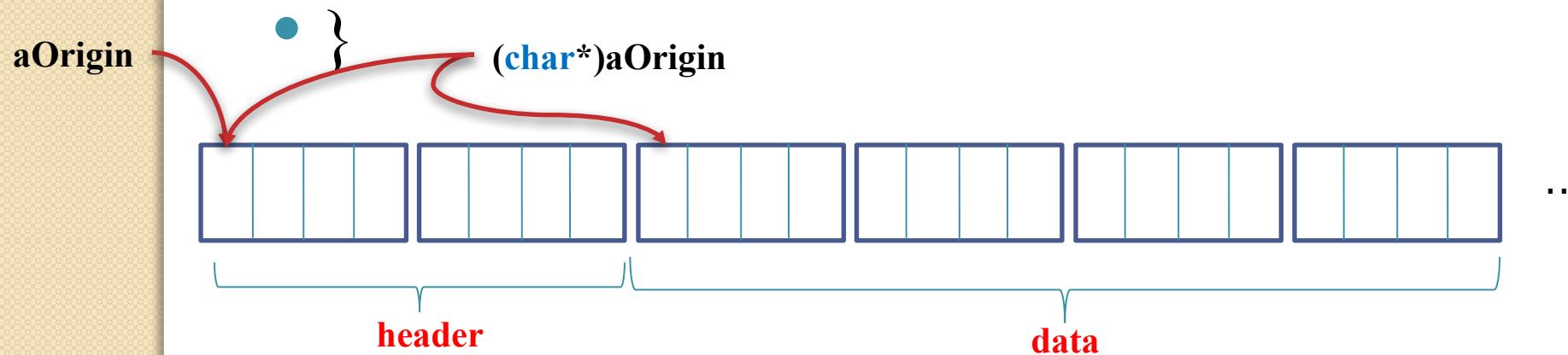
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm `void* origin_addr (void* aData){`
 - `if(aData != NULL)`
 - `return (char*)aData - headSize;`
 - `return NULL;`



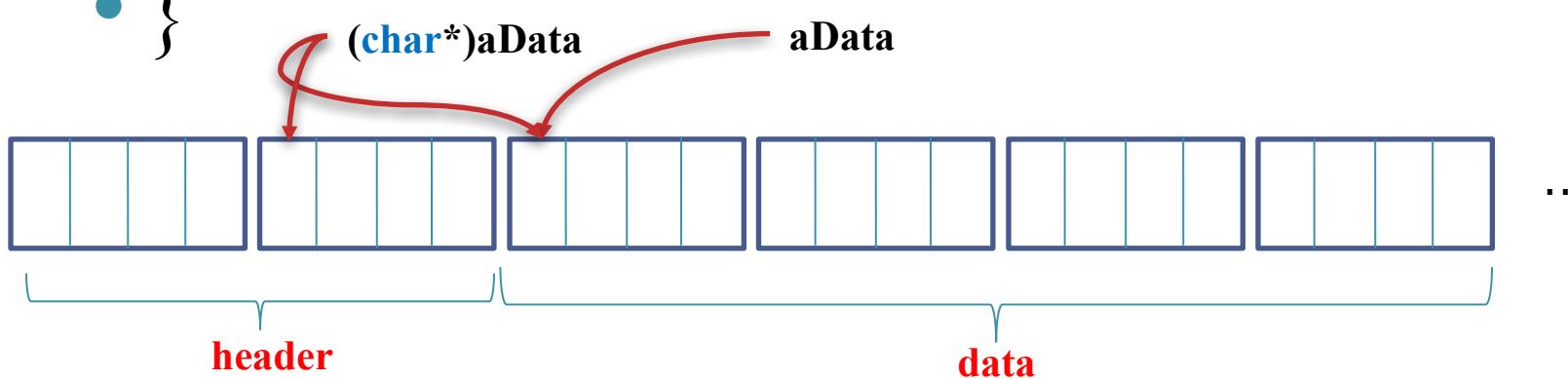
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm `void* data_addr (void* aOrigin){`
 - `if(aOrigin != NULL)`
 - `return (char*)aOrigin + headSize;`
 - `return NULL;`



