POINTER

PROGRAMMING TECHNIQUES

ADVISOR: Trương Toàn Thịnh

CONTENTS

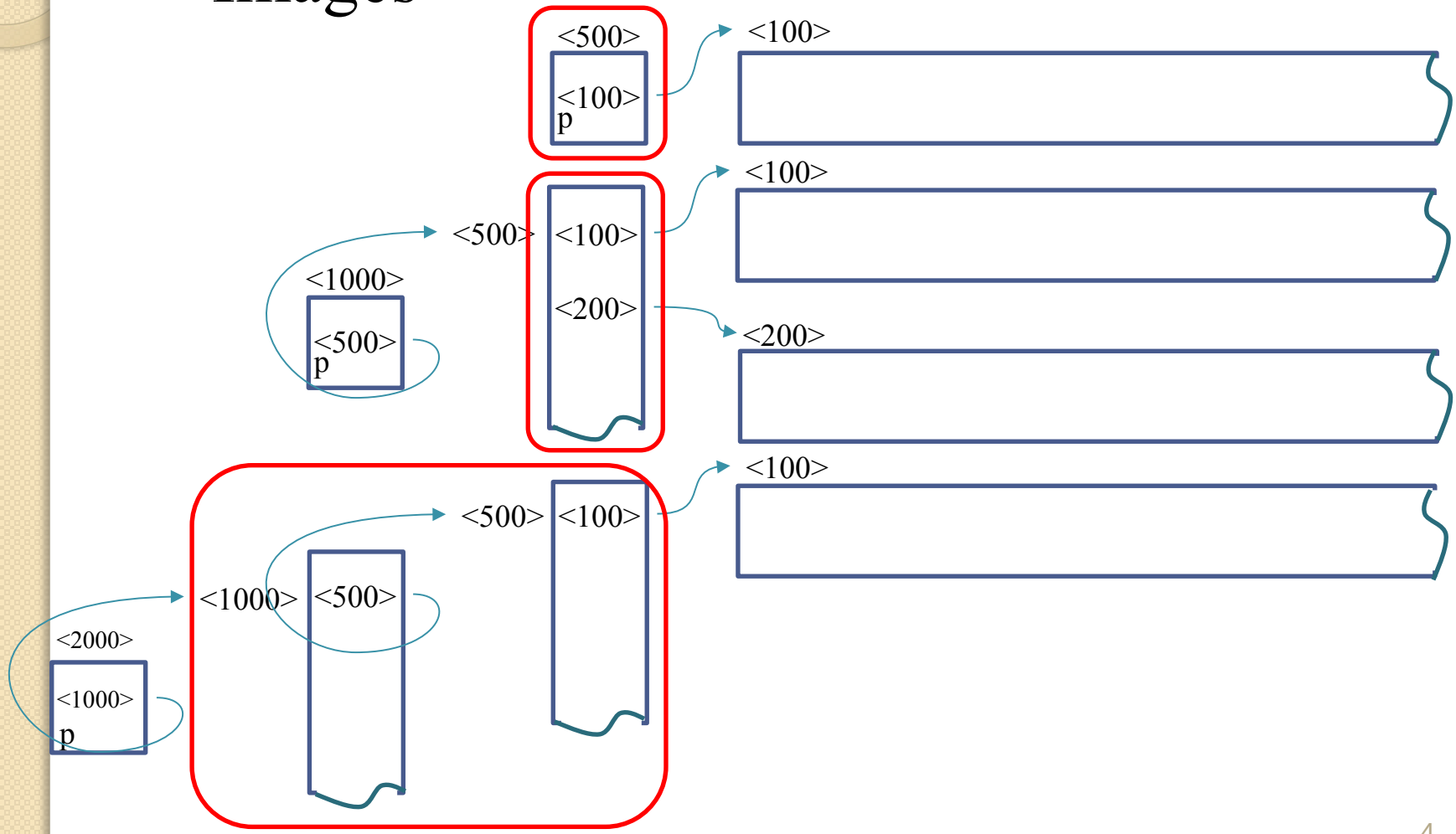
- Introduction
- Fixed-row 2D array
- Fixed-col 2D array
- Dynamic 2D array
- Dynamic multi-dimensional array

INTRODUCTION

- 2D array includes *m* rows and *n* columns
- May use pointer or vector<T> to build 2D array
- Pointer-views of array:
  - 1D array: one pointer points to memory
  - 2D array: array of pointers pointing to memory
  - 3D array: 2D array of pointers pointing to memory
  - …
- Need overloading operator '[]' for convenience

INTRODUCTION

- Images



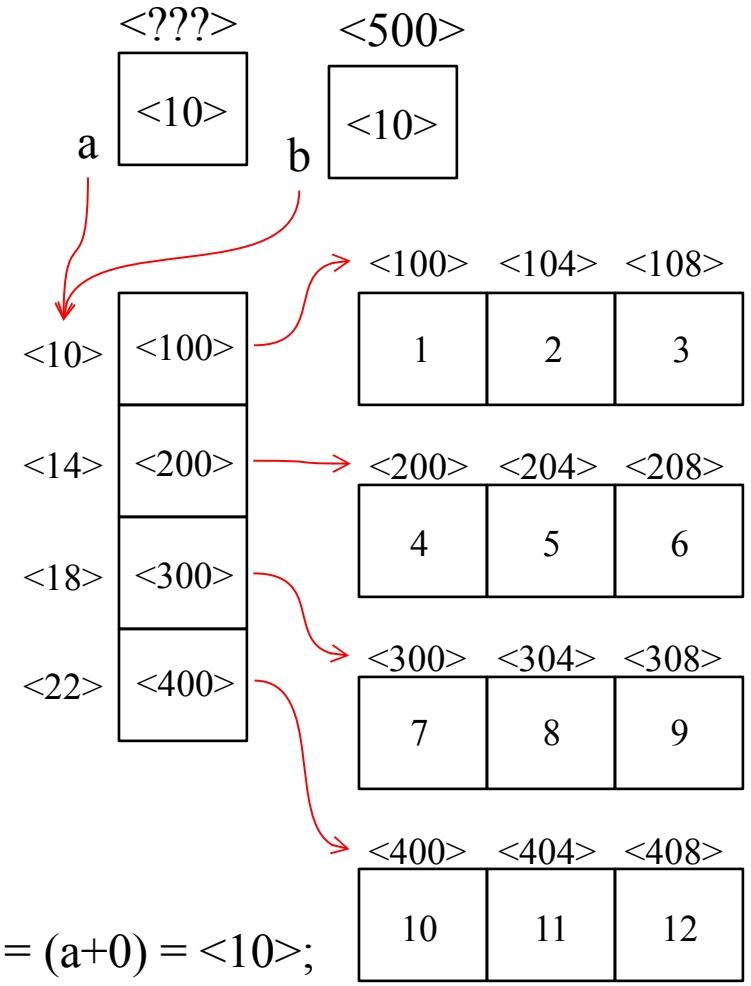
FIXED-ROW 2D ARRAY

- Be a matrix with dynamic columns
- Need to provide a number of columns used
- Need to destroy memory after using
- Need to declare 1D array of pointers standing for a **fixed-number** of rows
- Note some syntaxes of accessing the matrix's elements
- Need to have a method of allocation preventing the memory's fragmentation

FIXED-ROW 2D ARRAY

- Code

Lines	Description
1	int arr2D_alloc(int* b[], int m, int n){
2	int i, Success = 1;
3	for(i = 0; i < m; i++){
4	b[i] = (int*)malloc(n*sizeof(int));
5	if(b[i] == NULL) Success = 0;
6	}
7	return Success;}
8	void main(){
9	int n, *a[4];
10	cin >> n;
11	arr2D_alloc(a, 4, n);
12	}



&a = (a+0) = <10>;
*(a+0) = a[0] = <100>;
((a+0)) = a[0][0] = 1

FIXED-ROW 2D ARRAY

- Need to improve arr2D_alloc: immediately exit when failed to allocate
- Add int* a[4] = {NULL} into function main()

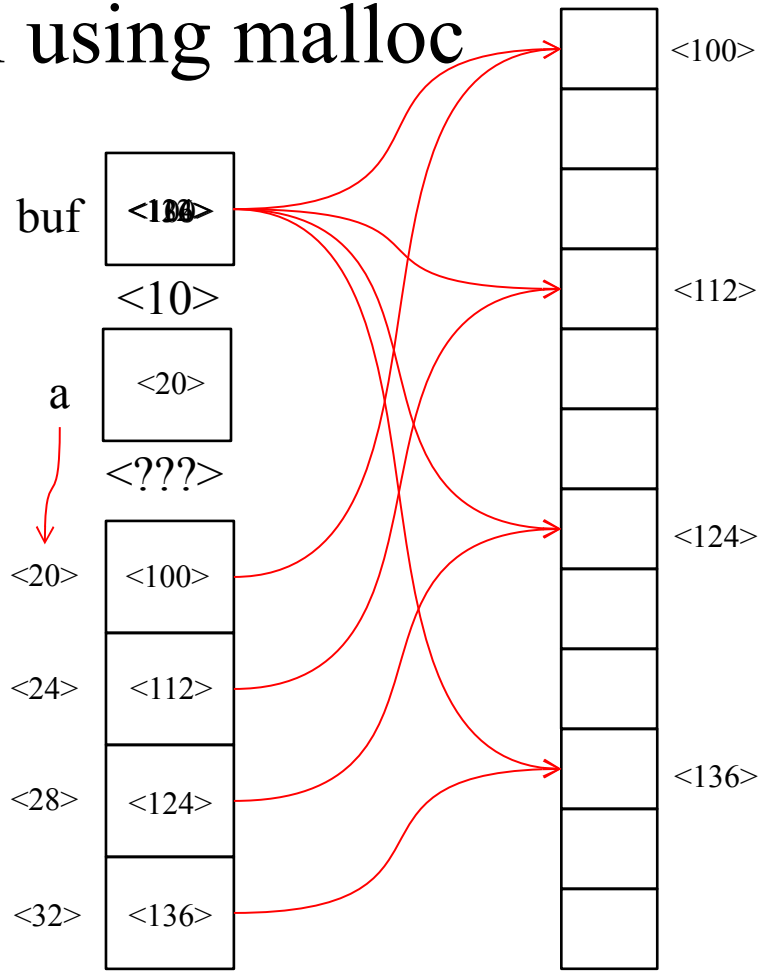
Lines	Description
1	int arr2D_alloc(int* a[], int m, int n){
2	int i, Success = 1;
3	for(i = 0; i < m; i++){
4	a[i] = (int*)malloc(n*sizeof(int));
5	if(a[i] == NULL) Success = 0;
6	}
7	return Success;
8	}

Lines	Description
9	int arr2D_alloc(int* a[], int m, int n){
10	int i, Success = 1;
11	for(i = 0; i < m; i++){
12	a[i] = (int*)malloc(n*sizeof(int));
13	if(a[i] == NULL) {
14	Success = 0; break;
15	}
16	return Success;}

FIXED-ROW 2D ARRAY

- Need to improve arr2D_alloc: prevent the fragmentation when using malloc

Lines	Description
1	int arr2D_alloc(int* a[], int m, int n){
2	int* buf = (int*)calloc(m*n, sizeof(int));
3	if(buf == NULL){
4	a[0] = NULL; return 0;
5	}
6	a[0] = buf;
7	for(int i = 1; i < m; i++){
8	buf += n; a[i] = buf
9	}
10	return 1;}



