POINTER

PROGRAMMING TECHNIQUES

ADVISOR: Trương Toàn Thịnh

CONTENTS

- Structure data
- Structural memory
- Operator [] for structural variable
- Structural array
- Enhanced techniques

STRUCTURAL DATA

- In practice, data follow another structure
- Use keyword struct combined with typedef to define a new datatype
- Use operator ‘.’ or ‘->’ to access to all members of a structure
- Use operator sizeof combined with pragma pack(1) to exactly determine the size of a new datatype
- Use keyword union if wanting some members of a structure to share a common memory

STRUCTURAL DATA (SIZE)

- Example (need using pragma pack(1))
  - typedef struct {
    - long offset;
    - union{ unsigned long cw; unsigned char cb[4];};
    - char mb;
  - }HeadStruct;
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - typedef struct{
    - short nVers;
    - Point* Vers;
  - } PolygonP;

```
void main(){
  HeadStruct aDat;
  HeadStruct* pDat = &aDat;
}
```

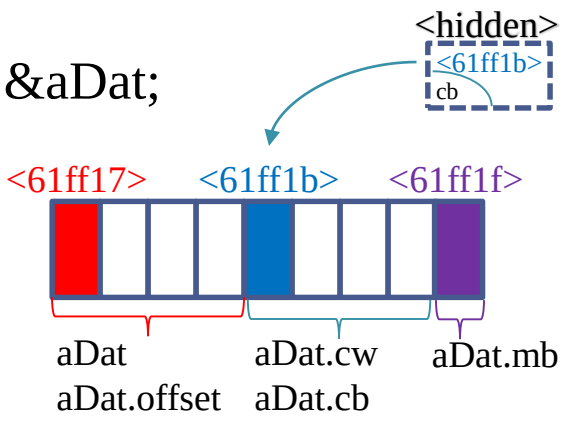
sizeof(aDat): 9
sizeof(pDat): 4
sizeof(HeadStruct): 9
sizeof(Point): 8
sizeof(Polygon): 10
sizeof(PolygonP): 6

STRUCTURAL DATA (MEMBER ADDRESS)

- Example (need using pragma pack(1))
  - typedef struct {
    - long offset;
    - union{ unsigned long cw; unsigned char cb[4];};
    - char mb;
  - }HeadStruct;
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - typedef struct{
    - short nVers;
    - Point* Vers;
  - } PolygonP;

```
void main(){
  HeadStruct aDat;
  HeadStruct* pDat = &aDat;
}
```

&aDat: 61ff17
&aDat.offset: 61ff17
&aDat.cw: 61ff1b
&aDat.cb: 61ff1b
&aDat.mb: 61ff1f

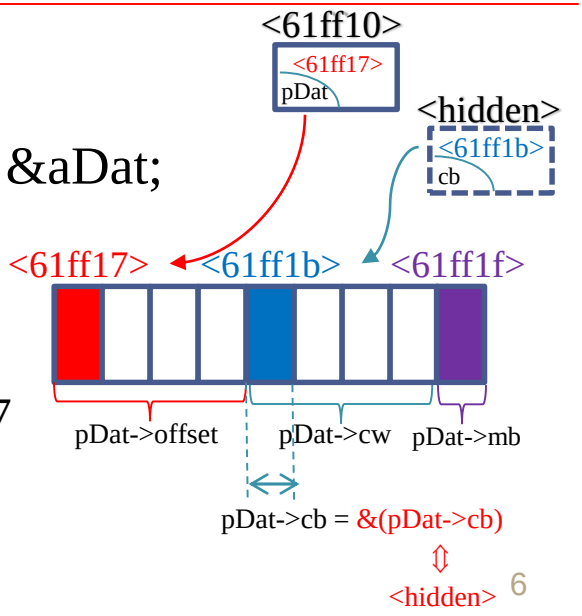


STRUCTURAL DATA (MEMBER ADDRESS)

- Example (need using pragma pack(1))
  - typedef struct {
    - long offset;
    - union{ unsigned long cw; unsigned char cb[4];};
    - char mb;
  - }HeadStruct;
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - typedef struct{
    - short nVers;
    - Point* Vers;
  - } PolygonP;

```
void main(){
  HeadStruct aDat;
  HeadStruct* pDat = &aDat;
}
```

&pDat: 61ff10
pDat: 61ff17
&(pDat->offset): 61ff17
pDat->cb: 61ff1b
&(pDat->cw): 61ff1b
&(pDat->mb): 61ff1f

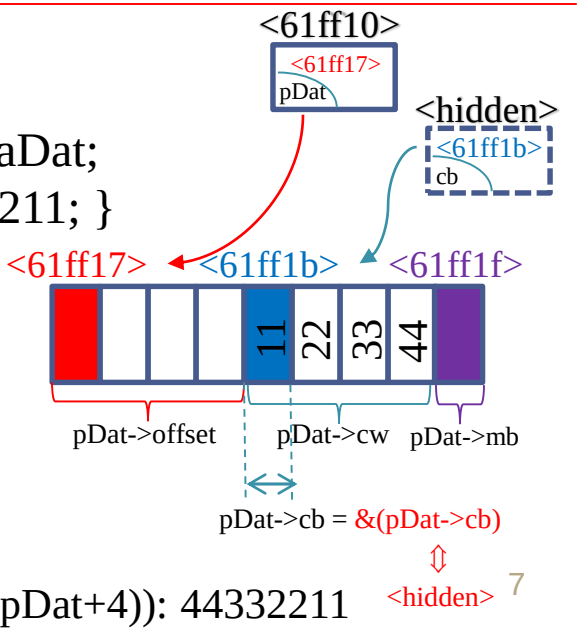


STRUCTURAL DATA (UNION VALUE)

- Example (need using pragma pack(1))
  - typedef struct {
    - long offset;
    - union{ unsigned long cw; unsigned char cb[4];};
    - char mb;
  - }HeadStruct;
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - typedef struct{
    - short nVers;
    - Point* Vers;
  - } PolygonP;

```
void main(){
  HeadStruct aDat;
  HeadStruct* pDat = &aDat;
  (*pDat).cw = 0x44332211; }
```

```
*(pDat->cb+0): 11
pDat->cb[1]: 22
*(pDat->cb+2): 33
pDat->cb[3]: 44
pDat->cw: 44332211
*(pDat+4): 44332211
*((unsigned long*)((char*)pDat+4)): 44332211
```



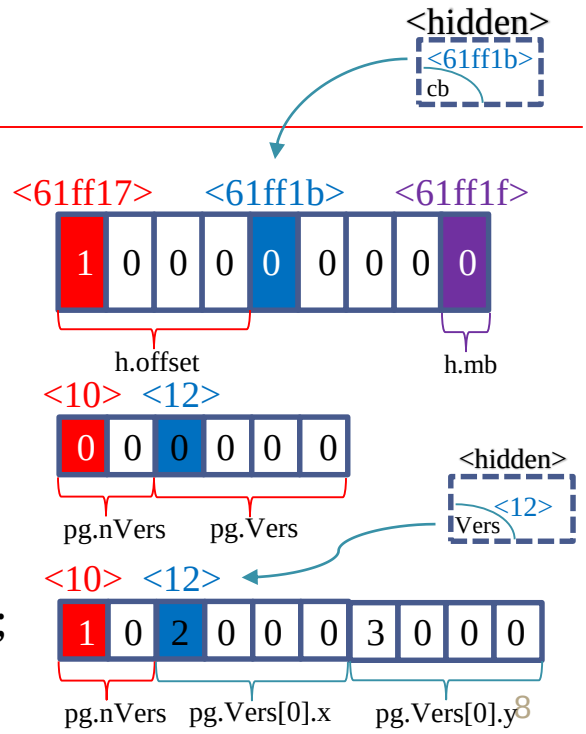
STRUCTURAL DATA (INITIALIZATION TECHNIQUE)

- Example (need using pragma pack(1))
  - typedef struct {
    - long offset;
    - union{ unsigned long cw; unsigned char cb[4];};
    - char mb;
  - }HeadStruct;
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - typedef struct{
    - short nVers;
    - Point* Vers;
  - } PolygonP;

HeadStruct h = {1};

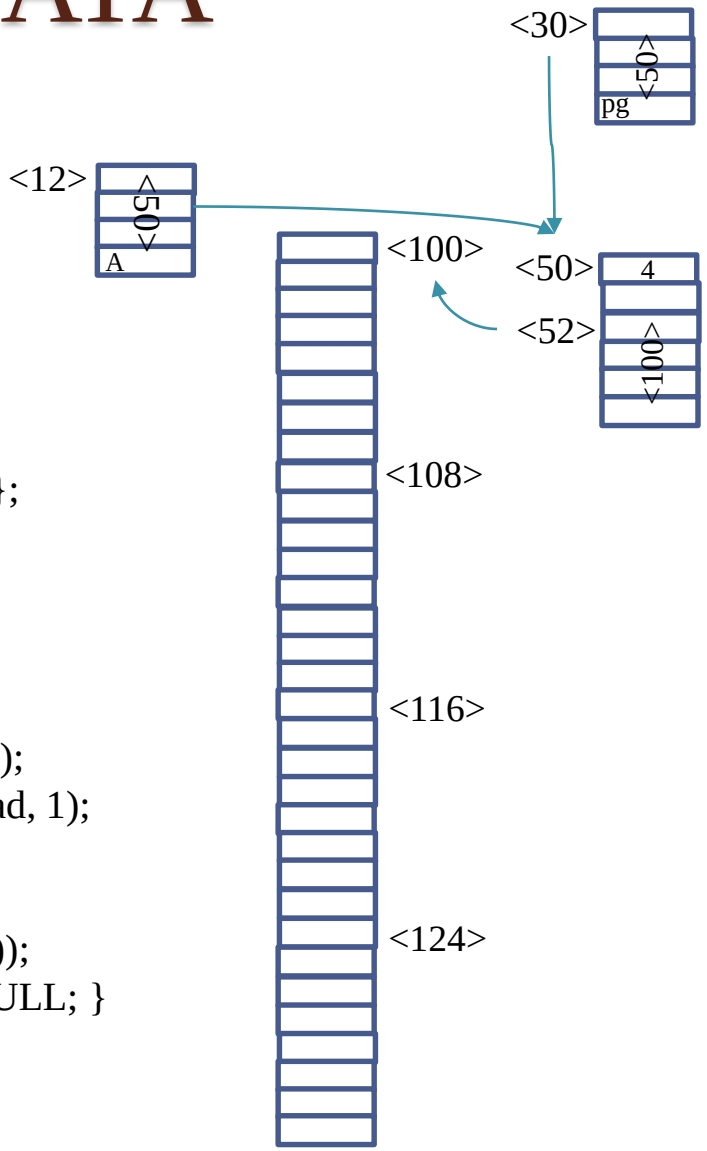
PolygonP pg = {0};