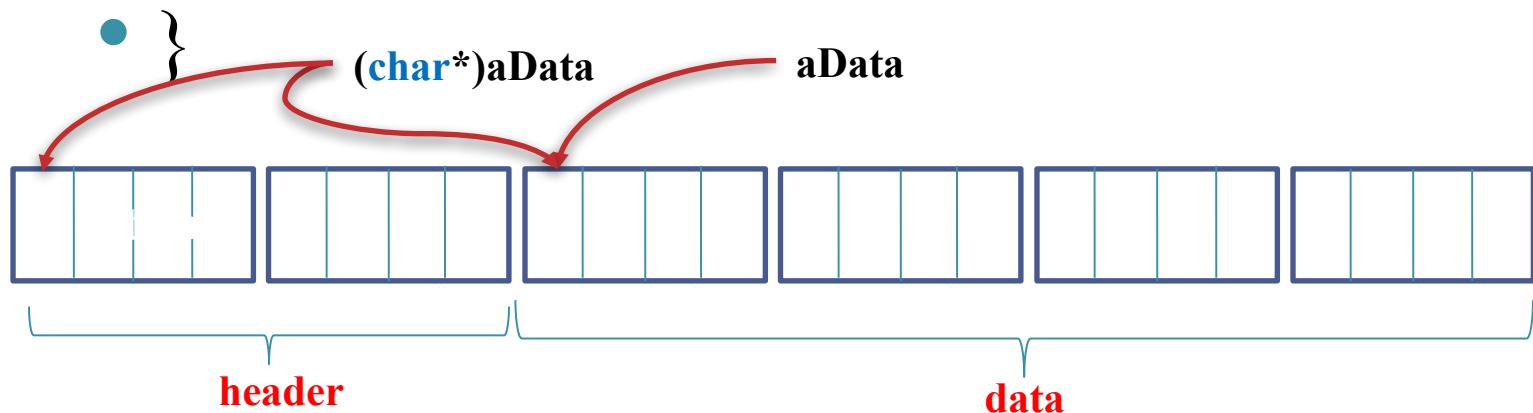
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm `void* sizeItem_addr (void* aData){`
 - `if(aData != NULL)`
 - `return (char*)aData - sizeof(int);`
 - `return NULL;`
- }



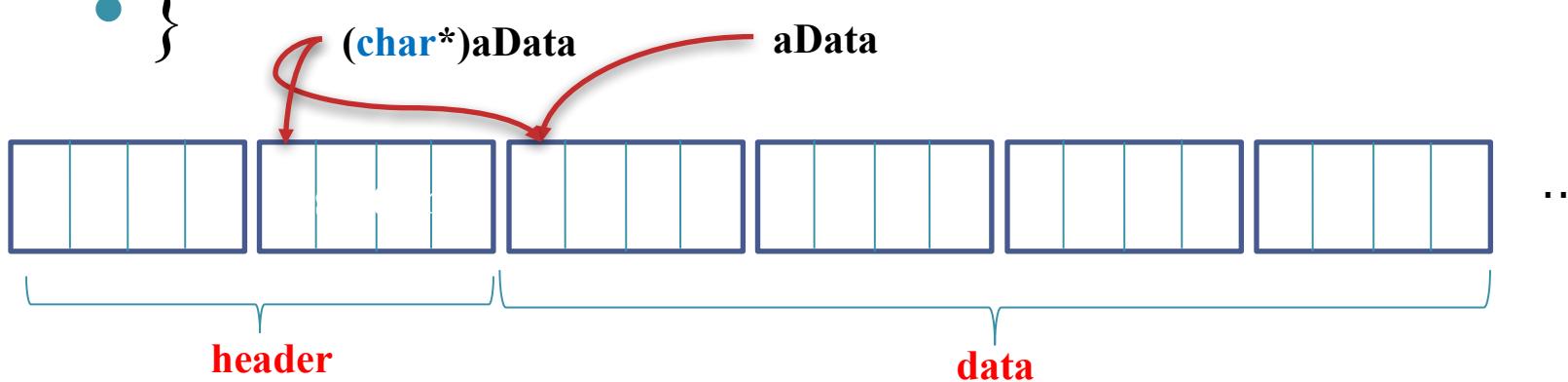
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm `int arrSize(void* aData){`
 - `if(aData != NULL)`
 - `return *(int*)origin_addr(aData);`
 - `return 0;`



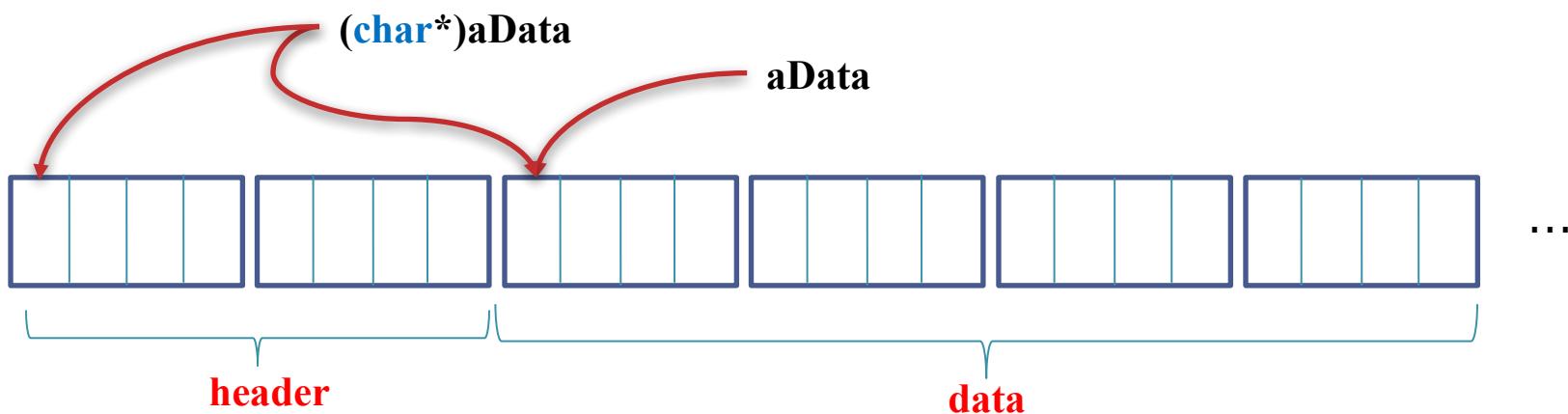
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm `int arrItemSize(void* aData){`
 - `if(aData != NULL)`
 - `return *(int*)sizeItem_addr(aData);`
 - `return 0;`
- `}`