FIXED-ROW 2D ARRAY

- Using template in C++ to generalize datatype

Lines	Description	template <class T>
1	int arr2D_alloc(int* a[], int m, int n){	int arr2D_alloc(**T*** a[], int m, int n){
2	int* buf = (int*)calloc(m*n, sizeof(int));	**T*** buf = (**T***)calloc(m*n, sizeof(**T**));
3	if(buf == NULL){	if(buf == NULL){
4	a[0] = NULL; return 0;	a[0] = NULL; return 0;
5	}	}
6	a[0] = buf;	a[0] = buf;
7	for(int i = 1; i < m; i++){	for(int i = 1; i < m; i++){
8	buf += n; a[i] = buf	buf += n; a[i] = buf
9	}	}
10	return 1;}	return 1;}

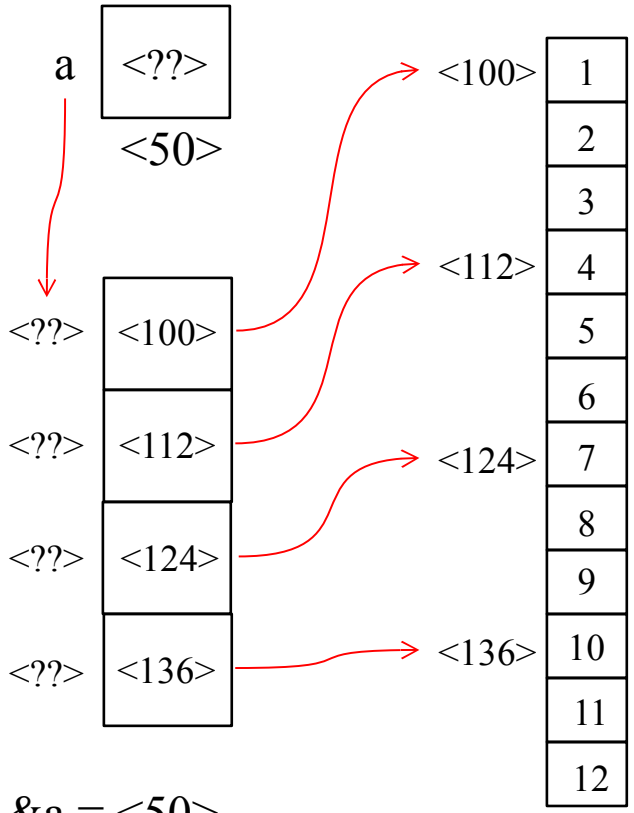
FIXED-COL 2D ARRAY

- Be a matrix with dynamic rows
- Need to provide a number of rows used
- Need to destroy memory after allocation
- Consider this is an array of 1D static arrays
- Note some syntaxes of accessing matrix's elements

FIXED-COL 2D ARRAY

- Code

Lines	Description
1	typedef float floatArr1D[3] //float[3] <-> floatArr1D
2	floatArr1D* arr2D_alloc(int m){
3	floatArr1D* a; // float[3]* a
4	a = (floatArr1D*)calloc(m, sizeof(floatArr1D));
5	return a;
6	}
7	void arr2D_input(floatArr1D* a, int m, int n){
7	for(int i = 0; i < m; i++)
8	for(int j = 0; j < n; j++)
9	cin >> a[i][j];
10	}



&a = <50>

a+0 = a[0] = <100>; a+1 = a[1] = <112>

*(a+0) = <100>; *(a+1) = <112>

((a+1) + 2) = a[1][2] = 6

FIXED-COL 2D ARRAY

- Using struct with template
  - #define NCOL 4
  - template <class T>
  - struct array1D{
    - T data[NCOL + 1];
    - T& operator[](int i){ //accessing without '.'
      - if(i >= 0 && i < NCOL) return data[i];
      - return data[NCOL];
    - }
  - };
- Example:
  - void main(){
    - array1D<int> a; a[0] = 2; //a.data[0] = 2
    - cout << a[0] << endl;
  - }

FIXED-COL 2D ARRAY

- Code

Lines	Description
1	template <class T>
2	void arr2D_alloc(array1D<T>* &a, int m){
3	a = (arr1D<T>*)calloc(m, sizeof(arr1D<T>));
4	}
5	
6	template <class T>
7	void arr2D_input(arr1D<T>* a, int m, int n){
7	for(int i = 0; i < m; i++)
8	for(int j = 0; j < n; j++)
9	cin >> a[i][j];
10	}

11	template <class T>
12	void arr2D_free(array1D<T>* a){
13	if(a != NULL) free(a);
14	}
15	
16	void main(){
17	int mB, nB = NCOL;
18	arr1D<float>* B;
19	cin>>mB; // Input row
20	arr2D_alloc(B, mB);
21	arr2D_input(B, mB, nB);
22	arr2D_free(B);
23	}

DYNAMIC 2D ARRAY

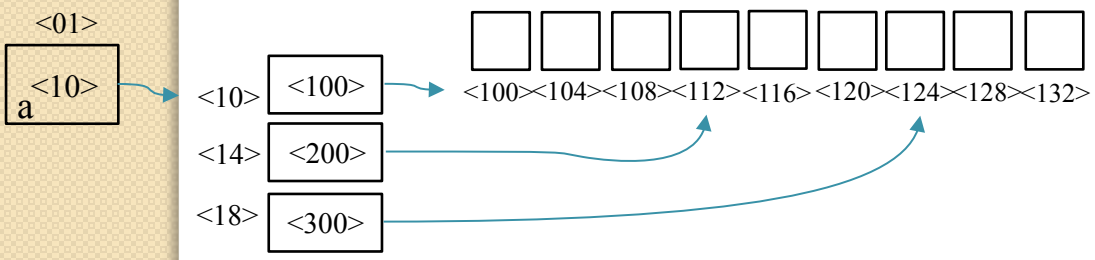
- Be a matrix with dynamic rows and columns
- Need to provide a number of columns and rows
- Need to destroy memory after allocation
- There are many methods of constructing
  - Allocate m row pointers, and each pointer points to address of memories of n elements
  - Allocate m row pointer, and a common memory of m * n elements
  - Allocate a common memory of m pointers and m * n elements

DYNAMIC 2D ARRAY

- Using int arr2D_alloc(T* a[], int m, int n) at "fixed row" to build a function of creating dynamic matrix

```
float** table_alloc(int m, int n){ // ex: m = n = 3
  float** a = (float**)calloc(m, sizeof(float*));
  if(a == NULL) return NULL;
  if(arr2D_alloc(a, m, n) == 0){ free(a); a = NULL; }
  return a;
}
```

```
template <class T>
int arr2D_alloc(T* a[], int m, int n){
  T* buf = (T*)calloc(m*n, sizeof(T));
  if(buf == NULL){
    a[0] = NULL; return 0;
  }
  a[0] = buf;
  for(int i = 1; i < m; i++){
    buf += n; a[i] = buf
  }
  return 1;}
```

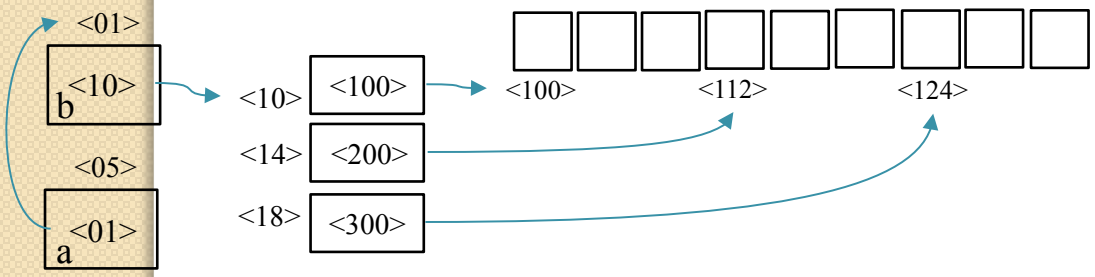


DYNAMIC 2D ARRAY

- Using int arr2D_alloc(T* a[], int m, int n) at "fixed row" to build a function of creating dynamic matrix

```
float** table_alloc(int m, int n){...}//version 1
void table_alloc(float*** a, int m, int n){//version 2
  *a = table_alloc(m, n);
}
void main{ float**b; table_alloc(&b, 3, 3);}
```

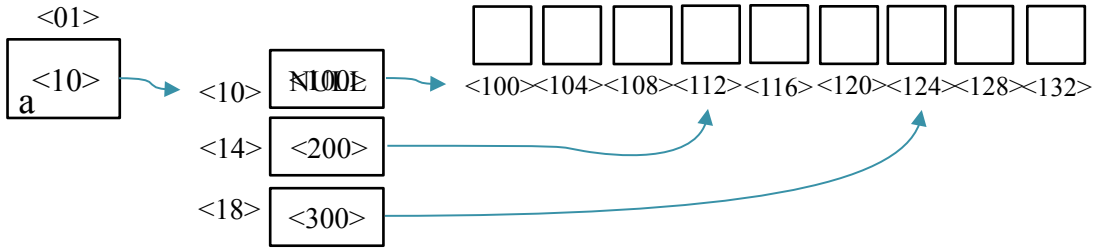
```
template <class T>
int arr2D_alloc(T* a[], int m, int n){
  T* buf = (T*)calloc(m*n, sizeof(T));
  if(buf == NULL){
    a[0] = NULL; return 0;
  }
  a[0] = buf;
  for(int i = 1; i < m; i++){
    buf += n; a[i] = buf
  }
  return 1;}
```



DYNAMIC 2D ARRAY

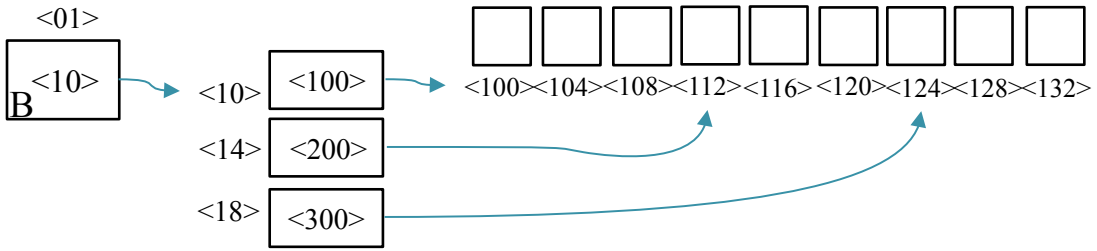
- Using void arr2D_free(**T*** a[]) at "fixed row" to build a function of destroying dynamic matrix

void table_free(float** a, int m){ // **ex: m = 3**	template <class T>
if(a == NULL || m <= 0) return;	void **arr2D_free**(**T*** a[]){
arr2D_free(a);	if(a[0] != NULL){
free(a);	free(a[0]); a[0] = NULL;
}	}



DYNAMIC 2D ARRAY

- Guiding of using table_alloc and table_free
  - void main{
    - int row, col;
    - cin>>row>>col; // ex: row = col = 3
    - float** B = table_alloc(row, col);
    - table_free(B, row);
  - }



DYNAMIC 2D ARRAY

- Add template to table_alloc and table_free

```
template <class T>
void table_alloc(T*** a, int m, int n){
  *a = (T**)calloc(m, sizeof(T*));
  if(*a == NULL) return;
  if(!arr2D_alloc(*a, m, n)){
    free(*a); *a = NULL;
  }
}
```