Polygon pg = {1, {2, 3}};



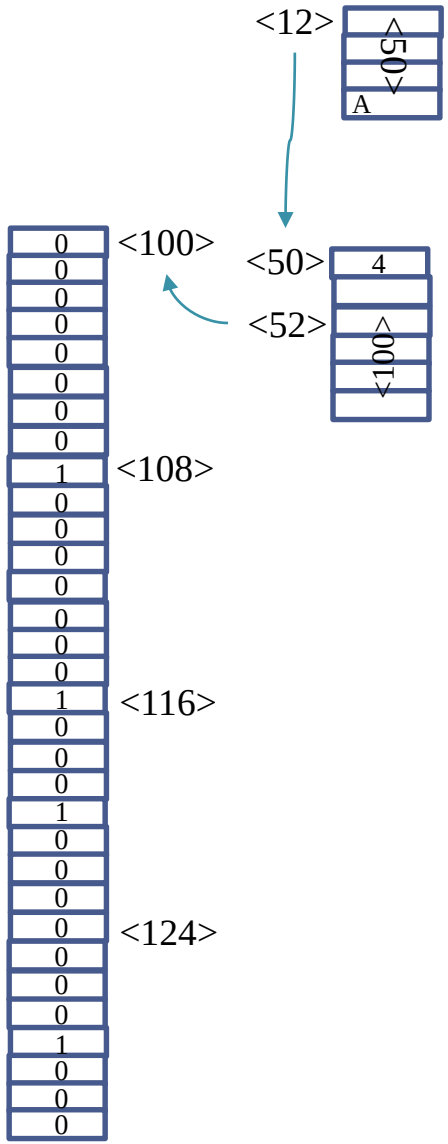
STRUCTURAL DATA

- Example of PolygonP (method 1)
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers; Point* Vers;
  - } PolygonP;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - PolygonP* A = PgAllocA(4);
  - }
  - PolygonP* PgAllocA(int n){
    - if(n < 0) return NULL;
    - int szHead = sizeof(short) + sizeof(Point*);
    - PolygonP* pg = (PolygonP*)calloc(szHead, 1);
    - if(pg == NULL) return NULL;
    - pg->nVers = n;
    - pg->Vers = (Point*)calloc(n, sizeof(Point));
    - if(pg->Vers == NULL){ free(pg); pg = NULL; }
    - return pg;
  - }



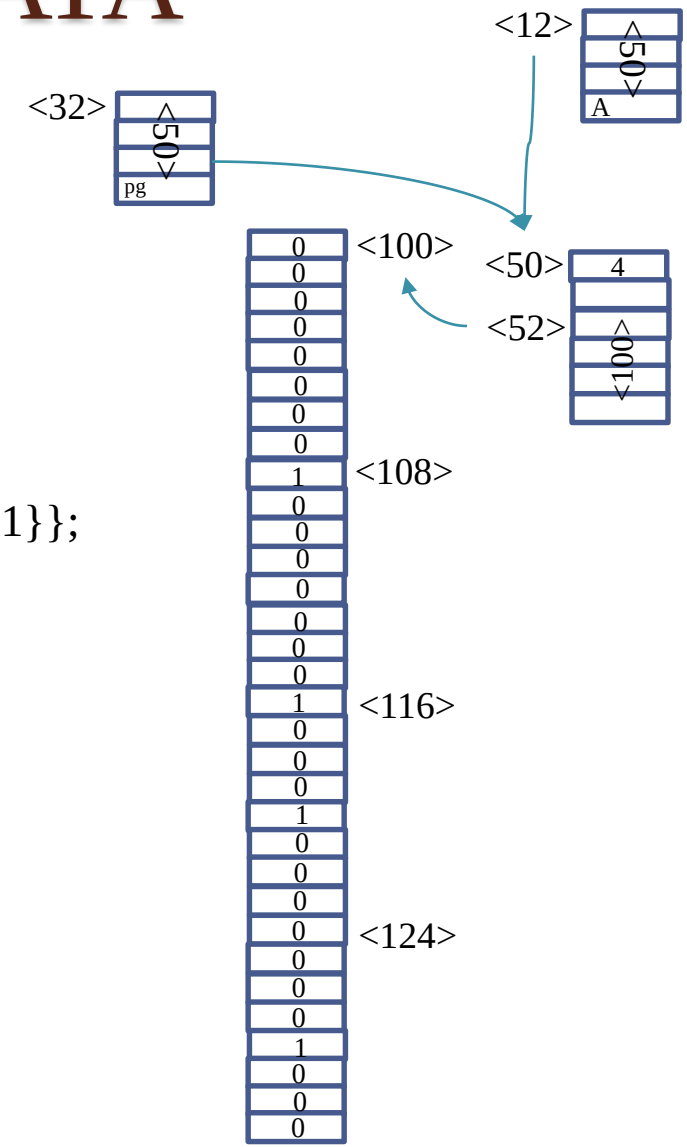
STRUCTURAL DATA

- Example of PolygonP (method 1)
  - typedef struct {
    - long x, y;
  - } Point;
  - typedef struct{
    - short nVers; Point* Vers;
  - } PolygonP;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - PolygonP* A = PgAllocA(4);
    - if(A != NULL){
      - for(int i = 0; i < 4; i++){
        - A->Vers[i] = P[i];
      - }
    - }
  - }



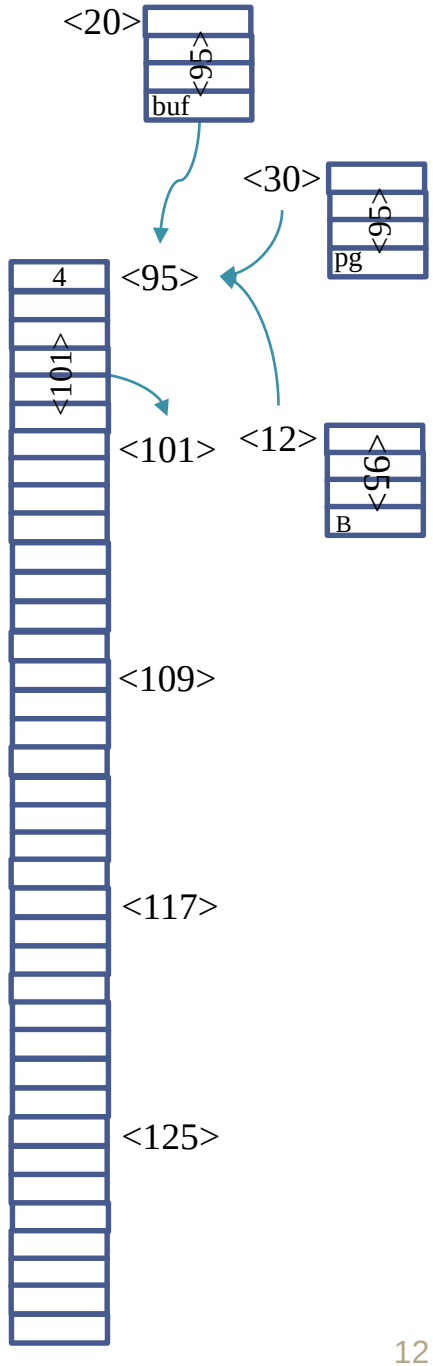
STRUCTURAL DATA

- Example of PolygonP (method 1)
  - typedef struct { long x, y; } Point;
  - typedef struct{
    - short nVers; Point* Vers;
  - } PolygonP;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - PolygonP* A = PgAllocA(4);
    - if(A != NULL){
      - for(int i = 0; i < 4; i++) A->Vers[i] = P[i];
      - pgFreeA(A);
    - }
  - }
  - void pgFreeA(PolygonP* pg){
    - if(pg != NULL){
      - if(pg->Vers != NULL) free(pg->Vers);
      - free(pg);
    - }
  - }



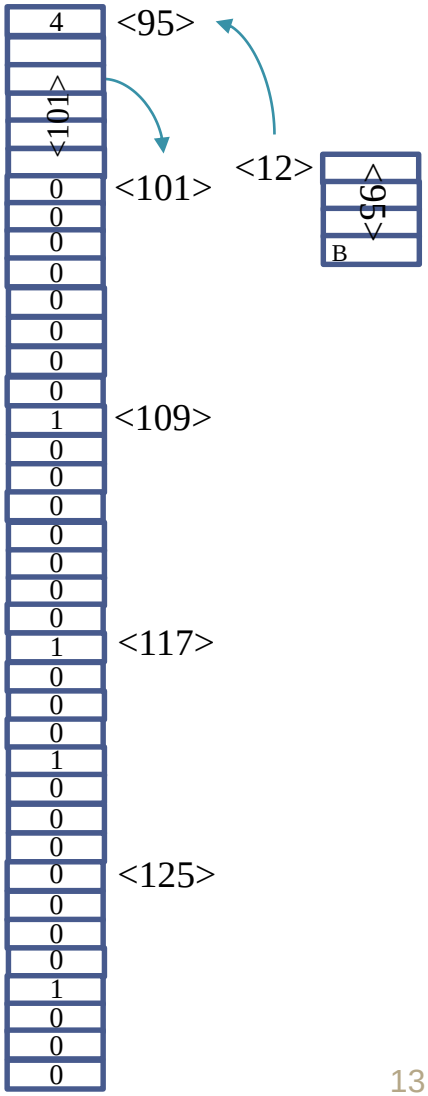
STRUCTURAL DATA

- Example of PolygonP (method 2)
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers; Point* Vers;
  - } PolygonP;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - PolygonP* B = PgAllocB(4);
  - }
  - PolygonP* PgAllocB(int n){
    - if(n < 0) return NULL;
    - int szHead = sizeof(short) + sizeof(Point*);
    - int szData = n * sizeof(Point);
    - void* buf = calloc(szHead + szData, 1);
    - if(buf == NULL) return NULL;
    - PolygonP* pg = (Polygon*)buf;
    - pg->nVers = n;
    - pg->Vers = (Point*)((char*)buf + szHead);
    - return pg;
  - }