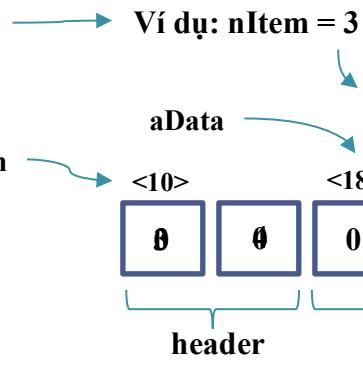


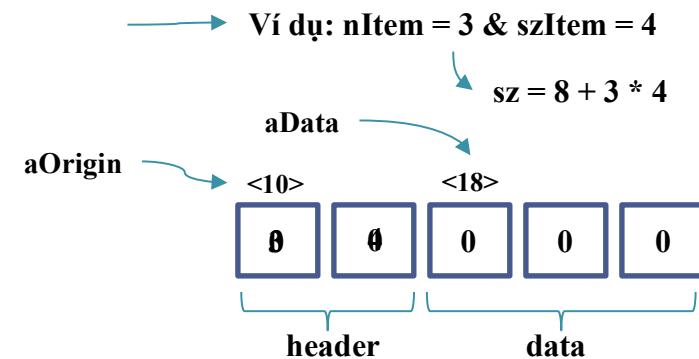
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm **void** arrFree(**void*** aData){
 - **if**(aData != NULL) free(origin_addr(aData));
- }



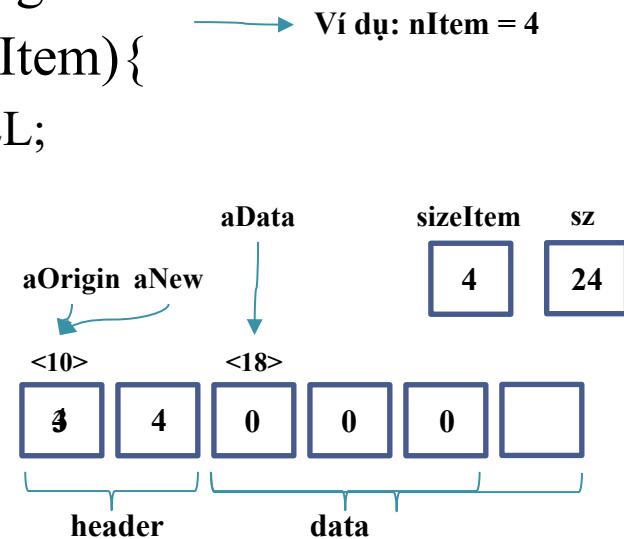
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
 - Hàm arrInit: Cấp **tổng lượng byte** cần thiết
 - Hàm `void* arrInit(int nItem, int szItem){`
 - `int sz = memSize(nItem, szItem);`
 - `void* aOrigin = malloc(sz);`
 - `if(aOrigin != NULL){`
 - `memset(aOrigin, 0, sz);`
 - `void *aData = data_addr(aOrigin);`
 - `*(int*)origin_addr(aData) = nItem;`
 - `*(int*)sizeItem_addr(aData) = szItem;`
 - `return aData;`
 - `}`
 - `return NULL;`
 - 



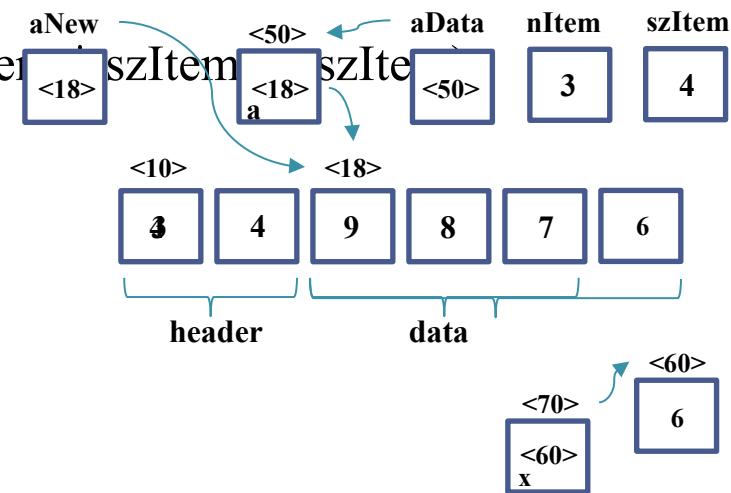
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm arrResize: Thay đổi kích thước vùng aData
- Hàm `void* arrResize(void* aData, int nItem){`
 - `if(aData == NULL || nItem < 0) return NULL;`
 - `void* aOrigin = origin_addr(aData);`
 - `int sizeItem = *(int*)sizeItem_addr(aData);`
 - `int sz = memSize(nItem, sizeItem);`
 - `void *aNew = realloc(aOrigin, sz);`
 - `if(aNew != NULL){`
 - `aData = data_addr(aNew);`
 - `*(int*)origin_addr(aData) = nItem;`
 - `return aData;`
 - `}`
 - `return NULL;`
- `}`