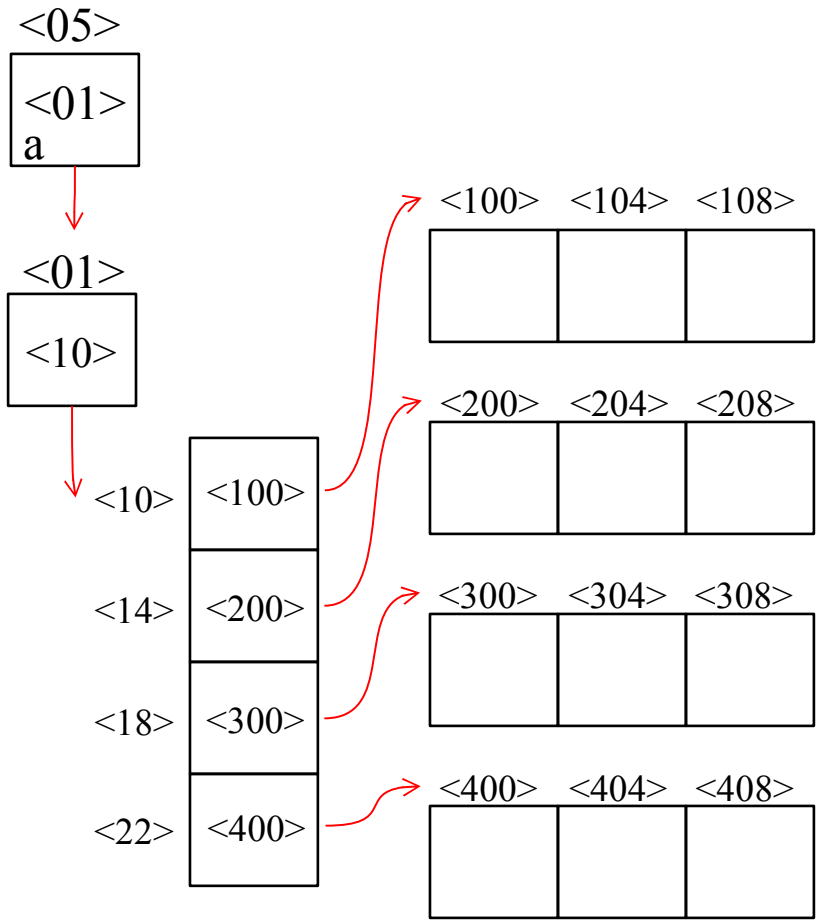
```
template <class T>
void table_free(T** a, int m){
  if(a == NULL || m <= 0) return;
  arr2D_free(a);
  free(a);
}
```

- Note: arr2D_alloc and arr2D_free are template function

DYNAMIC 2D ARRAY (SOME METHODS)

- Method 1: allocate m row pointers, each pointer points to memory's address of n elements

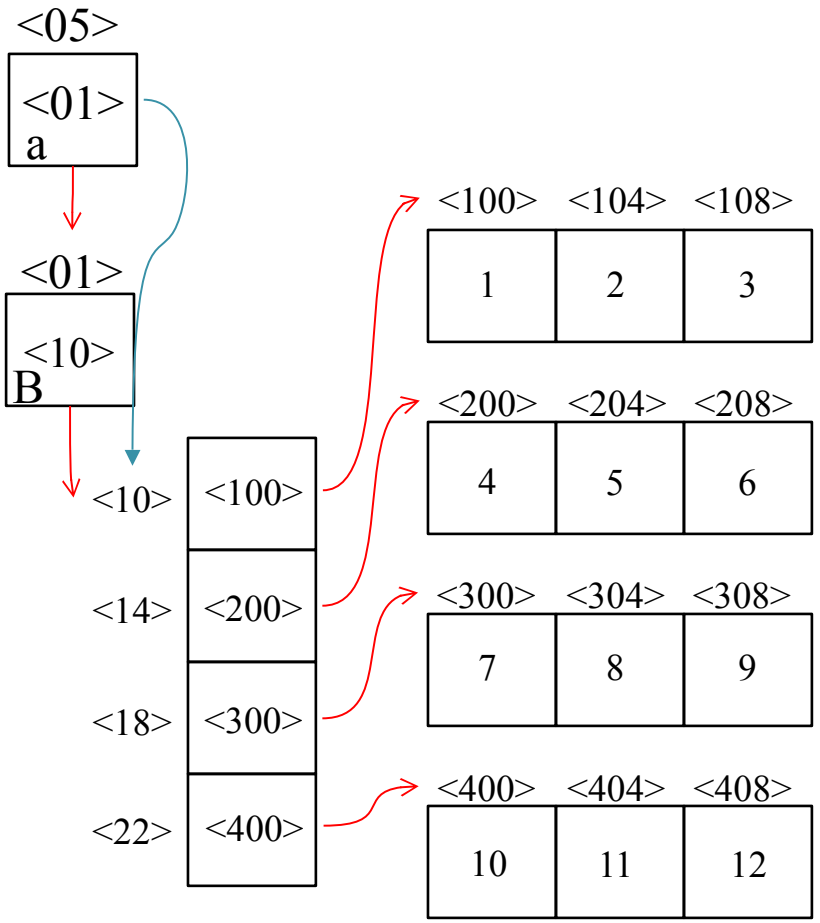
Lines	Description
1	void arr2D_alloc(float*** a, int m, int n){
2	int Success = 1;
3	*a = (float**)calloc(m, sizeof(float*));
4	for(int i = 0; i < m; i++){
5	(*a)[i] = (float*)calloc(n, sizeof(float));
6	if((*a)[i] == NULL){
7	Success = 0; break;
8	}
9	if(!Success){
10	arr2D_free(*a, m); *a = NULL;
11	}
12	}



DYNAMIC 2D ARRAY (SOME METHODS)

- Method 1: allocate m row pointers, each pointer points to memory's address of n elements

Lines	Description
1	void arr2D_free(float** a, int m){
2	if(a == NULL || m <= 0) return;
3	for(int i = 0; i < m; i++){
4	if(a[i] != NULL) free(a[i]);
5	}
6	free(a);
7	}
8	void main(){
9	int d, c; float** B; cin >> d >> c;//d=4;c=3
10	arr2D_alloc(&B, d, c); arr2D_input(B, d, c)
11	arr2D_output(B, d, c); arr2D_free(B, d);
12	}



DYNAMIC 2D ARRAY (SOME METHODS)

- Method 1: using template in C++

	template <class T>
void arr2D_alloc(float*** a, int m, int n){	void arr2D_alloc(T*** a, int m, int n){
int Success = 1;	int Success = 1;
*a = (float**)calloc(m, sizeof(float*));	*a = (T**)calloc(m, sizeof(T*));
for(int i = 0; i < m; i++){	for(int i = 0; i < m; i++){
(*a)[i] = (float*)calloc(n, sizeof(float));	(*a)[i] = (T*)calloc(n, sizeof(T));
if((*a)[i] == NULL){Success = 0; break;}	if((*a)[i] == NULL){Success = 0; break;}
if(!Success){arr2D_free(*a, m); *a = NULL;}	if(!Success){arr2D_free(*a, m); *a = NULL;}
}}	}}
	template <class T>
void arr2D_free(float** a, int m){	void arr2D_free(T** a, int m){
if(a == NULL || m <= 0) return;	if(a == NULL || m <= 0) return;
for(int i = 0; i < m; i++){	for(int i = 0; i < m; i++){
if(a[i] != NULL) free(a[i]);	if(a[i] != NULL) free(a[i]);
}	}
free(a);}	free(a);}

DYNAMIC 2D ARRAY (SOME METHODS)

- Method 1: using void* in C

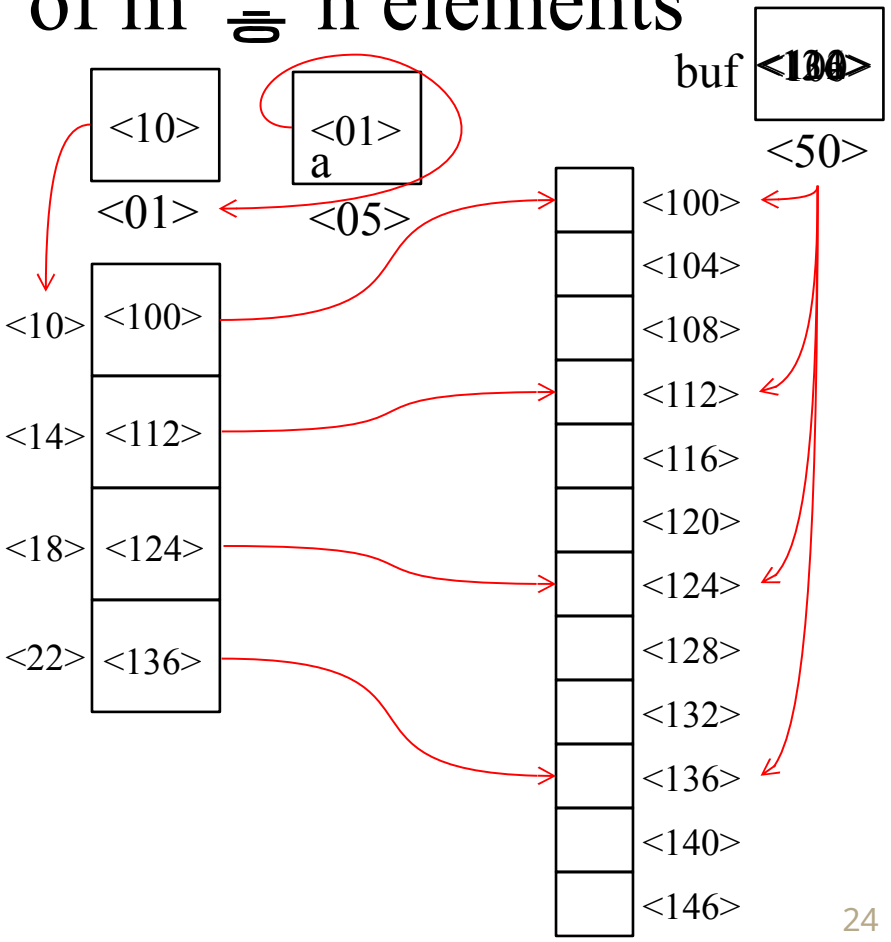
```
void arr2D_alloc(float*** a, int m, int n){
  int Success = 1;
  *a = (float**)calloc(m, sizeof(float*));
  for(int i = 0; i < m; i++){
    (*a)[i] = (float*)calloc(n, sizeof(float));
    if((*a)[i] == NULL){Success = 0; break;}
    if(!Success){arr2D_free(*a, m); *a = NULL;}
  }}
void arr2D_free(float** a, int m){
  if(a == NULL || m <= 0) return;
  for(int i = 0; i < m; i++){
    if(a[i] != NULL) free(a[i]);
  }
  free(a);}
```

```
void arr2D_alloc(void*** a, int m, int n, int szItem){
  int Success = 1;
  *a = (void**)calloc(m, sizeof(void*));
  for(int i = 0; i < m; i++){
    (*a)[i] = (void*)calloc(n, szItem);
    if((*a)[i] == NULL){Success = 0; break;}
    if(!Success){arr2D_free(*a, m); *a = NULL;}
  }}
void arr2D_free(void** a, int m){
  if(a == NULL || m <= 0) return;
  for(int i = 0; i < m; i++){
    if(a[i] != NULL) free(a[i]);
  }
  free(a);}
```

DYNAMIC 2D ARRAY (SOME METHODS)