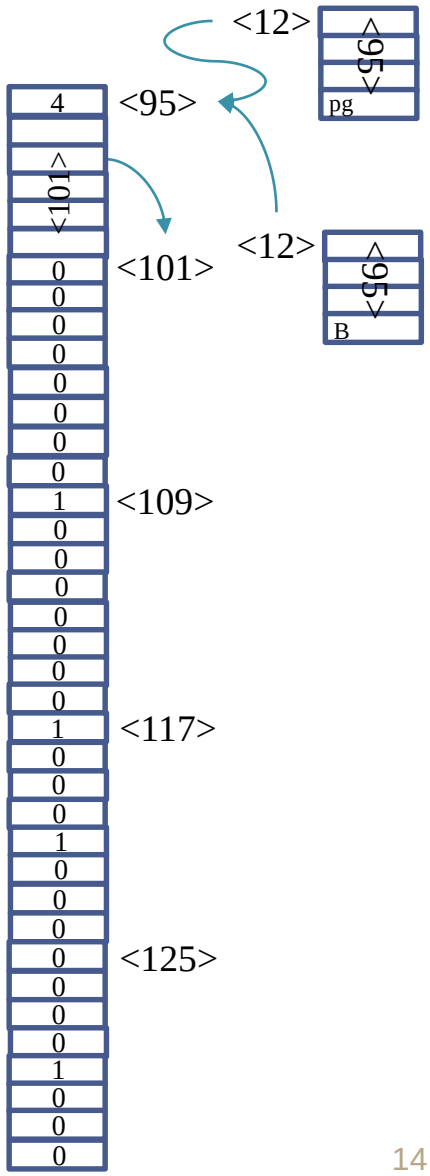
STRUCTURAL DATA

- Example of PolygonP (method 2)
  - typedef struct {
    - long x, y;
  - } Point;
  - typedef struct{
    - short nVers; Point* Vers;
  - } PolygonP;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - PolygonP* B = PgAllocB(4);
    - if(B != NULL){
      - for(int i = 0; i < 4; i++){
        - B->Vers[i] = P[i];
      - }
    - }
  - }



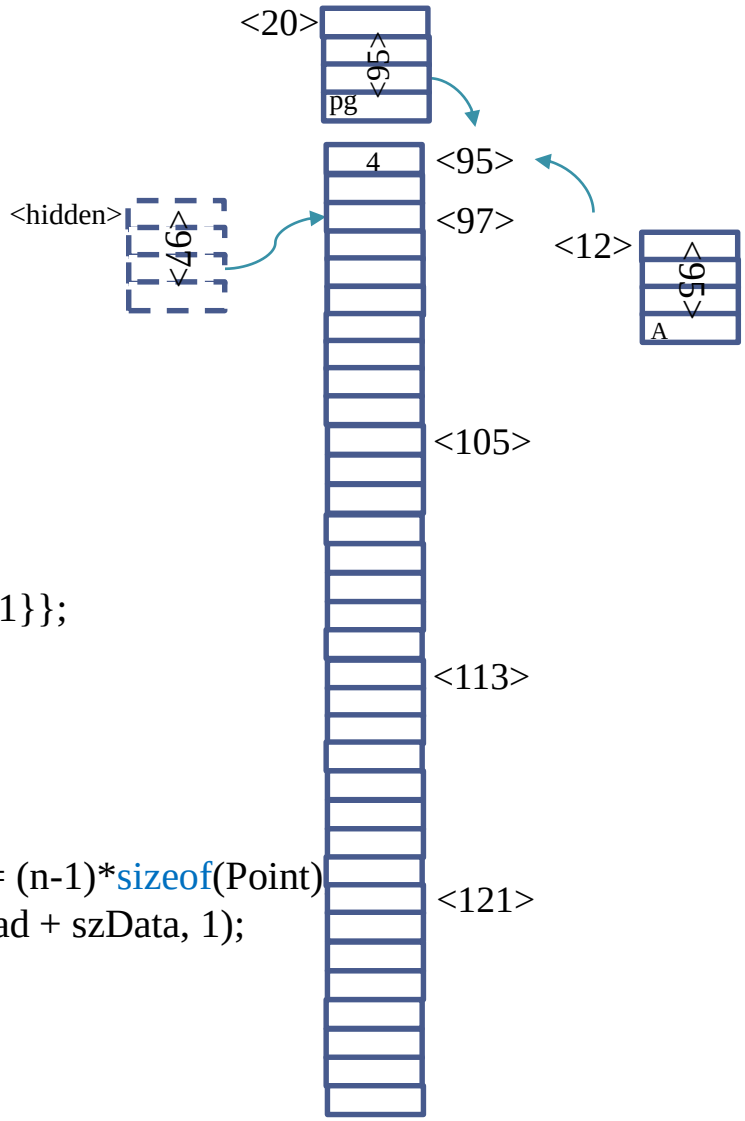
STRUCTURAL DATA

- Example of PolygonP (method 2)
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers; Point* Vers;
  - } PolygonP;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - PolygonP* B = PgAllocB(4);
    - if(B != NULL){
      - for(int i = 0; i < 4; i++) B->Vers[i] = P[i];
      - pgFreeB(B);
    - }
  - }
  - void pgFreeB(PolygonP* pg){
    - if(pg != NULL) free(pg);
  - }



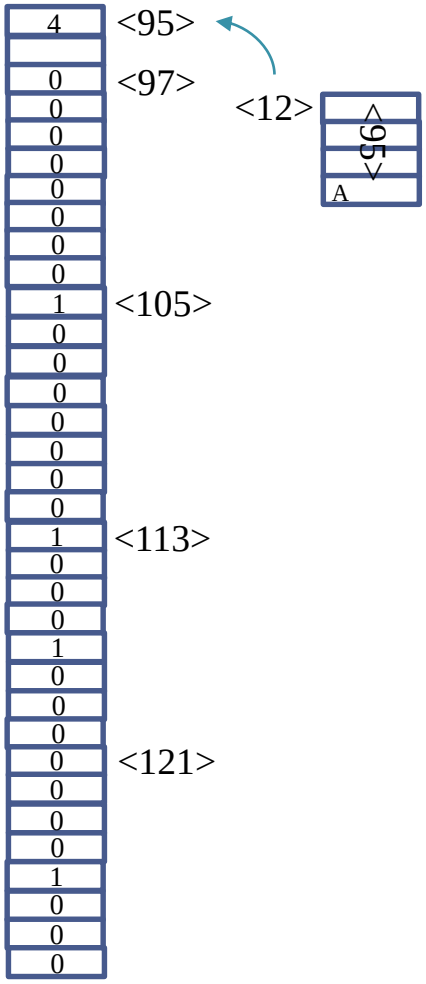
STRUCTURAL DATA

- Example of Polygon
  - typedef struct {
    - long x, y;
  - } Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - Polygon* A = PgAlloc(4);
  - }
  - Polygon* PgAlloc(int n){
    - if(n < 0) return NULL;
    - int szHead = sizeof(Polygon), szData = (n-1)*sizeof(Point);
    - Polygon* pg = (Polygon*)calloc(szHead + szData, 1);
    - if(pg == NULL) return NULL;
    - pg->nVers = n;
    - return pg;
  - }



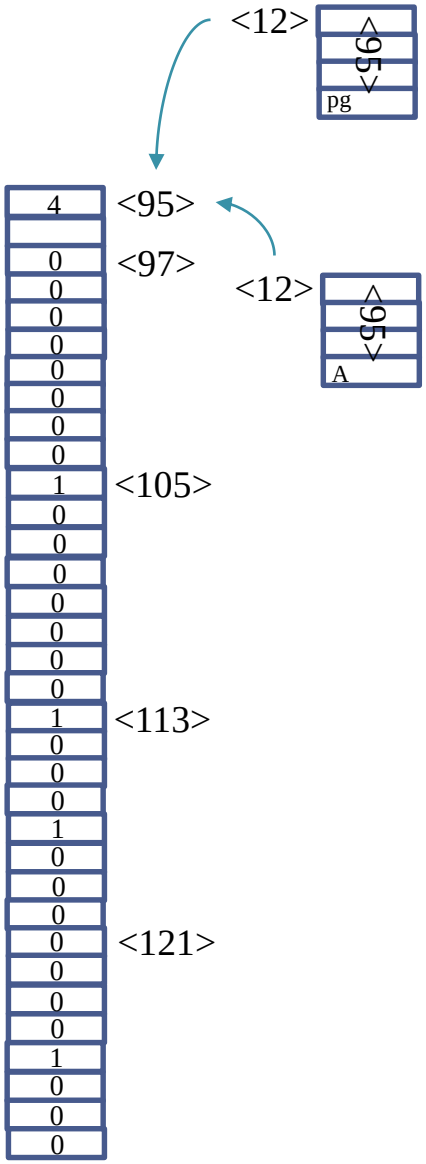
STRUCTURAL DATA

- Example of Polygon
  - typedef struct {
    - long x, y;
  - } Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - Polygon* A = PgAlloc(4);
    - if(A != NULL){
    - for(int i = 0; i < A->nVers; i++)
      - A->Vers[i] = P[i];
  - }