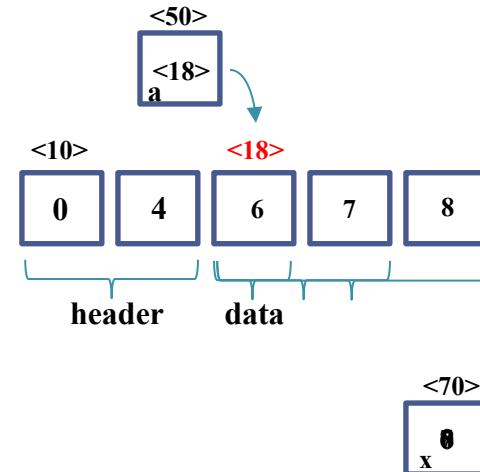
CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm arrPushback: Thêm x vào mảng một chiều $aData$
- Hàm `int arrPushback(void** aData, void* x){`
 - `int nItem = arrSize(*aData), szItem = arrItemSize(*aData);`
 - `void* aNew = arrResize(*aData, 1 + nItem);`
 - `if(aNew != NULL){`
 - `memmove((char*)aNew + nItem, *aData, szItem);`
 - `*aData = aNew;`
 - `return 1;`
 - `}`
 - `return 0;`
- }



CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Ví dụ xây dựng lại hàm floatArrIn: Tạo và trả ra mảng một chiều chứa các phần tử có kiểu **float**
- Hàm **float*** floatArrIn(){
 - **float*** a = (**float***)arrInit(0, **sizeof(float)**), x = 0;
 - **while**(cin >> x){
 - arrPushback((**void****)&a, &x);
 - }
 - cin.clear();
 - return a;
- }

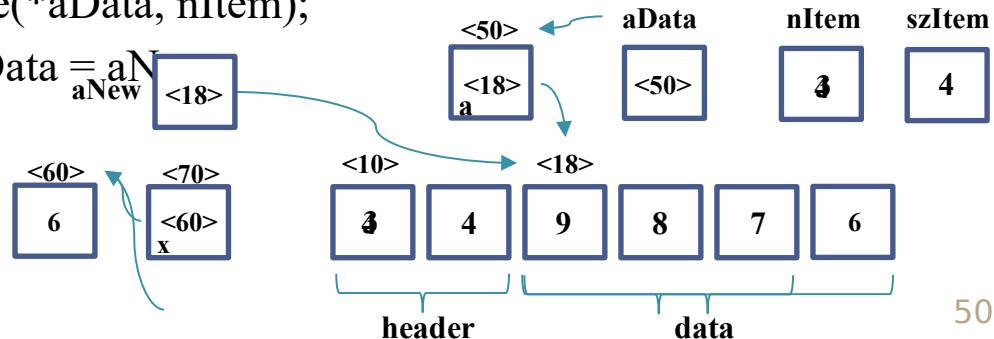


CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Ví dụ xây dựng lại hàm PhanSoArrIn: Tạo và trả ra mảng một chiều chứa các phần tử có kiểu PhanSo
- Hàm **PhanSo*** PhanSoArrIn(){
 - **PhanSo*** a = (**PhanSo***)arrInit(0, sizeof(**PhanSo**));
 - **PhanSo** x = {0, 1};
 - **while**(cin >> x){
 - arrPushback((**void****)&a, &x);
 - }
 - cin.clear();
 - **return** a;
- }

CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Hàm arrPopback: Lấy phần tử ở cuối mảng một chiều aData, đồng thời giảm kích thước xuống một phần tử
- Hàm `void* arrPopback(void** aData){`
 - `int nItem = arrSize(*aData), szItem = arrItemSize(*aData);`
 - `void* x = malloc (szItemSize);`
 - `if(*aData != NULL && nItem > 0){`
 - `nItem--;`
 - `memmove(x, (char*)(*aData) + nItem * szItem, szItem);`
 - `void* aNew = arrResize(*aData, nItem);`
 - `if(aNew != NULL) *aData = aNew;`
 - `}`
 - `return x;`
- `}`