- Method 2: Allocate m row pointers, and a common memory of m * n elements

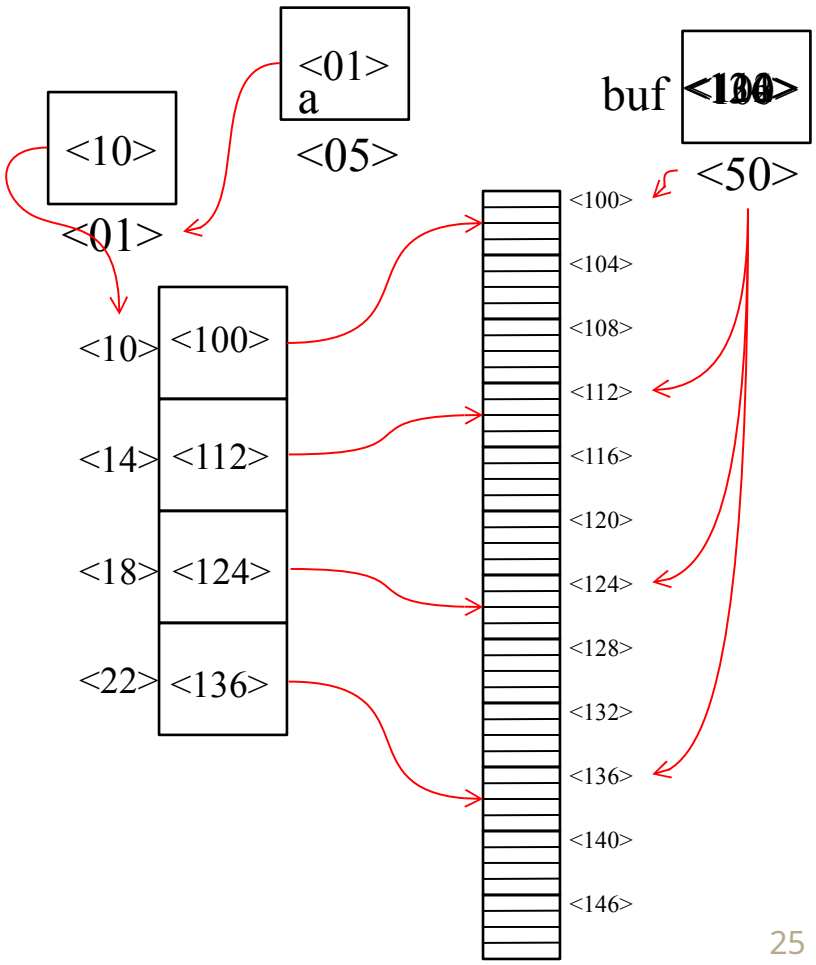
	Description
1	void arr2D_alloc(float*** a, int m, int n){
2	if(m <= 0 || n <= 0) return;
3	*a = (float**)calloc(m, sizeof(float*));
4	if(*a == NULL) return;
5	float* buf = (float*)calloc(m * n, sizeof(float));
6	if(!buf) { free(*a); return; }
7	(*a)[0] = buf; // *(*a + 0) = buf
8	for(int i = 1; i < m; i++){
9	buf += n; (*a)[i] = buf; // *(*a + i) = buf
10	}
11	}



DYNAMIC 2D ARRAY (SOME METHODS)

- Method 2: using void* in C

	Description
1	void arr2D_alloc(void*** a, int m, int n, int sItem){
2	if(m <= 0 || n <= 0) return;
3	*a = (void**)calloc(m, sizeof(void*));
4	if(*a == NULL) return;
5	void* buf = (void*)calloc(m * n, szItem);
6	if(!buf) { free(*a); return; }
7	*((*a) + 0) = buf; int szRow = n * szItem;
8	for(int i = 1; i < m; i++){
9	buf = (char*)buf + szRow;
10	*(*a + i) = buf;
11	}
12	}



DYNAMIC 2D ARRAY (SOME METHODS)

- Methods 2: using template in C++

	template <class T>
void arr2D_alloc(float*** a, int m, int n){	void arr2D_alloc(**T**** a, int m, int n){
if(m <= 0 || n <= 0) return;	if(m <= 0 || n <= 0) return;
*a = (float**)calloc(m, sizeof(float*));	*a = (**T****)calloc(m, sizeof(**T***));
if(*a == NULL) return;	if(*a == NULL) return;
float* buf = (float*)calloc(m * n, sizeof(float));	**T*** buf = (**T***)calloc(m * n, sizeof(**T**));
if(!buf) { free(*a); return; }	if(!buf) { free(*a); return; }
(*a)[0] = buf; // *(*a + 0) = buf	(*a)[0] = buf; // *(*a + 0) = buf
for(int i = 1; i < m; i++){	for(int i = 1; i < m; i++){
buf += n; (*a)[i] = buf; // *(*a + i) = buf	buf += n; (*a)[i] = buf; // *(*a + i) = buf
}}	}}
	template <class T>
void arr2D_free(void** a){	void arr2D_free(**T**** a){
if(!a){	if(!a){
if(a[0] != NULL) free(a[0]);	if(a[0] != NULL) free(a[0]);
free(a); }}	free(a); }}

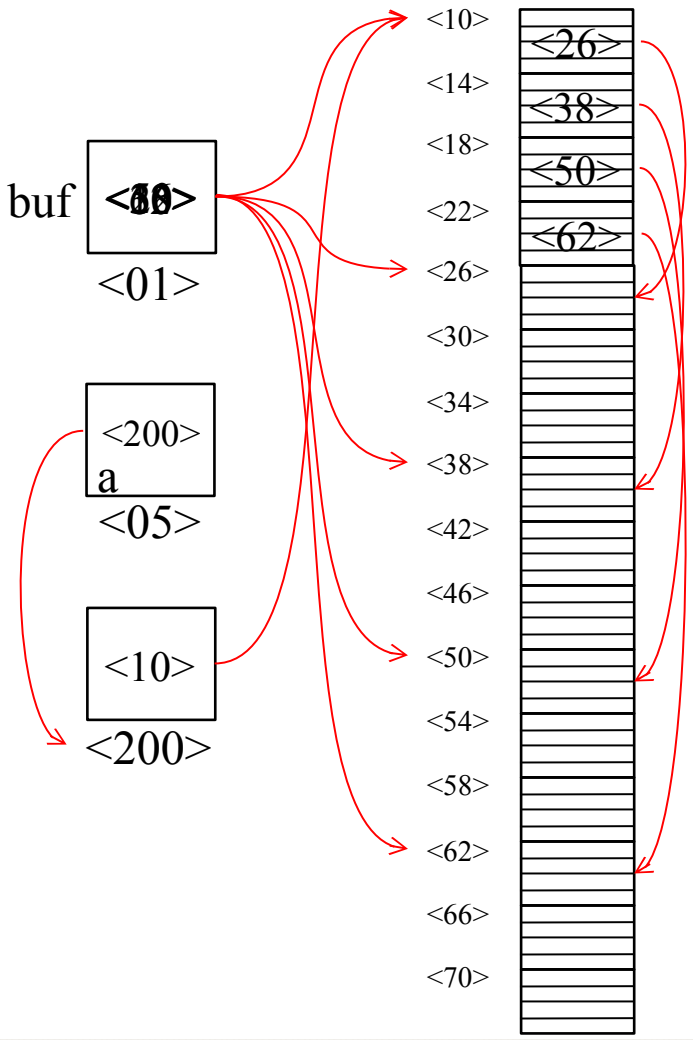
DYNAMIC 2D ARRAY (SOME METHODS)

- Method 2: demonstrate how to use void* and template
  - void main(){
    - int d, c;
    - float** B;
    - cin >> d >> c;
    - arr2D_alloc((void***)&B, d, c, sizeof(float));
    - **//arr2D_alloc<float>(&B, d, c);**
    - arr2D_input(B, d, c);
    - arr2D_output(B, d, c);
    - arr2D_free((void**)B);
    - **//arr2D_free<float>(B);**
  - }

DYNAMIC 2D ARRAY (SOME METHODS)

- Method 3: Allocate a common memory of m row pointers and m * n elements

Lines	Description
1	void arr2D_alloc(float*** a, int m, int n){
2	if(m <= 0 || n <= 0) return;
3	int sz1 = m*sizeof(float*); //example m = 4
4	int sz2 = m*n*sizeof(float);// example n = 3
5	void* buf = calloc(sz1 + sz2, 1);
6	if(buf == NULL) return;
7	*a = (float**)buf; buf = (char*)buf + sz1;
8	(*a)[0] = (float*)buf;
9	for(int i = 1; i < m; i++){
10	buf = (float*)buf + n; (*a)[i] = (float*)buf;
11	}
12	}



DYNAMIC 2D ARRAY (SOME METHODS)

- Method 3: using void* in C

	Description	
1	void arr2D_alloc(void*** a, int m, int n, int szItem){	void arr2D_alloc(float*** a, int m, int n){
2	if(m <= 0 || n <= 0) return;	if(m <= 0 || n <= 0) return;
3	int sz1 = m * sizeof(void*);	int sz1 = m * sizeof(float*);
4	int sz2 = m * n * szItem, szRow = n * szItem;	int sz2 = m * n * sizeof(float);
5	void* buf = calloc(sz1 + sz2, 1);	void* buf = calloc(sz1 + sz2, 1);
6	if(buf == NULL) return;	if(buf == NULL) return;
7	*a = (void**)buf; buf = (char*)buf + sz1;	*a = (float**)buf; buf = (char*)buf + sz1;
8	(*a)[0] = (void*)buf;	(*a)[0] = (float*)buf;
9	for(int i = 1; i < m; i++){	for(int i = 1; i < m; i++){
10	buf = (char*)buf + szRow;	buf = (float*)buf + n;
11	(*a)[i] = buf;	(*a)[i] = (float*)buf;
12	}	}
13	}	}

DYNAMIC 2D ARRAY (SOME METHODS)

- Method 3: using template in C++

	template <class T>	
1	void arr2D_alloc(T*** a, int m, int n){	void arr2D_alloc(float*** a, int m, int n){
2	if(m <= 0 || n <= 0) return;	if(m <= 0 || n <= 0) return;
3	int sz1 = m * sizeof(T*);	int sz1 = m * sizeof(float*);
4	int sz2 = m * n * sizeof(T);	int sz2 = m * n * sizeof(float);
5	void* buf = calloc(sz1 + sz2, 1);	void* buf = calloc(sz1 + sz2, 1);
6	if(buf == NULL) return;	if(buf == NULL) return;
7	*a = (T**)buf; buf = (char*)buf + sz1;	*a = (float**)buf; buf = (char*)buf + sz1;
8	(*a)[0] = (T*)buf;	(*a)[0] = (float*)buf;
9	for(int i = 1; i < m; i++){	for(int i = 1; i < m; i++){
10	buf = (T*)buf + n;	buf = (float*)buf + n;
11	(*a)[i] = (T*)buf;	(*a)[i] = (float*)buf;
12	}	}
13	}	}

DYNAMIC 2D ARRAY (SOME METHODS)