STRUCTURAL DATA

- Example of Polygon
  - typedef struct { long x, y;} Point;
  - typedef struct{
    - short nVers;
    - Point Vers[1];
  - } Polygon;
  - void main(){
    - Point P[] = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
    - Polygon* A = PgAlloc(4);
    - if(A != NULL){
    - for(int i = 0; i < A->nVers; i++)
      - A->Vers[i] = P[i];
    - pgFree(A);
  - }
  - void pgFree(Polygon* pg){
    - if(pg != NULL) free(pg);
  - }

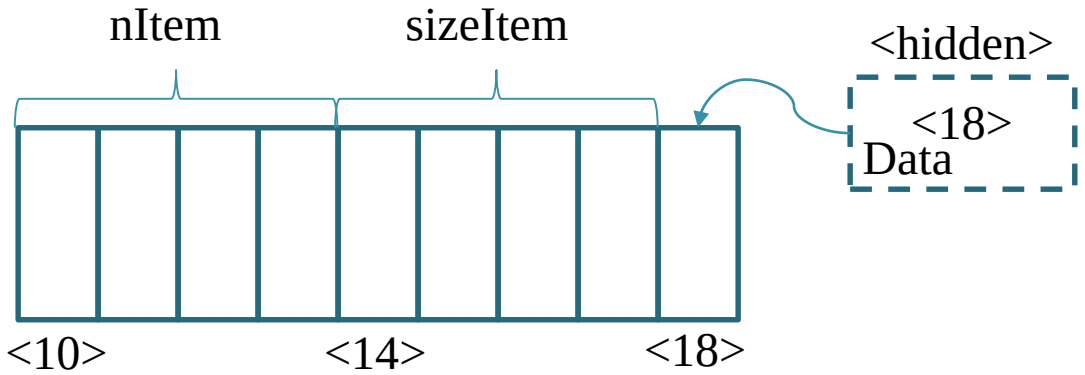


STRUCTURAL MEMORY

- Use structural variable to replace the computation on memory address
- Help source-code be abstract, easy-to-read and easy-to-maintain
- Program is more efficient due to without computation on memory address
- Apply structural variables for the functions of 1D/2D

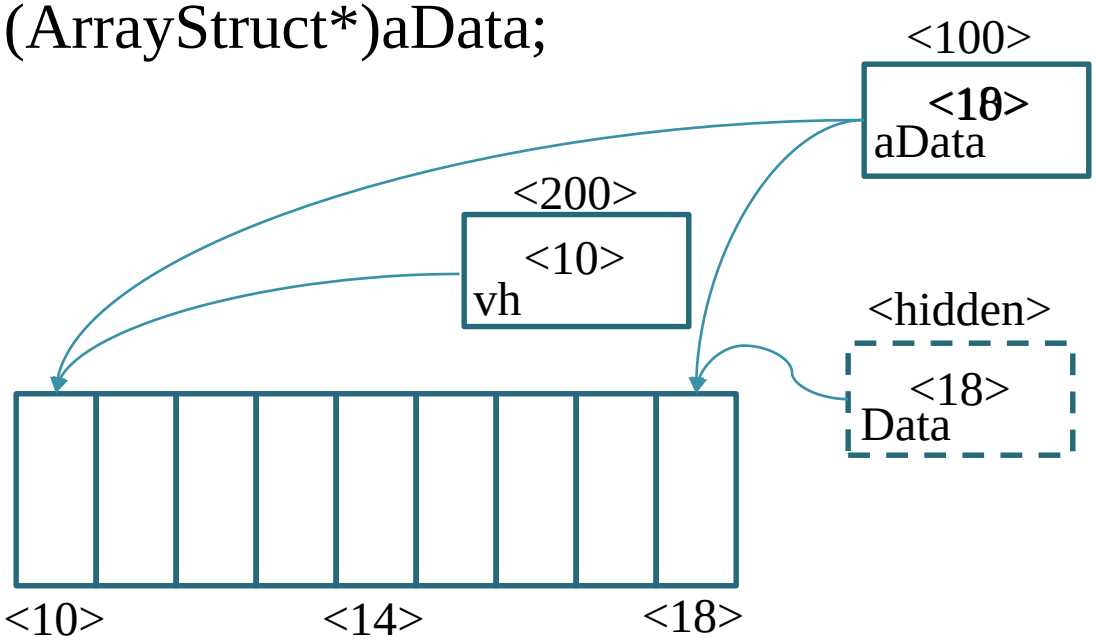
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - typedef struct {
    - int nItem, sizeItem;
    - char Data[1];
  - }ArrayStruct;
  - static int headSize = sizeof(int) + sizeof(int);



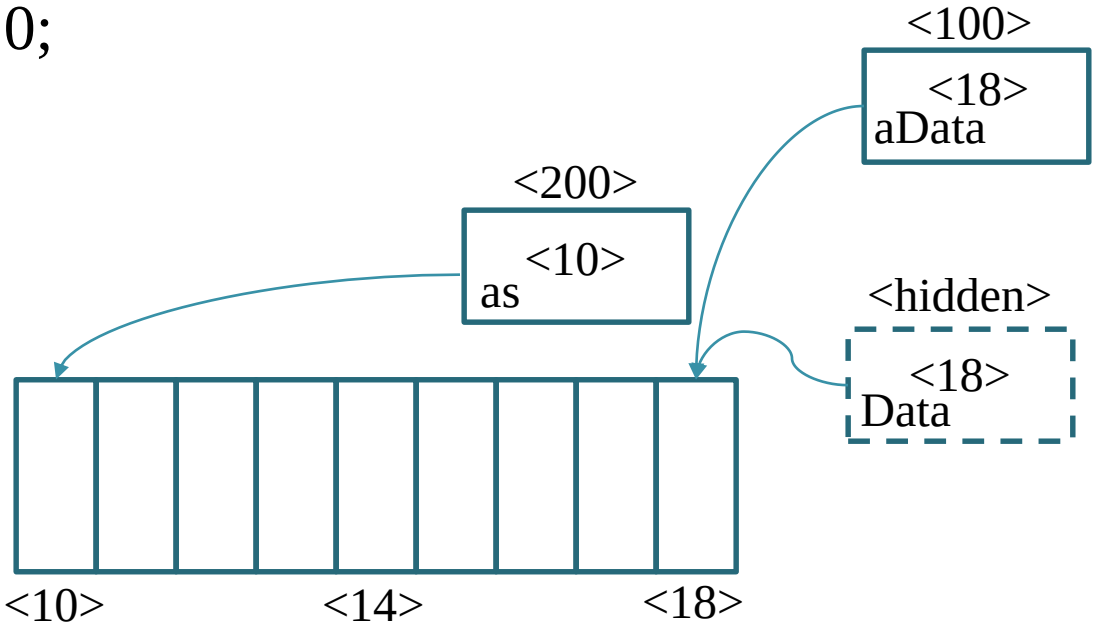
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - ArrayStruct* StructOf(void* aData){
    - if(aData != NULL)
      - aData = (char*)aData - headSize;
    - return (ArrayStruct*)aData;
  - }



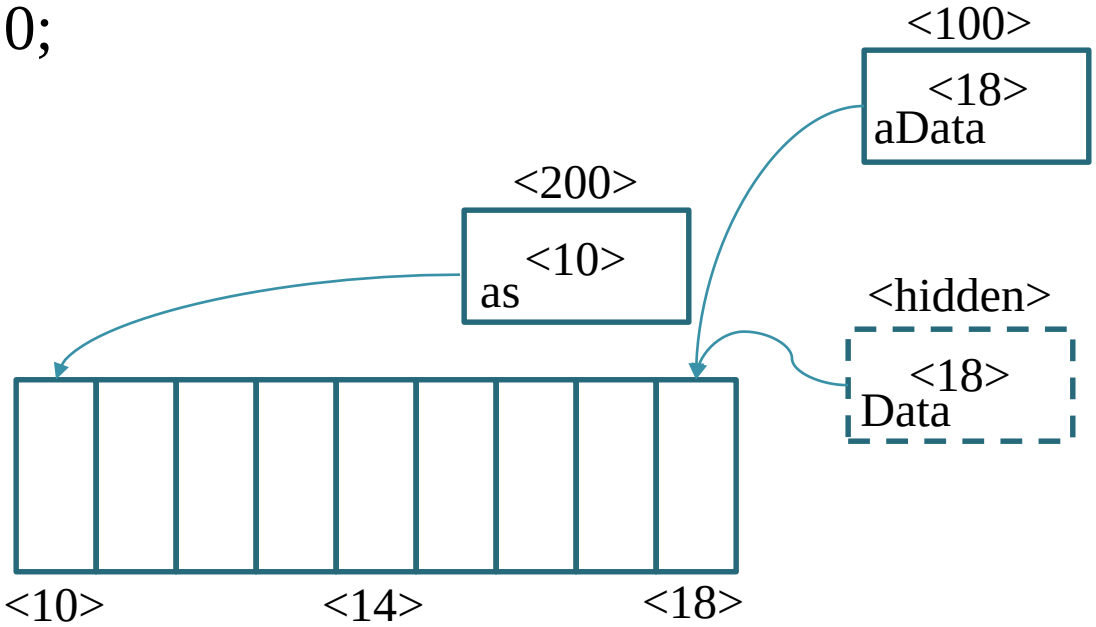
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - int arrSize(void* aData){
    - ArrayStruct* as = StructOf(aData);
    - if(as != NULL) return as->nItem;
    - return 0;
  - }