CẤP PHÁT & DÙNG MẢNG ĐỘNG

- Mục tiêu: xây dựng các hàm tiện ích xử lý mảng một chiều kiểu TỔNG QUÁT
- Ví dụ hàm main hoàn chỉnh với **float**
- **void main()**{
 - cout << “Nhập các phần tử:\n”;
 - **float*** B = floatArrIn();
 - **float*** x = (**float***)arrPopback((**void****)&B);
 - cout << “After pop: \n”;
 - floatArrOut(B);
 - cout << “\nPopped element: ” << *x << endl;
 - free(x);
 - arrFree((**float***)B)
- }

CẤP PHÁT & DÙNG MẢNG ĐỘNG

(Toán tử mốc vuông ‘[]’)

- Mục tiêu: giúp đoạn mã tự nhiên hơn khi dùng cấu trúc tạo mảng
- Cần trả ra tham chiếu khi cài đặt toán tử []

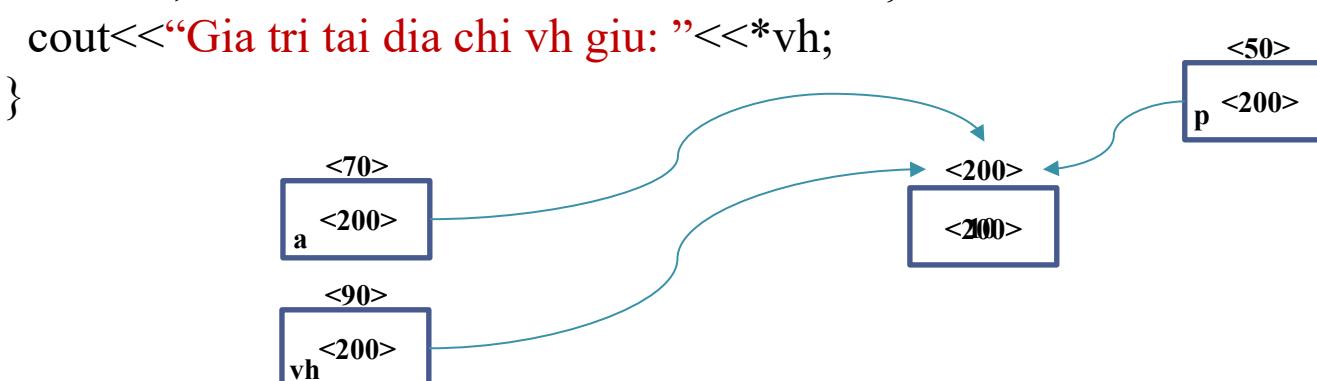
//File.h	//File.cpp
struct floatArray{	#include “File.h”
int n;	float& floatArray::operator[](int i){
float* arr;	if(arr != NULL && i >= 0 && i < n) return arr[i];
float& operator[](int);	return dummy;
};	}
void floatArrayOutput(floatArray& a){	
if(a.arr == NULL) return;	
for(int i = 0; i < a.n; i++) cout << a[i] << “ ”; // a.operator[](i)	
}	

KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Có thể chuyển biến thường → con trỏ
- Cần tự kiểm soát kiểu sau khi chuyển

```
void markMem(void* a){  
    cout<<"Dia chi cua a: "<<&a;  
    cout<<"Dia chi a giu: "<<a;  
    void** vh = (void**)a;  
    cout<<"Dia chi cua vh: "<<&vh;  
    cout<<"Dia chi vh giu: "<<vh;  
    cout<<"Gia tri tai dia chi vh giu: "<<*vh;  
    *vh = a;  
    cout<<"Gia tri tai dia chi vh giu: "<<*vh;  
}
```

```
void main(){  
    int* p = new int; *p = 10;  
    cout<<"Dia chi cua p: "<<&p;  
    cout<<"Dia chi p giu: "<<p;  
    cout<<"Gia tri tai dia chi p giu: "<<*p;  
    markMem(p);  
    free(p);  
}
```

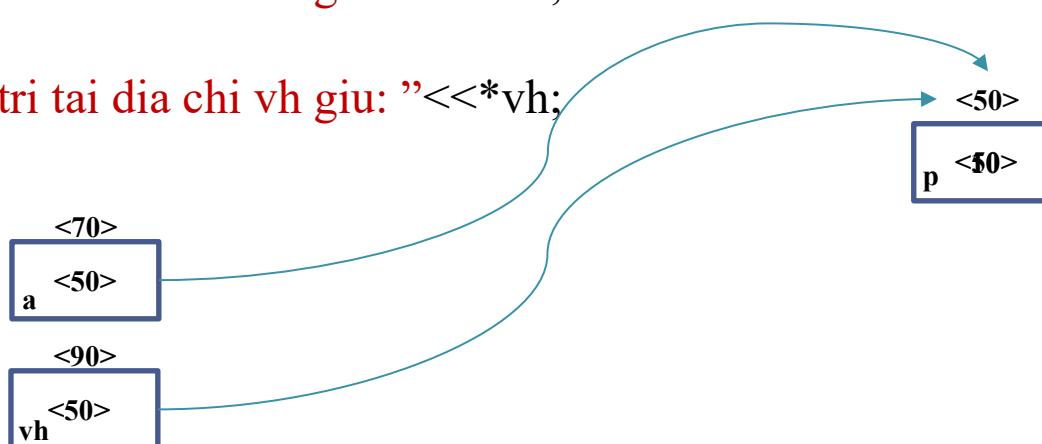


KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Có thể chuyển biến thường → con trỏ
- Cần tự kiểm soát kiểu sau khi chuyển