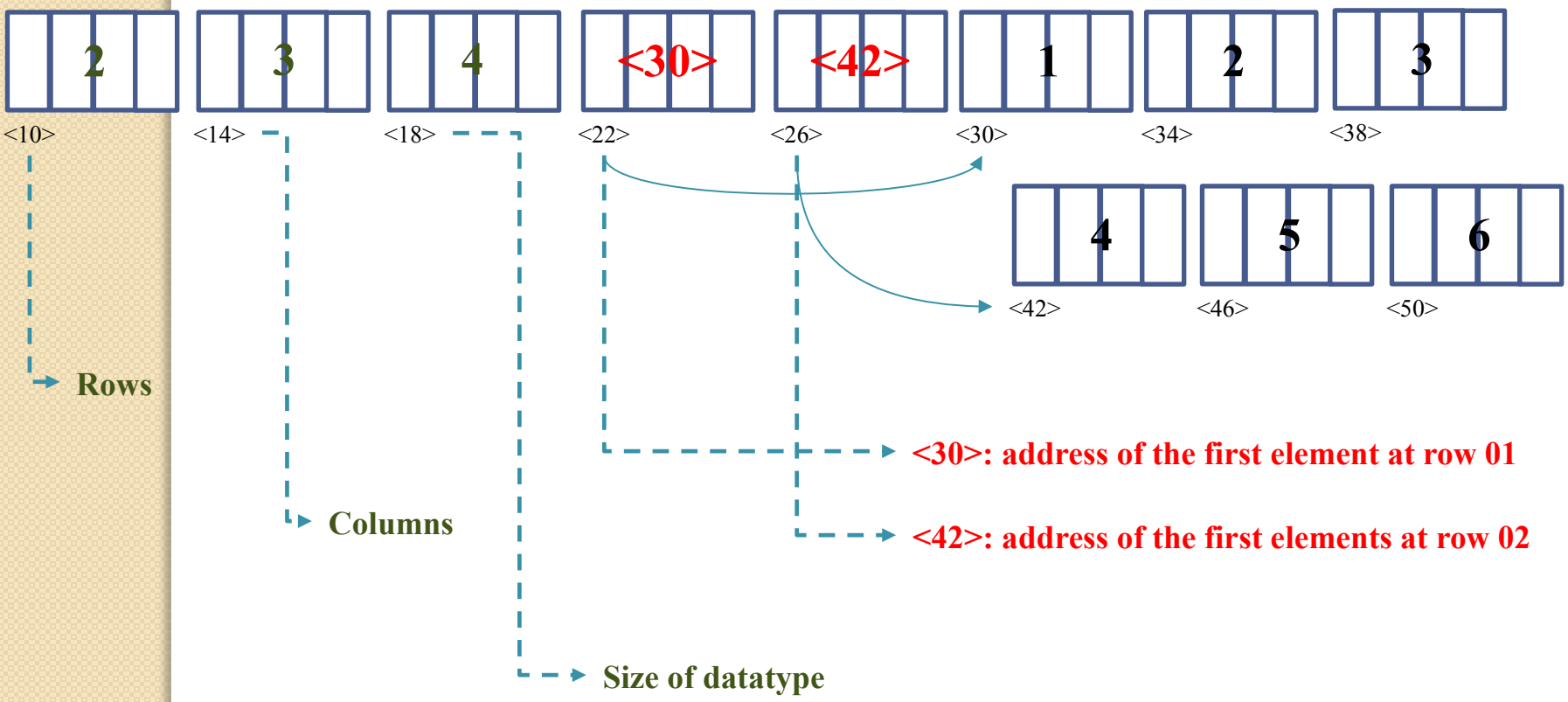
- Method 3: implement destroy function very easily
  - void arr2D_free(float** a){
    - if(a != NULL) free(a);
  - }
  - void arr2D_free(void** a){
    - if(a != NULL) free(a);
  - }
  - template <class T>
  - void arr2D_free(T** a){
    - if(a != NULL) free(a);
  - }

DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- Goals:
  - Simple and easy-to-use programming interface
  - Can be used with various datatype
  - Should build the functions with simple datatypes
  - Process the memory problems with pointer
  - Lightweight source code
- Some key functions
  - void alloc2D(void***, int, int, int);
  - void free2D(void**);
  - void resize2D(void***, int, int);
  - int nRow(void**);
  - int nCol(void**);

DYNAMIC 2D ARRAY (CONVENIENT METHODS)

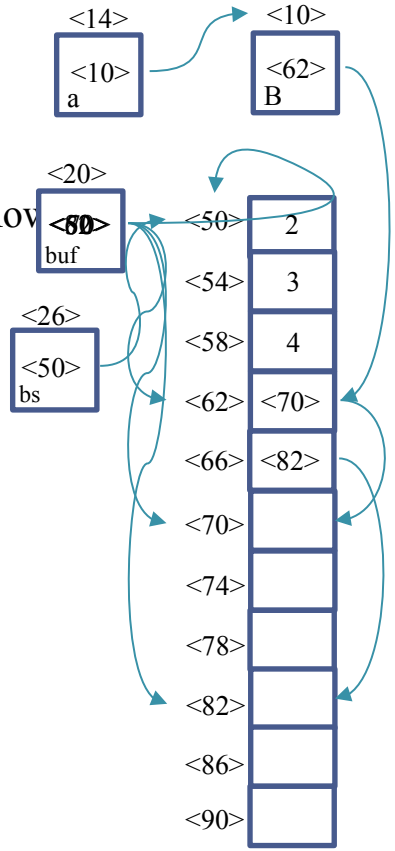
- Data structure (Example of an array of 2 rows 3 columns with float)



DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- How to use from main()

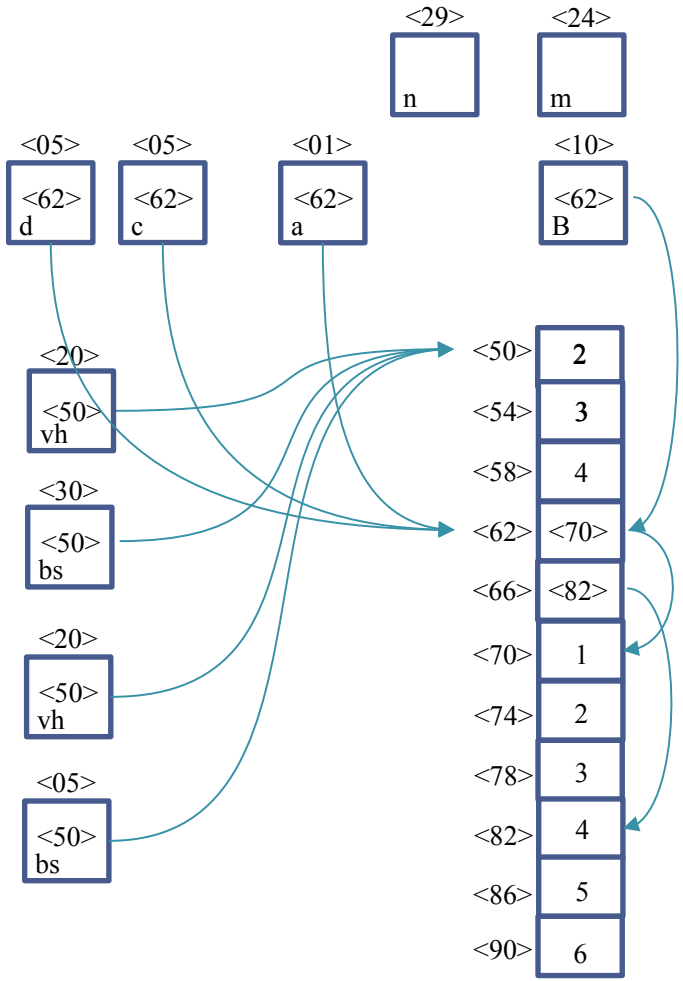
```
void main(){
    int d, c, **B; cin >> d >> c; // d = 2, c = 3
    alloc2D((void***)&B, d, c, sizeof(int));
}
void alloc2D(void*** a, int m, int n, int szItem){
        if(m <= 0 || n <= 0 || szItem <= 0) return;
        int szRow = n*szItem, sz1 = m*sizeof(void*), sz2 = m*szRow;
        void* buf = calloc(12 + sz1 + sz2, 1); // 12 ≡ headsize
        if(buf == NULL) return;
        int* bs = (int*)buf;
        bs[0] = m; bs[1] = n; bs[2] = szItem;
        buf = (char*)buf + 12;
        *a = (void**)buf;
        buf = (char*)buf + sz1; (*a)[0] = buf;
        for(int i = 1; i < m; i++){
          buf = (char*)buf + szRow;
          (*a)[i] = buf; // *(*a + i) = buf
        }
}
```



DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- How to use from main()

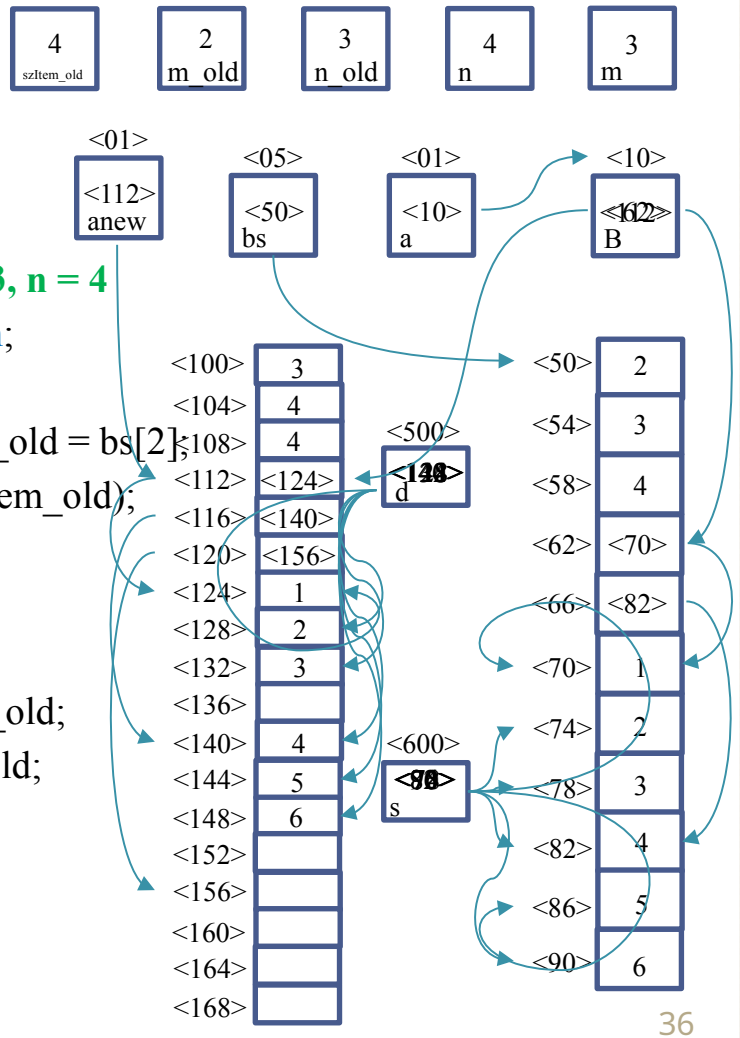
```
void main(){
    int d, c, **B; cin >> d >> c; // d = 2, c = 3
    alloc2D((void***)&B, d, c, sizeof(int));
    arr2D_Input(B);
}
void arr2D_Input(int* a[]){
    int m = nRow((void**)a), n = nCol((void**)a);
    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
            cin >> a[i][j];
}
int nRow(void** c){
    int* bs = (int*)((char*)c - 12);
    return bs[0];
}
int nCol(void** d){
    int* bs = (int*)((char*)d - 12);
    return bs[1];
}
```



DYNAMIC 2D ARRAY (CONVENIENT METHODS)