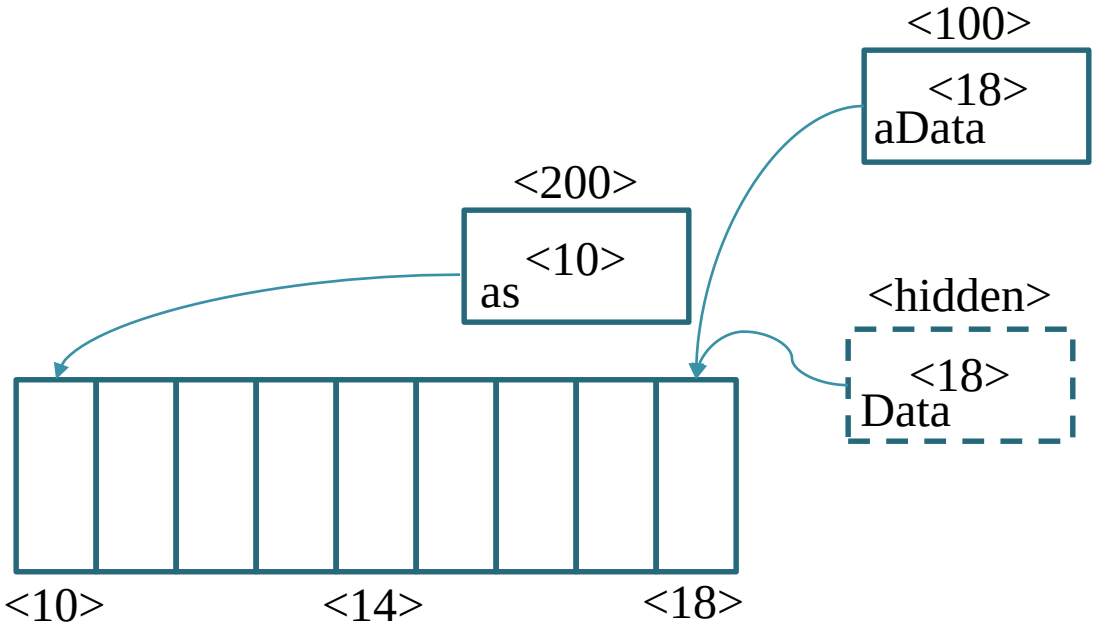
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - int arrItemSize(void* aData){
    - ArrayStruct* as = StructOf(aData);
    - if(as != NULL) return as->sizeItem;
    - return 0;
  - }



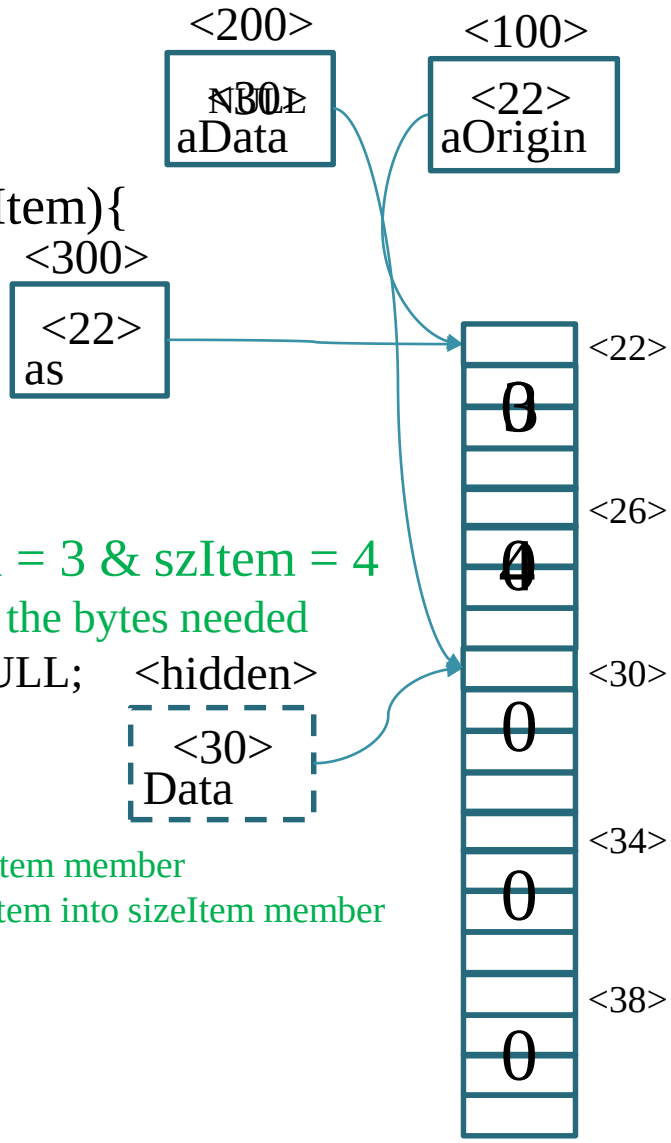
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - void arrFree(void* aData){
    - ArrayStruct* as = StructOf(aData);
    - if(as != NULL) free(as);
  - }



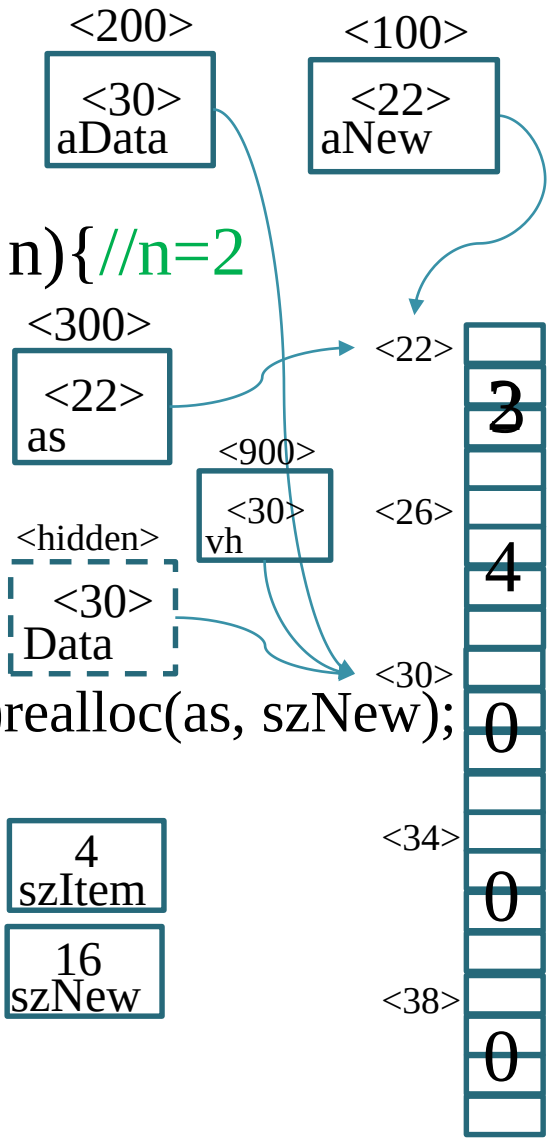
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - static int memSize(int nItem, int sizeItem){
    - if(sizeItem < 0) sizeItem = -sizeItem;
    - if(sizeItem == 0) sizeItem = 1;
    - if(nItem < 0) nItem = -nItem;
    - return headSize + nItem * sizeItem;
  - }
  - void* arrInit(int n, int szItem){//ex: n = 3 & szItem = 4
    - int sz = memSize(n, szItem); // Compute the bytes needed
    - void* aOrigin = malloc(sz), *aData = NULL;
    - if(aOrigin != NULL){
      - ArrayStruct* as = (ArrayStruct*)aOrigin;
      - memset(aOrigin, 0, sz);
      - as->nItem = n; // Assign value of nItem into nItem member
      - as->sizeItem = szItem; // Assign value of sizeItem into sizeItem member
      - aData = as->Data;
    - }
    - return aData;
  - }



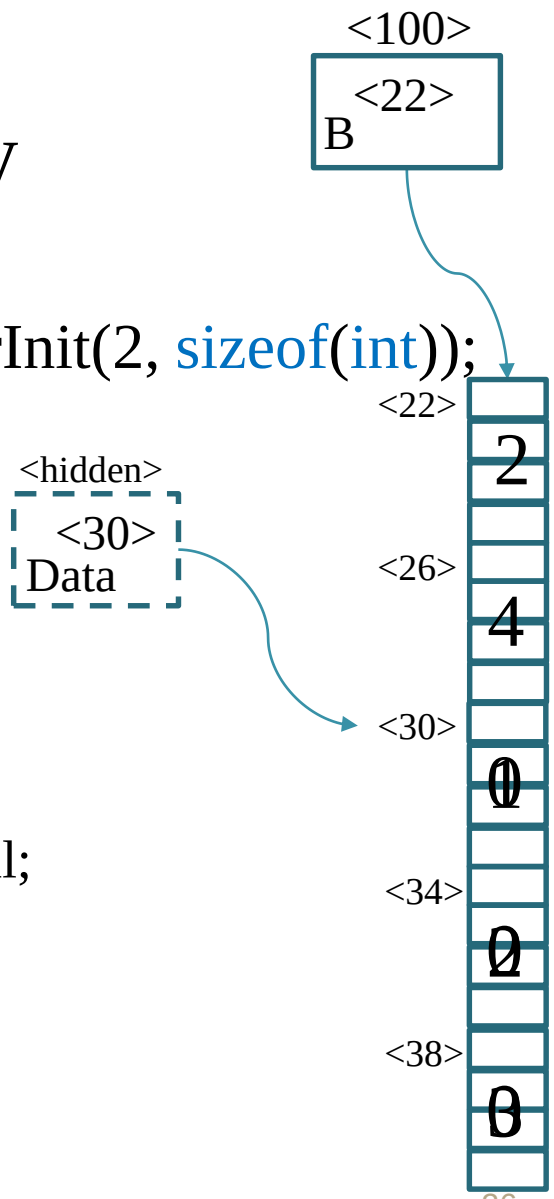
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - void* arrResize(void* aData, int n){//n=2
    - if(aData == NULL || n < 0)
      - return NULL;
    - ArrayStruct* as = StructOf(aData);
    - int szItem = as->sizeItem;
    - int szNew = memSize(n, sizeItem);
    - ArrayStruct* aNew = (ArrayStruct*)realloc(as, szNew);
    - if(aNew != NULL){
      - aNew->nItem = n;
      - return aNew->Data;
    - }
    - return NULL;
  - }