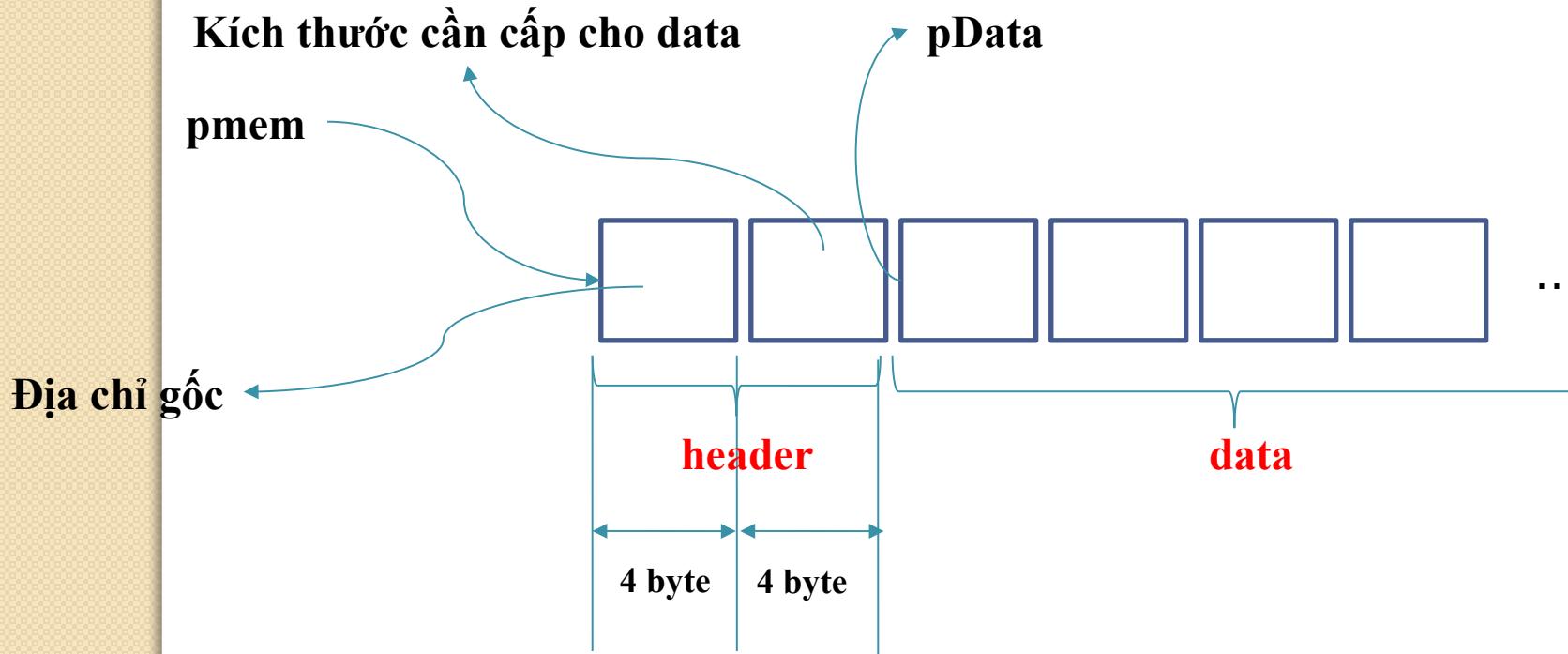
```
void markMem(void* a){  
    cout<<“Dia chi cua a: ”<<&a;  
    cout<<“Dia chi a giu: ”<<a;  
    void** vh = (void**)a;  
    cout<<“Dia chi cua vh: ”<<&vh;  
    cout<<“Dia chi vh giu: ”<<vh;  
    cout<<“Gia tri tai dia chi vh giu: ”<<*vh;  
    *vh = a;  
    cout<<“Gia tri tai dia chi vh giu: ”<<*vh;  
}
```

```
void main(){  
    int p = 10;  
    cout<<“Dia chi cua p: ”<<&p;  
    cout<<“Gia tri cua p: ”<<p;  
    markMem(&p);  
}
```



KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Tổ chức vùng nhớ thành 2 phần:
 - Header: chứa thông tin **con trỏ gốc** và **kích thước cần cấp** phần dữ liệu
 - Data: chứa nội dung dữ liệu