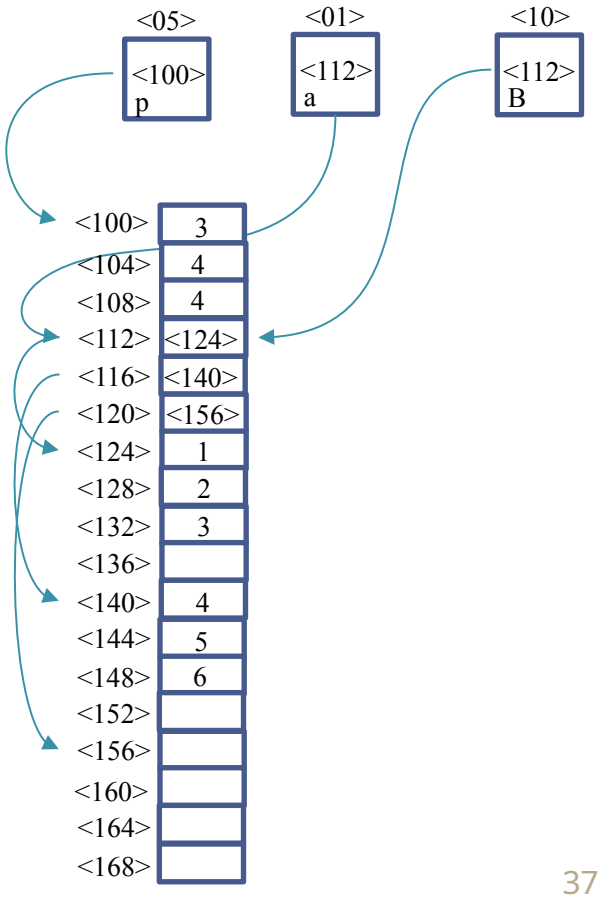
- How to use from main()

```
void main(){
    //...
    arr2D_Output(B); // easy
    resize2D((void***)&B, 3, 4);
}
void resize2D(void*** a, int m, int n){ // m = 3, n = 4
        if(*a == NULL || m <= 0 || n <= 0) return;
        int* bs = (int*)((char*)(*a) - 12);
        int m_old = bs[0], n_old = bs[1], szItem_old = bs[2];
        void** anew; alloc2D(&anew, m, n, szItem_old);
        if(anew == NULL) return;
        for(int i = 0; i < m_old && i < m; i++){
          for(int j = 0; j < n_old && j < n; j++){
            char* d = (char*)anew[i] + j * szItem_old;
            char* s = (char*)(*a)[i] + j * szItem_old;
            memmove(d, s, szItem_old);
          }
        }
        free2D(*a); *a = anew;
}
```



DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- How to use from main()
  - void main(){
    - //…
    - arr2D_Output(B); // easy
    - resize2D((void***)&B, 3, 4);
    - free2D((void**)B);
  - }
  - void free2D(void** a){
    - if(a != NULL){
      - void* p = (char*)a - 12;
      - free(p);
    - }
  - }



DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- Using template in C++:

```
template <class T>
void arr2D_alloc(T*** a, int m, int n){
  if(m <= 0 || n <= 0) return;
  int szRow=n*sizeof(T), sz1=m*sizeof(T*),
sz2=m*szRow;
  void* buf = calloc(8 + sz1 + sz2, 1);
  if(buf == NULL) return;
  int* bs = (int*)buf;
  bs[0] = m; bs[1] = n;
  buf = (char*)buf + 8;
  *a = (T**)buf;
  buf = (char*)buf + sz1; (*a)[0] = (T*)buf;
  for(int i = 1; i < m; i++){
    buf = (char*)buf + szRow;
    (*a)[i] = (T*)buf;
  }
}
```

```
void arr2D_alloc(void*** a, int m, int n, int szItem){
  if(m <= 0 || n <= 0 || szItem <= 0) return;
  int szRow=n*szItem, sz1=m*sizeof(void*), sz2=m*szRow;
  void* buf = calloc(12 + sz1 + sz2, 1); // 12 ≥ headsize
  if(buf == NULL) return;
  int* bs = (int*)buf;
  bs[0] = m; bs[1] = n; bs[2] = szItem;
  buf = (char*)buf + 12;
  *a = (void**)buf;
  buf = (char*)buf + sz1; (*a)[0] = buf;
  for(int i = 1; i < m; i++){
    buf = (char*)buf + szRow;
    (*a)[i] = buf; // *(*a + i) = buf
  }
}
```

DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- using template in C++:

```
template <class T>
int nRow(T** c){
  int* bs = (int*)((char*)c - 8);
  return bs[0];
}
template <class T>
int nCol(T** d){
  int* bs = (int*)((char*)d - 8);
  return bs[1];
}
template <class T>
void free2D(T** a){
  if(a != NULL){
    void* p = (char*)a - 8; free(p);
  }
}
```

```
int nRow(void** c){
  int* bs = (int*)((char*)c - 12);
  return bs[0];
}
int nCol(void** d){
  int* bs = (int*)((char*)d - 12);
  return bs[1];
}
void free2D(void** a){
  if(a != NULL){
    void* p = (char*)a - 12; free(p);
  }
}
```

DYNAMIC 2D ARRAY (CONVENIENT METHODS)

- using template in C++:

```
template <class T>
void resize2D(T*** a, int m, int n){
  if(*a == NULL || m <= 0 || n <= 0) return;
  int* bs = (int*)((char*)(*a) - 8);
  int m_old = bs[0], n_old = bs[1];
  T** anew; alloc2D(&anew, m, n);
  if(anew == NULL) return;
  for(int i = 0; i < m_old && i < m; i++){
    for(int j = 0; j < n_old && j < n; j++){

      anew[i][j] = (*a)[i][j]; // Overload '=' if T is not basic
      //(*a)[i][j] ⟷ *(*(*a+i) + j)
    }
  }
  free2D(*a); *a = anew;
}
```

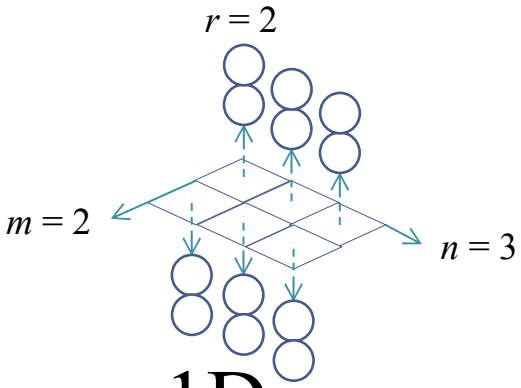
```
void resize2D(void*** a, int m, int n, int szItem){
  if(*a == NULL || m <= 0 || n <= 0) return;
  int* bs = (int*)((char*)(*a) - 12);
  int m_old = bs[0], n_old = bs[1], szItem_old = bs[2];
  void** anew; alloc2D(&anew, m, n, szItem_old);
  if(anew == NULL) return;
  for(int i = 0; i < m_old && i < m; i++){
    for(int j = 0; j < n_old && j < n; j++){
      char* d = (char*)anew[i] + j * szItem_old;
      char* s = (char*)(*a)[i] + j * szItem_old;
      memmove(d, s, szItem_old);
    }
  }
  free2D(*a); *a = anew;
}
```

DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

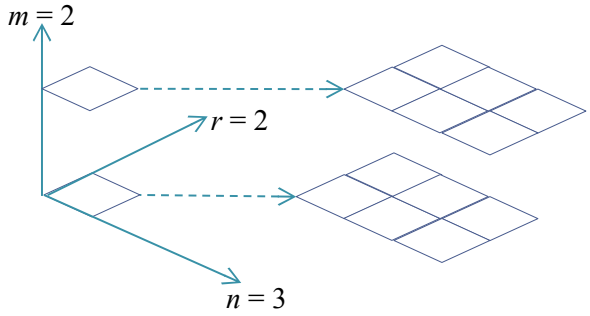
- Rarely used as with 1D/2D arrays
- Processing can be made based from operations of 1D or 2D array
- Still can build pure convenient functions for 3D array
- 3D array has a number of rows, columns and high/depth (m $\triangleq$ n $\triangleq$ r)
- There are two methods of building convenient functions based on 1D/2D array
  - Similar to 2D array (m $\triangleq$ n), and each element is a 1D array having r elements
  - Similar to 1D array having m elements, and each element is a 2D array (n $\triangleq$ r)

DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- 3D array: be a 2D array (m ≡ n), and each element is a 1D array having r elements

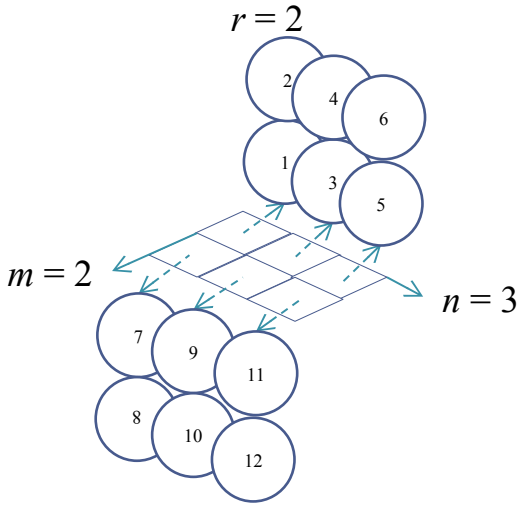


- 3D array: be a 1D array having m elements, and each element is a matrix (n ≡ r)

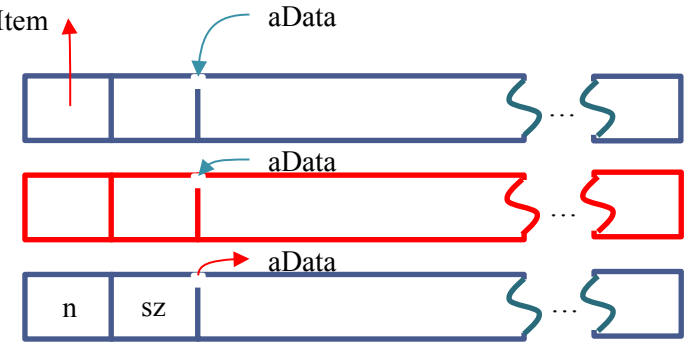


DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The first solution: ex m = r = 2 and n = 3

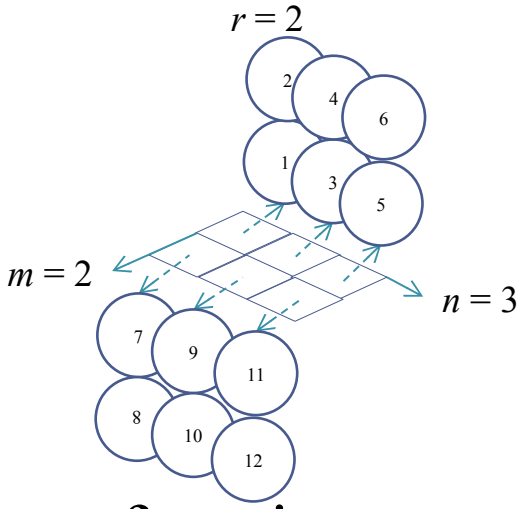


  - Remind some functions
    - int arrSize(void* aData)
    - void arrFree(void* aData)
    - void* arrInit(int n, int sz)