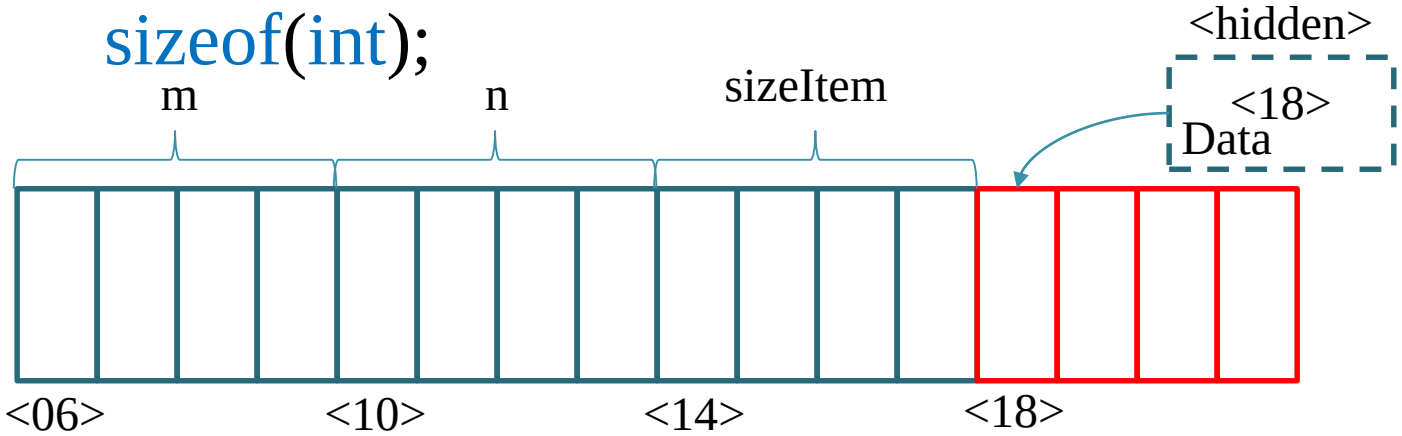
STRUCTURAL MEMORY (1D ARRAY)

- Review example of 1D array
  - void main(){
    - ArrayStruct*B=(ArrayStruct*)arrInit(2, sizeof(int));
    - if(B != NULL){
      - *((int*)(B->Data) + 0) = 1;
      - *((int*)(B->Data) + 1) = 2;
      - *((int*)(B->Data) + 2) = 3;
      - cout<< *((int*)(B->Data) + 0) << “, ”
        << *((int*)(B->Data) + 1) << “, ”
        << *((int*)(B->Data) + 2) << endl;
      - arrFree(B);
    - }
  - }



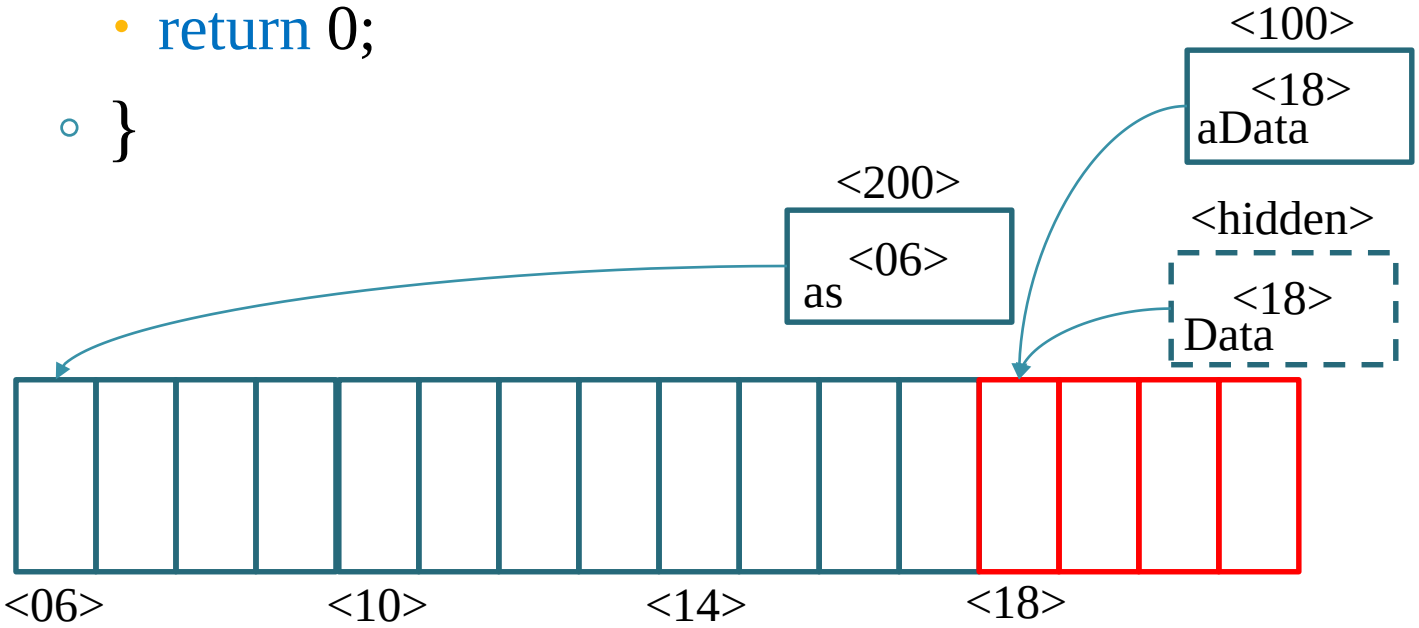
STRUCTURAL MEMORY (2D ARRAY)

- Review example of 2D array
  - typedef struct {
    - int m, n, sizeItem;
    - void* Data[1];
  - }aStruct;
  - static int headSize = sizeof(int) + sizeof(int) + sizeof(int);



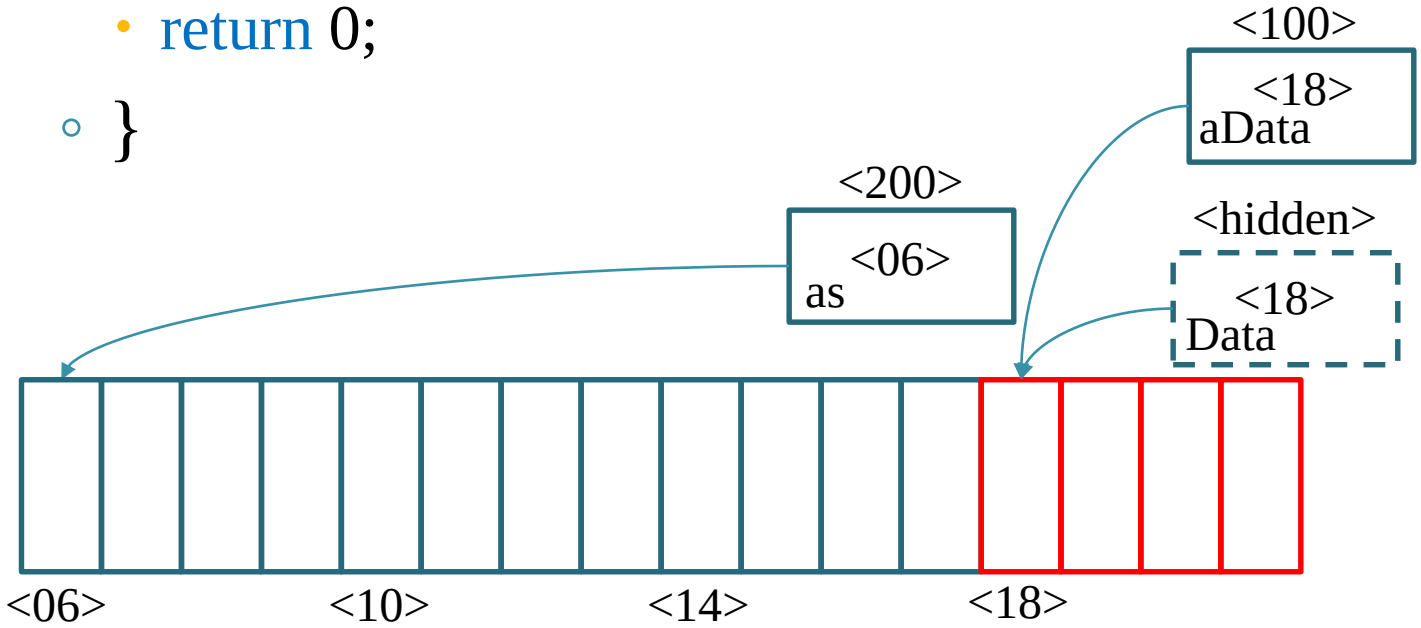
STRUCTURAL MEMORY (2D ARRAY)

- Review example of 2D array
  - int nRow(void** aData){
    - aStruct* as = (aStruct*)((char*)aData – headSize);
    - if(as != NULL) return as->m;
    - return 0;
  - }