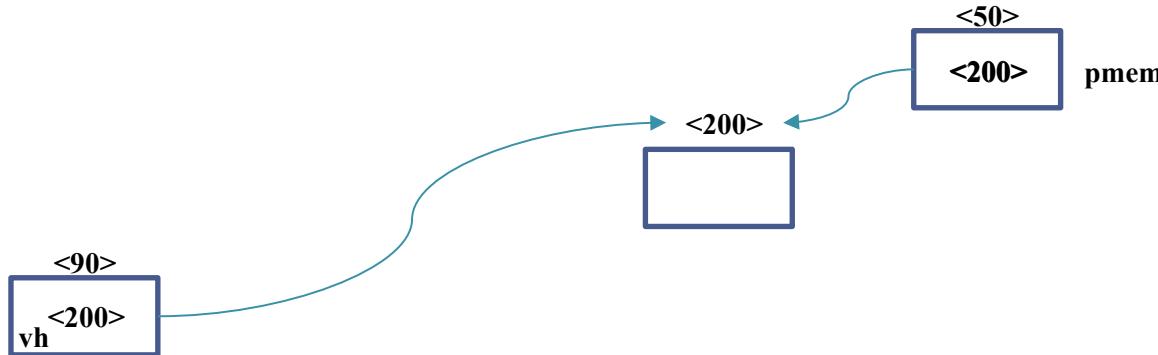
KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Xây dựng hàm markMem và checkMem

- static void markMem(void* pmem){

- *(void**)pmem = pmem;

- }



- static int checkMem(void* pmem){

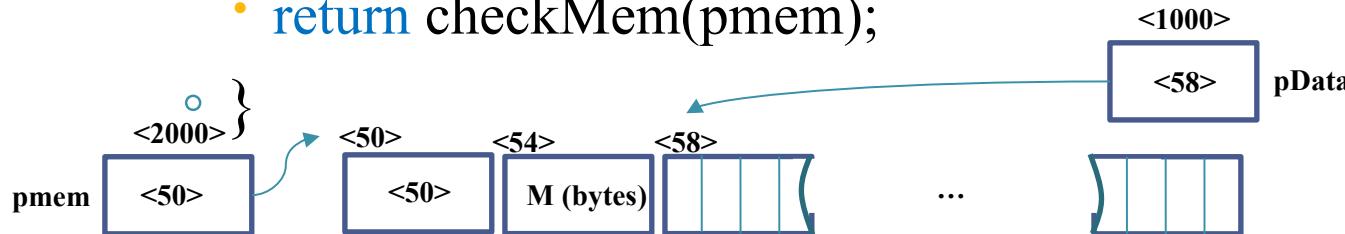
- return *(void**)pmem == pmem;

- }

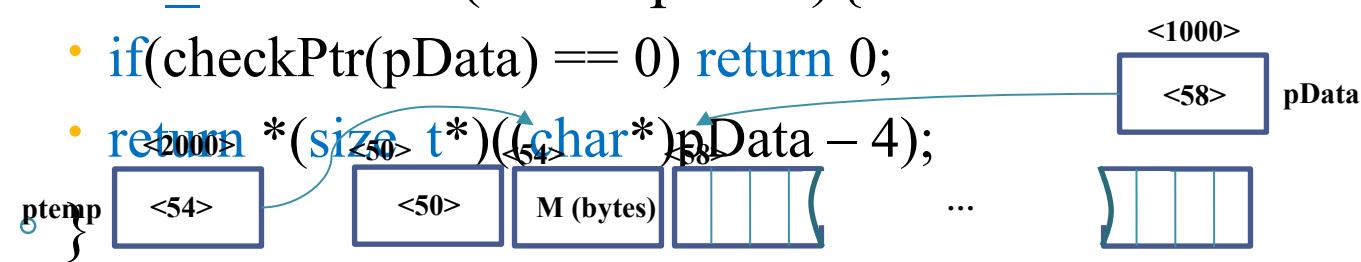
KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Xây dựng hàm checkPtr và safeSize

- `int checkPtr(void* pData){`
 - `if(pData == NULL) return 0;`
 - `void* pmem = (char*)pData - 8;`
 - `return checkMem(pmem);`



- `size_t safeSize(void* pData){`
 - `if(checkPtr(pData) == 0) return 0;`
 - `return *(size_t*)(char*)pData - 4;`



KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

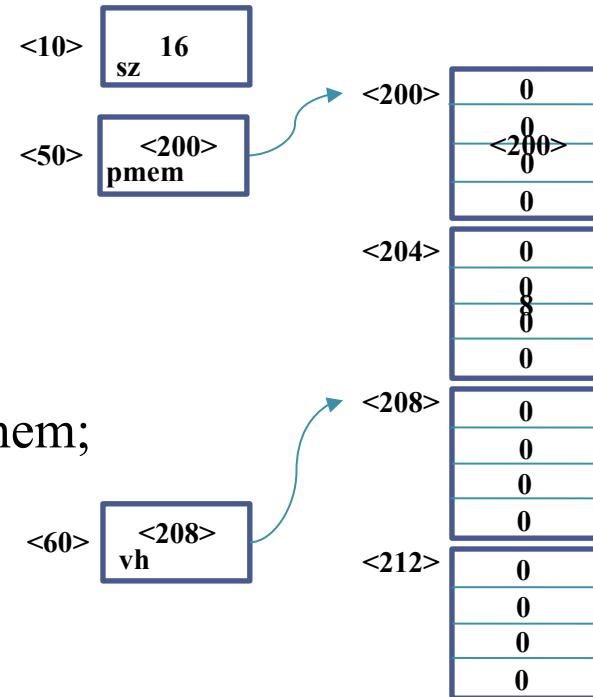
- Xây dựng hàm safeMalloc

- `void* safeMalloc(size_t szmem){`

- `size_t sz = szmem + 8;`
 - `void* pmem = malloc(sz);`
 - `if(pmem != NULL){`
 - `memset(pmem, 0, sz);`
 - `markMem(pmem);`
 - `*(size_t*)((char*)pmem + 4) = szmem;`
 - `return (char*)pmem + 8;`
 - `}`
 - `return NULL;`