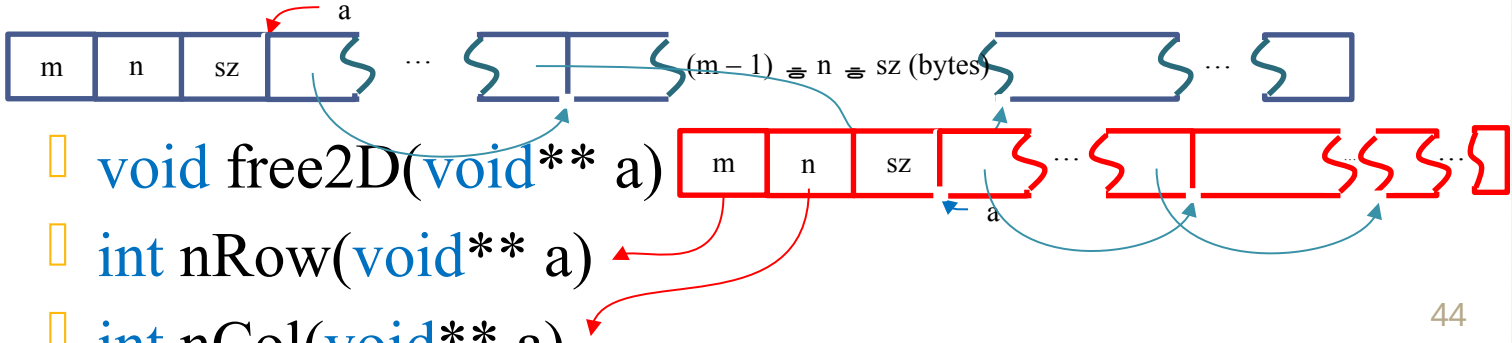


DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The first solution: ex m = r = 2 and n = 3



  - Remind some functions
    - void alloc2D(void*** a, int m, int n, int sz)



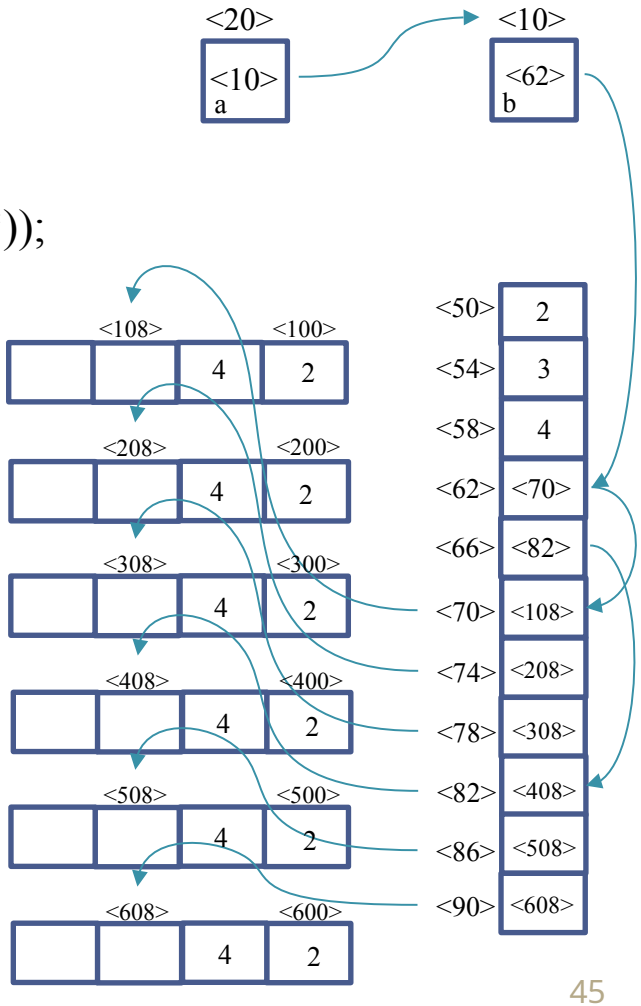
    - void free2D(void** a)
    - int nRow(void** a)
    - int nCol(void** a)

DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The first solution: ex m = r = 2 and n = 3
  - float*** alloc3D(int m, int n, int r){

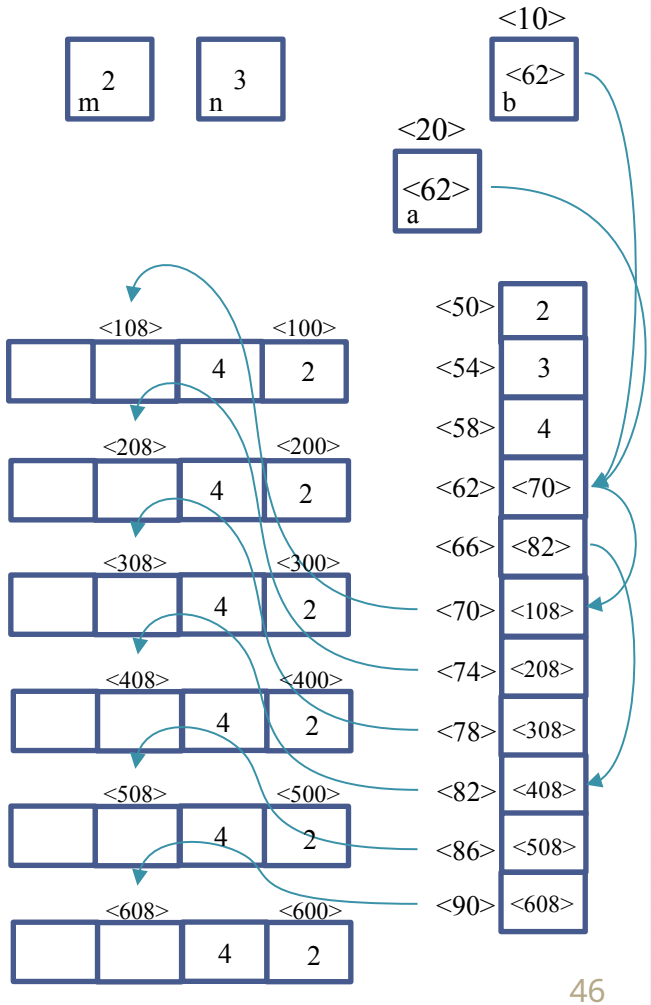
```
if(m <= 0 || n <= 0 || r <= 0) return NULL;
float*** b; int err = 0;
alloc2D((void***)(&b), m, n, sizeof(float*));
if(b == NULL) return NULL;
for(int i = 0; i < m; i++){
    for(int j = 0; j < n; j++){
        b[i][j] = (float*)arrInit(r, sizeof(float));
        if(b[i][j] == NULL) {err = 1;}
    }
}
if(err == 1){
    free3D(b);
    b = NULL;
}
return b;
```

  - }



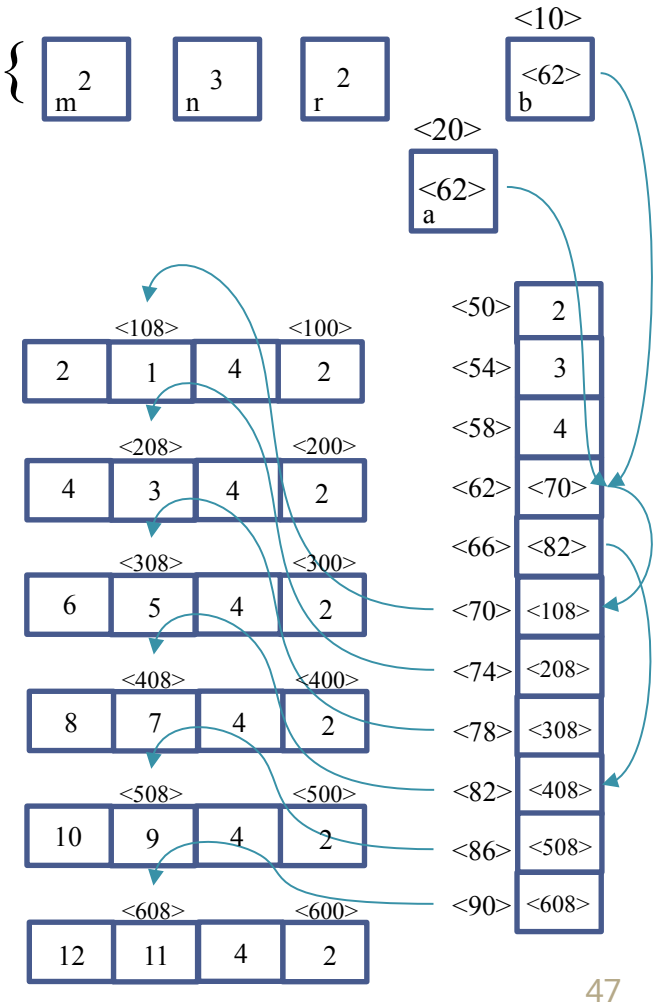
DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The first solution: ex m = r = 2 and n = 3
  - void free3D(float*** a){
    - if(a != NULL){
      - int m = nRow((void**)a);
      - int n = nCol((void**)a);
      - for(int i = 0; i < m; i++){
        - for(int j = 0; j < n; j++){
          - arrFree(a[i][j]);
        - }
      - }
      - free2D((void**)a);
    - }
  - }



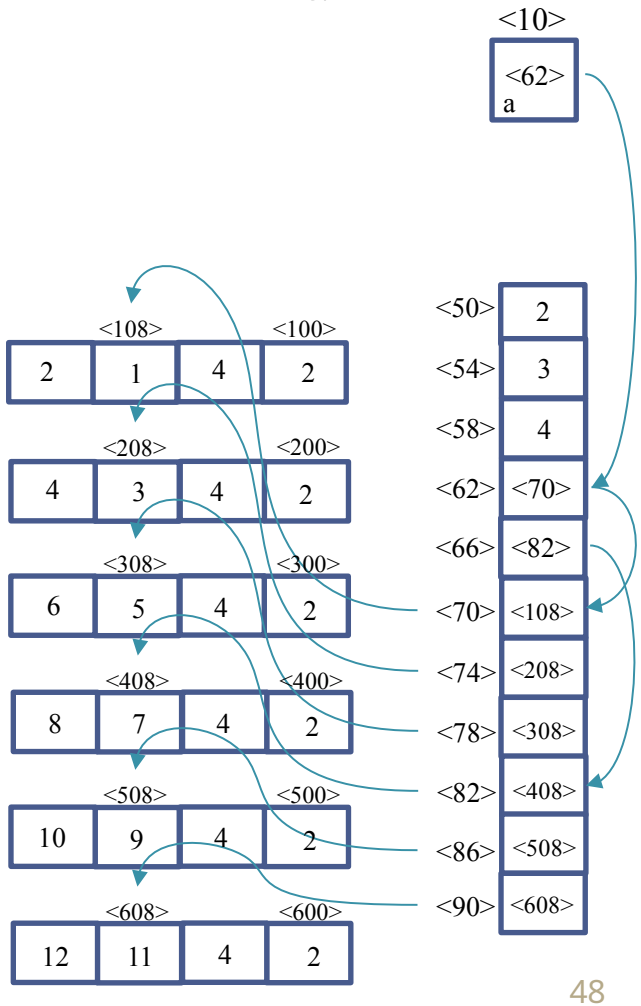
DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The first solution: ex m = r = 2 and n = 3
  - void arr3D_input(float*** a){
    - int m = nRow((void**)a);
    - int n = nCol((void**)a);
    - int r = arrSize(a[0][0]);
    - for(int i = 0; i < m; i++){
      - for(int j = 0; j < n; j++){
        - for(int k = 0; k < r; k++){
          - cin >> a[i][j][k];
          - //cin >>*(*(*(a+i)+j)+k);
        - }
      - }
    - }
  - }



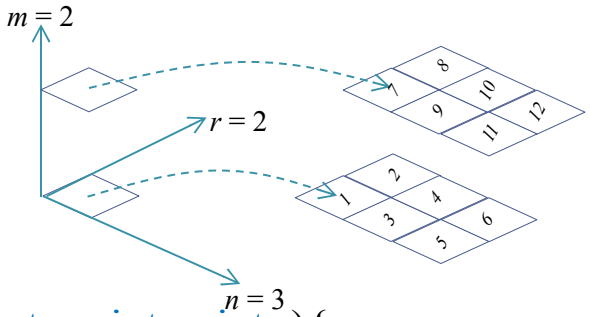
DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The first solution: ex m = r = 2 and n = 3
  - void main(){
    - float*** a = alloc3D(2, 3, 2);
    - arr3D_input(a);
    - arr3D_output(a);
    - free3D(a);
  - }