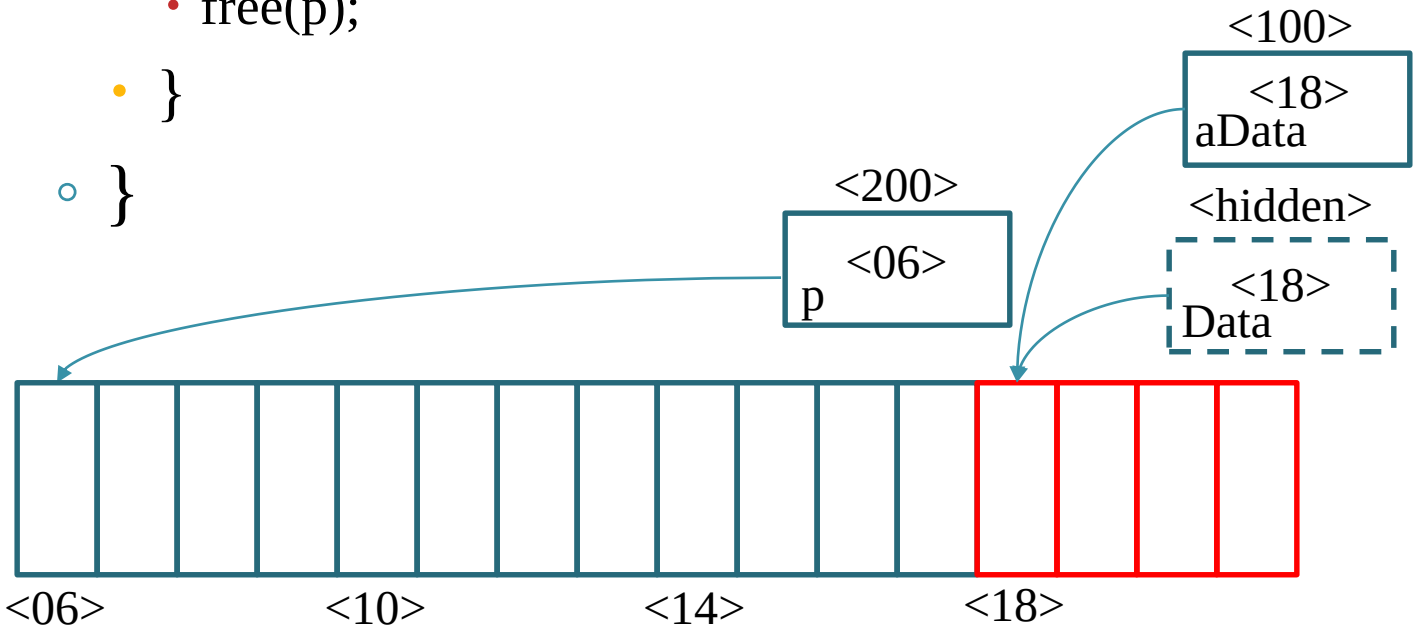
STRUCTURAL MEMORY (2D ARRAY)

- Review example of 2D array
  - int nCol(void** aData){
    - aStruct* as = (aStruct*)((char*)aData – headSize);
    - if(as != NULL) return as->n;
    - return 0;
  - }



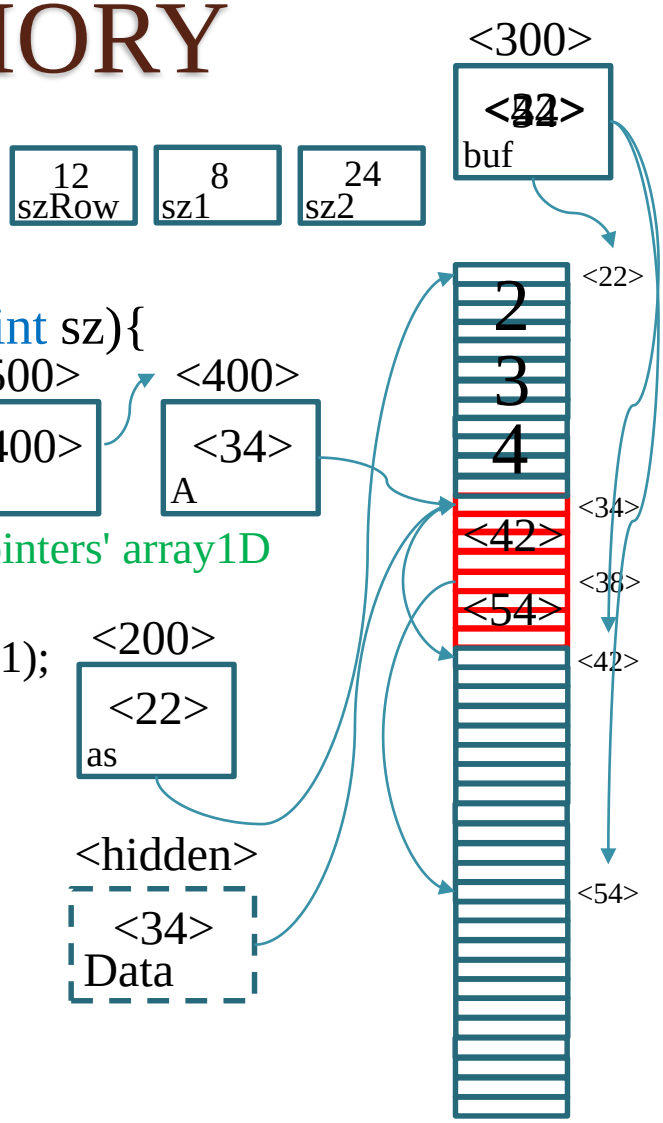
STRUCTURAL MEMORY (2D ARRAY)

- Review example of 2D array
  - void Free2D(void** aData){
    - if(aData != NULL){
      - void* p = (char*)aData – headSize;
      - free(p);
    - }
  - }



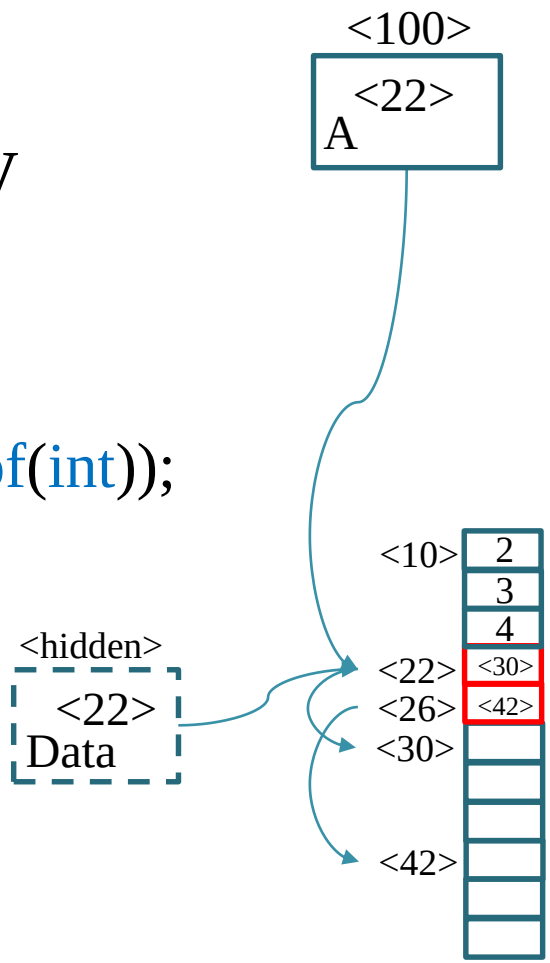
STRUCTURAL MEMORY (2D ARRAY)

- Review example of 2D array
  - void alloc2D(void*** a, int m, int n, int sz){
    - //m = 2, n = 3, sz = 4
    - if(m <= 0 || n <= 0 || sz <= 0) return;
    - int szRow = n * sz; // bytes per row
    - int sz1 = m * sizeof(void*); // bytes of pointers' array1D
    - int sz2 = m * szRow; // bytes of all data
    - void* buf = calloc(headSize + sz1 + sz2, 1);
    - if(buf == NULL) return;
    - aStruct* as = (aStruct*)buf;
    - as->m = m; as->n = n; as->sizeItem = sz;
    - buf = (char*)buf + headSize + sz1;
    - as->Data[0] = buf;
    - for(int i = 1; i < m; i++){
      - buf = (char*)buf + szRow;
      - as->Data[i] = buf;
    - }
    - *a = as->Data;
  - }



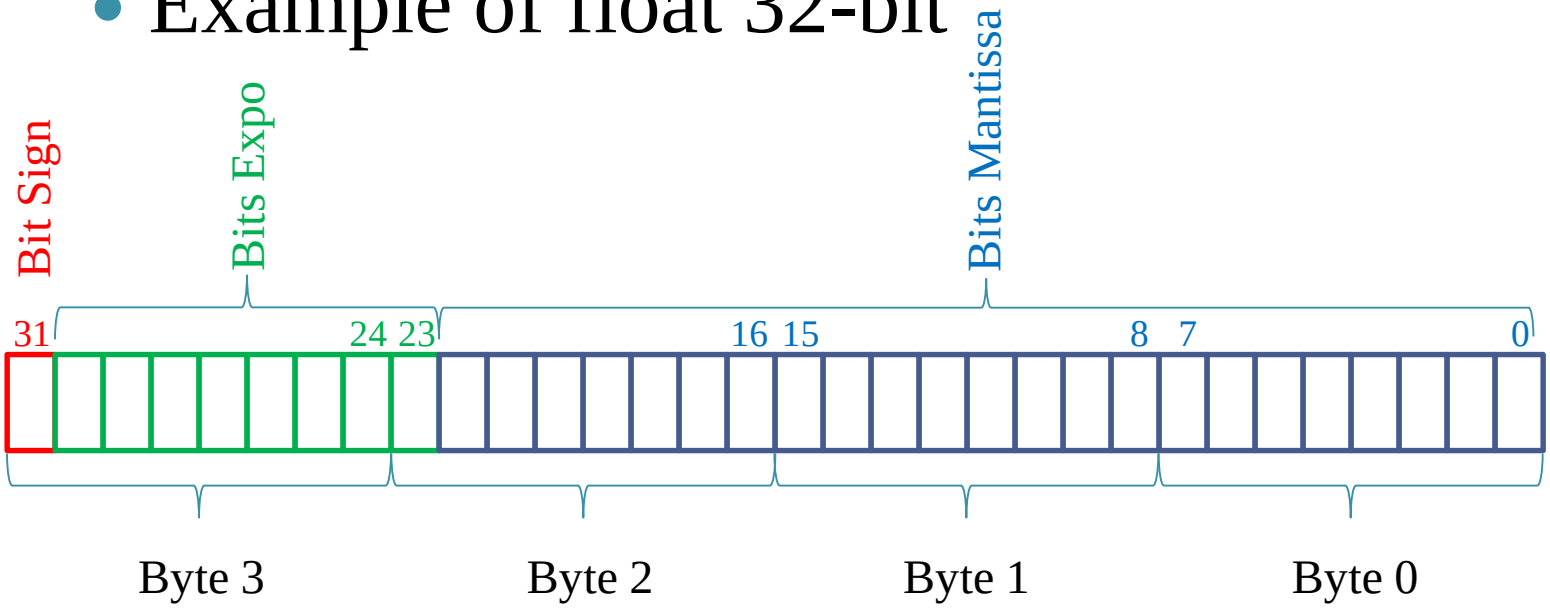
STRUCTURAL MEMORY (2D ARRAY)