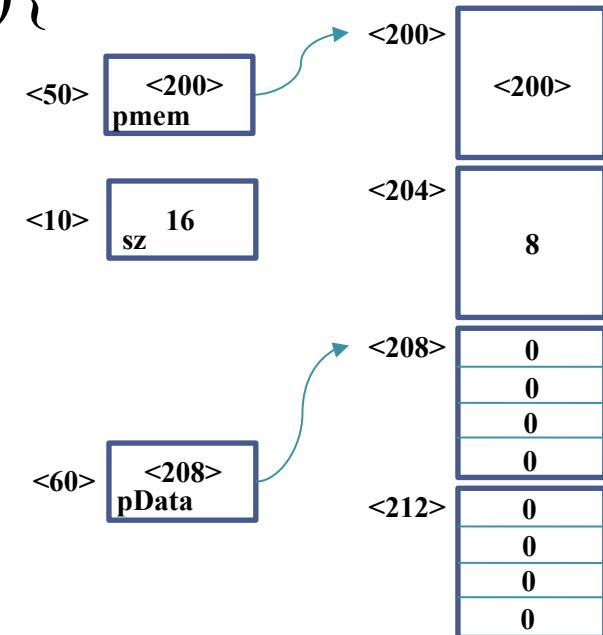
- `}`

Ví dụ: szmem = 8



KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Xây dựng hàm safeFree
 - `void* safeFree(void* pData){`
 - `if(pData != NULL){`
 - `void* pmem = (char*)pData - 8;`
 - `if(checkMem(pmem)){`
 - `size_t sz = safeSize(pData) + 8;`
 - `memset(pmem, 0 ,sz);`
 - `free(pmem);`
 - `}`
 - `}`
 - `}`



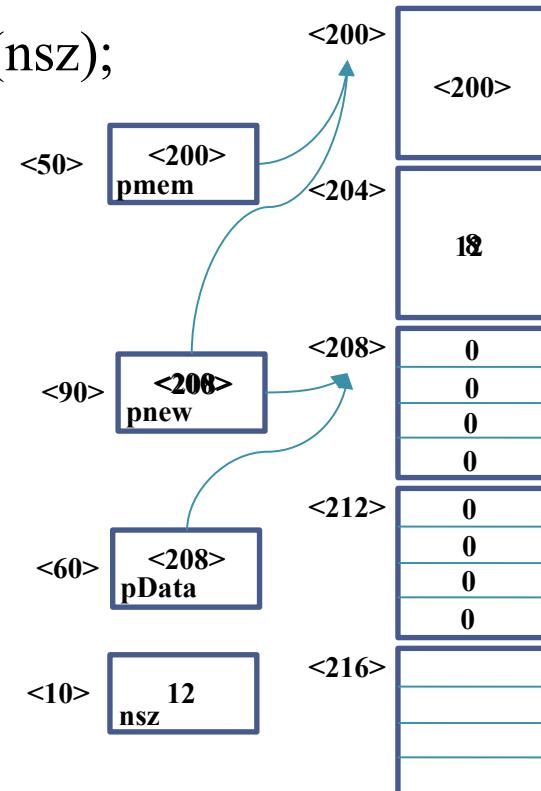
KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Xây dựng hàm safeRealloc

- ```
void* safeRealloc(void* pData, size_t nsz){
```

  - if(nsz <= 0) return NULL;
  - if(pData == NULL) return safeMalloc(nsz);
  - void\* pmem = (char\*)pData - 8;
  - if(!checkMem(pmem)) return NULL;
  - void\* pnew = realloc(pmem, nsz + 8);
  - if(pnew != NULL){
    - markMem(pnew);
    - \*(size\_t\*)((char\*)pnew + 4) = nsz;
    - pnew = (char\*)pnew + 8;
  - }
  - return pnew;
- }

Ví dụ: nsz = 12



# KỸ THUẬT KIỂM TRA CON TRỎ AN TOÀN

- Hướng dẫn sử dụng

- void main(){
  - int\* arr = (int\*)safeMalloc(sizeof(int) \* 4);
  - if(checkPtr(arr) != 0){
    - arr[0] = 1; arr[1] = 2;
    - arr[2] = 3; arr[3] = 4;
    - for(int i = 0; i < 4; i++) cout << arr[i];
  - }
  - safeFree(arr);
- }