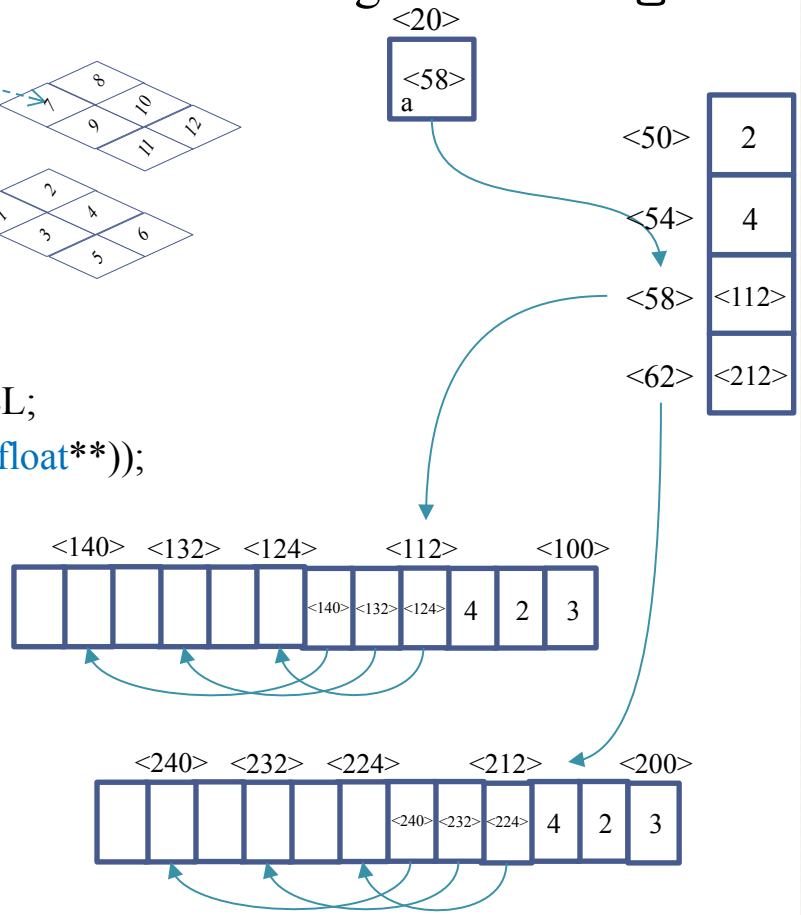


DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The second solution: 1D array with m element being a matrix n ⨯ r.

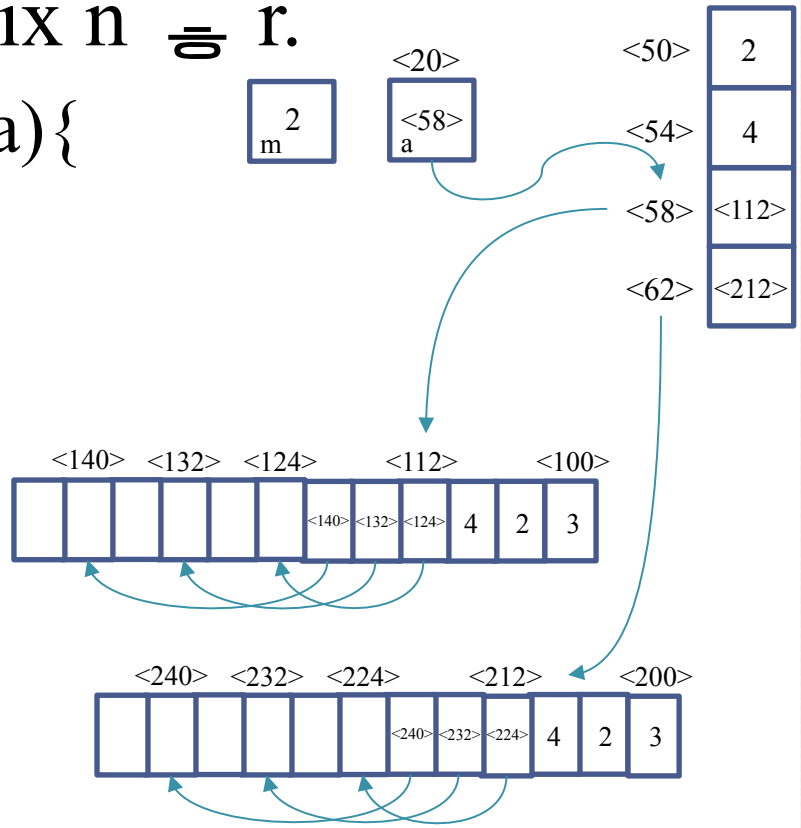



```
float*** alloc3D(int m, int n, int r){
    if(m <= 0 || n <= 0 || r <= 0) return NULL;
    float*** a = (float***)arrInit(m, sizeof(float**));
    if(a == NULL) return NULL;
    int error = 0;
    for(int i = 0; i < m; i++){
        alloc2D((void***)&a[i], n, r, sizeof(float));
        if(a[i] == NULL){ error = 1; break; }
    }
    if(error == 1){ free3D(a); a = NULL;}
    return a;
}
```



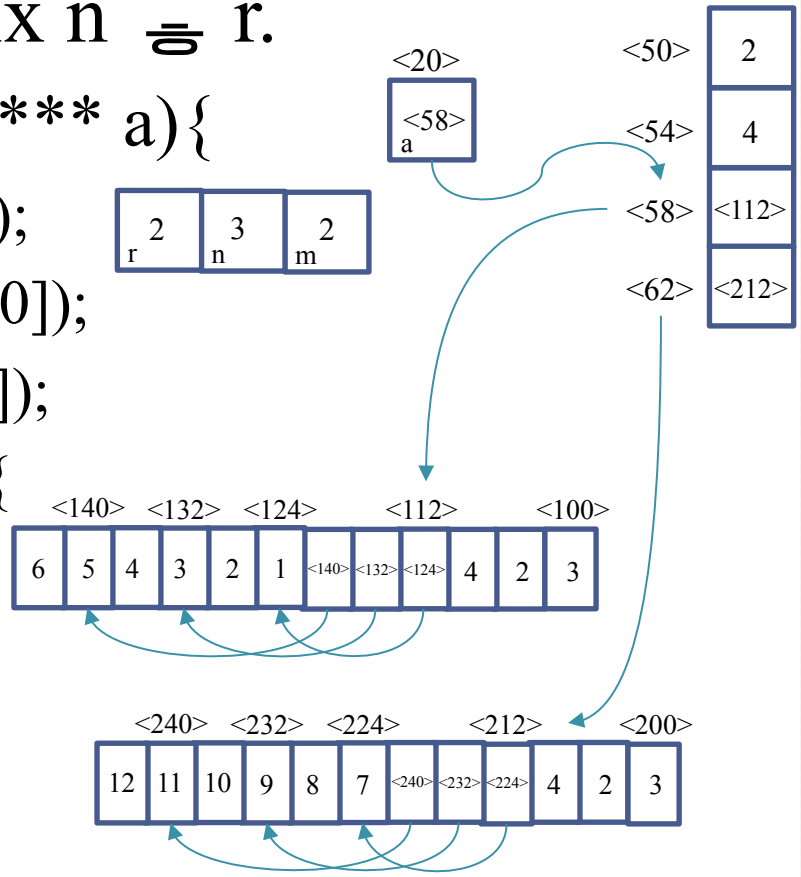
DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The second solution: 1D array with m element being a matrix n ⨯ r.
  - void free3D(float*** a){
    - if(a != NULL){
      - int m = arrSize((void*)a);
      - for(int i = 0; i < m; i++){
        - if(a[i] != NULL){
          - free2D((void**)a[i]);
        - }
      - }
      - arrFree((void*)a);
    - }
  - }



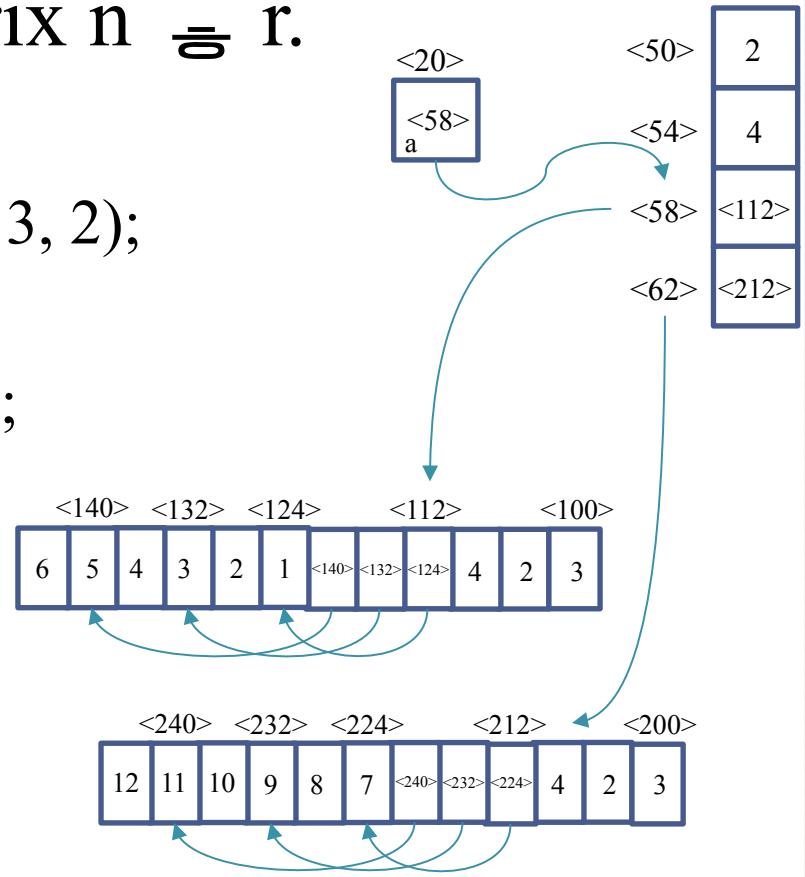
DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The second solution: 1D array with m element being a matrix n ₹ r.
  - void arr3D_input(float*** a){
    - int m = arrSize((void*)a);
    - int n = nRow((void**)a[0]);
    - int r = nCol((void**)a[0]);
    - for(int i = 0; i < m; i++){
      - for(int j = 0; j < n; j++)
        - for(int k = 0; k < r; k++)
          - cin>>a[i][j][k];
    - }
  - }



DYNAMIC MULTI-DIMENSIONAL ARRAY (3D ARRAY)

- The second solution: 1D array with m element being a matrix n ₌ r.
  - void main(){
    - float*** a = alloc3D(2, 3, 2);
    - arr3D_input(a);
    - arr3D_output(a);
    - free3D(a);
  - }



DYNAMIC MULTI-DIMENSIONAL ARRAY (FREE MEMORY)

- What if source code has any memory-error:
  - Inefficiently using memory
  - Easy to cause interruption when running
  - Ram may be run out of if any program using this source code runs again and again
  - More serious if any memory-error program is running in service provider (server)
  - Data which are not free may be sensitive, such as password or financial key…
  - The bytes being free are assigned the value of 0xDD