- Review example of 2D array
  - void main(){
    - int** A;
    - alloc2D((void***)&A, 2, 3, sizeof(int));
    - arr2D_Input(A);
    - arr2D_Output(A);
    - free2D((void**)A);
  - }



STRUCTURAL MEMORY (FLOAT STANDARD 32-bit IEEE)

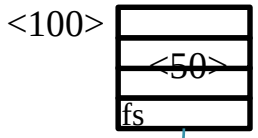
- Example of float 32-bit

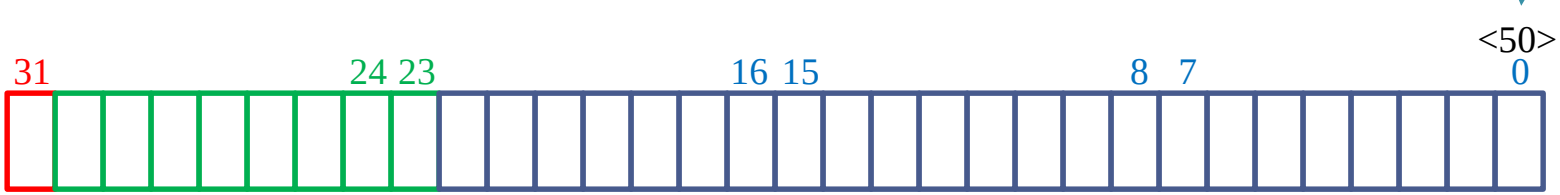



```
#pragma GCC diagnostic ignored "-Wpedantic"
typedef union{
  float Value; unsigned long dWord; unsigned short Words[2]; unsigned char Bytes[4];
  struct{ unsigned long Mantissa: 23; unsigned int Expo: 8; unsigned int Sign: 1; };
}floatStruct;
```

STRUCTURAL MEMORY (FLOAT STANDARD 32-bit IEEE)

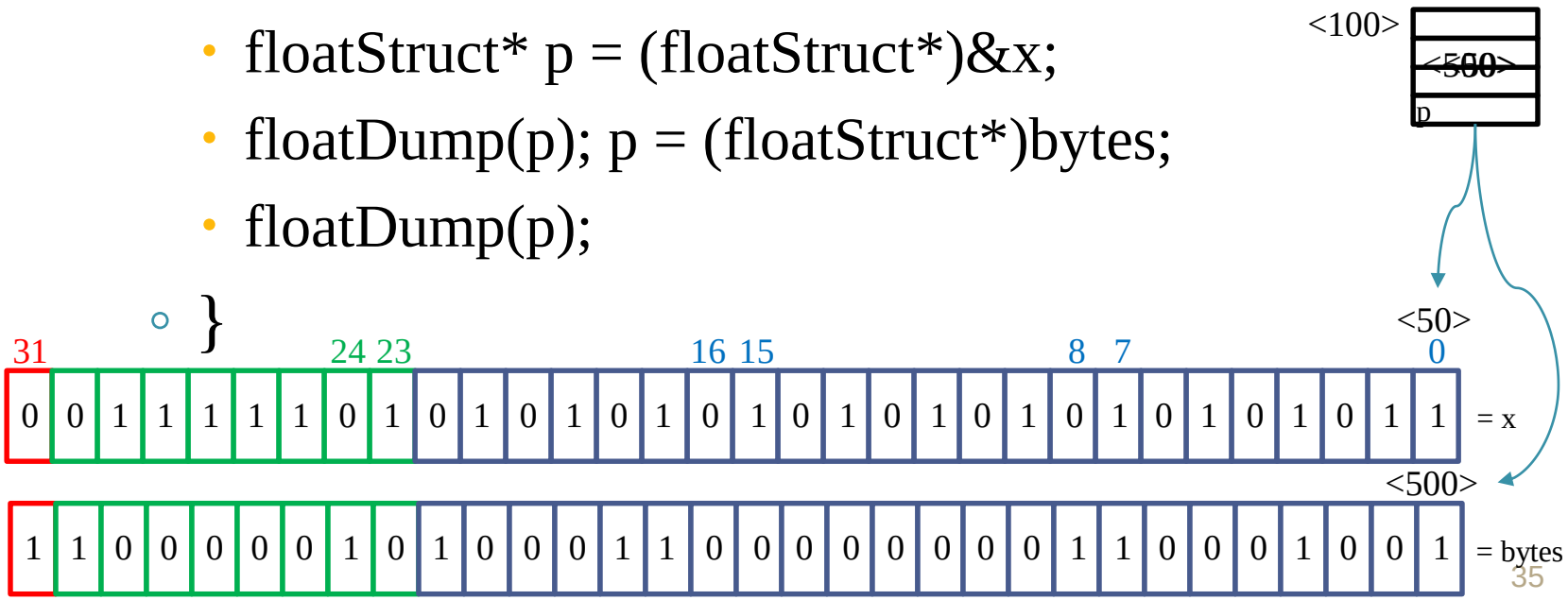
- Example of float 32-bit
  - void floatDump(floatStruct* fs){
    - if(fs == NULL) return;
    - printf("------------------------\n");
    - printf(" +value: %f\n", fs->Value);
    - printf(" +stored dword: 0x%lX\n", fs->dWord);
    - printf(" +stored words: 0x%04X 0x%04X\n", fs->Words[0], fs->Words[1]);
    - printf(" +stored bytes: ");
    - for(int i = 0; i < sizeof(float); i++)
      - printf("0x%02X ", fs->Bytes[i]);
    - printf("\n");
    - printf(" + IEEE stored parts:\n");
    - printf("   Sign: %d\n", fs->Sign);
    - printf("   Expo: 0x%X\n", fs->Expo);
    - printf("   Mantissa: 0x%lX\n", fs->Mantissa);
  - }





STRUCTURAL MEMORY (FLOAT STANDARD 32-bit IEEE)

- Example of float 32-bit
  - void main(){
    - float x = 1/(float)3;
    - unsigned char bytes[] = {0x89, 0x01, 0x46, 0xC1};
    - //bytes = -12.375375f
    - floatStruct* p = (floatStruct*)&x;
    - floatDump(p); p = (floatStruct*)bytes;
    - floatDump(p);
  - }

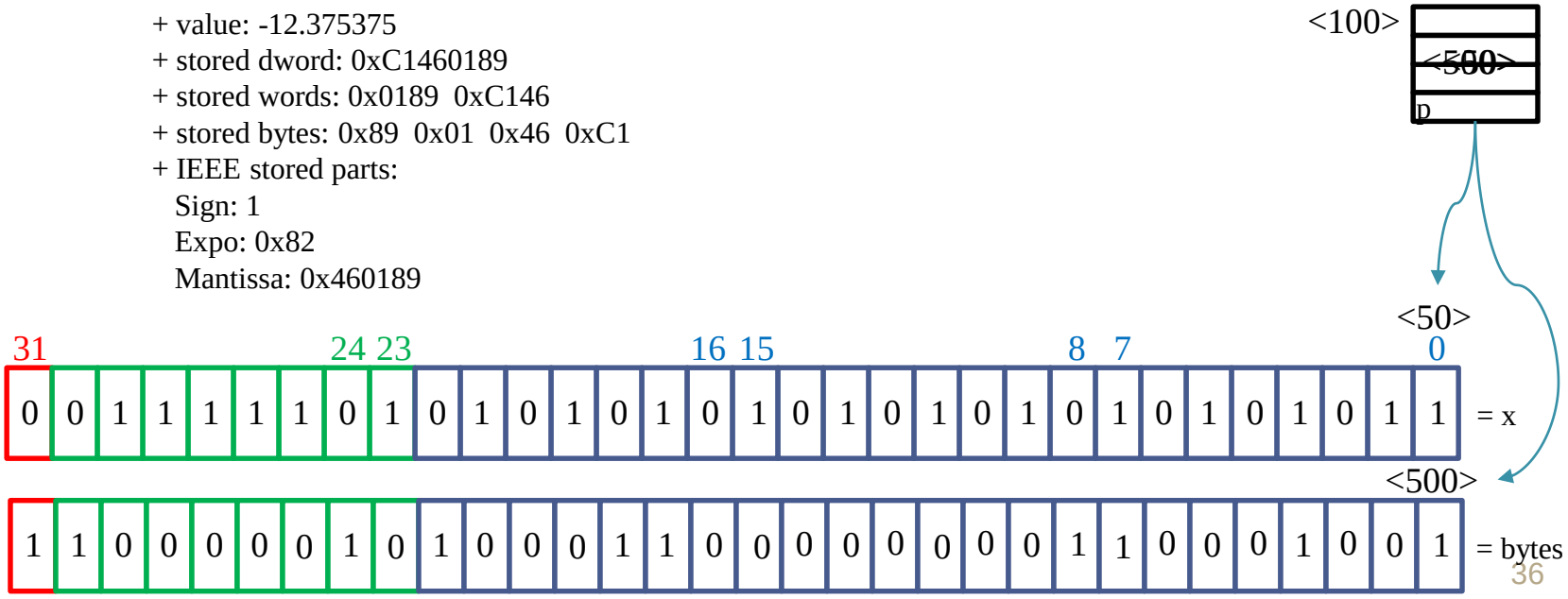


STRUCTURAL MEMORY (FLOAT STANDARD 32-bit IEEE)

- Example of float 32-bit

+ value: 0.333333
+ stored dword: 0x3EAAAAAB
+ stored words: 0xAAAB 0x3EAA
+ stored bytes: 0xAB 0xAA 0xAA 0x3E
+ IEEE stored parts:
  Sign: 0
  Expo: 0x7D
  Mantissa: 0x2AAAAB

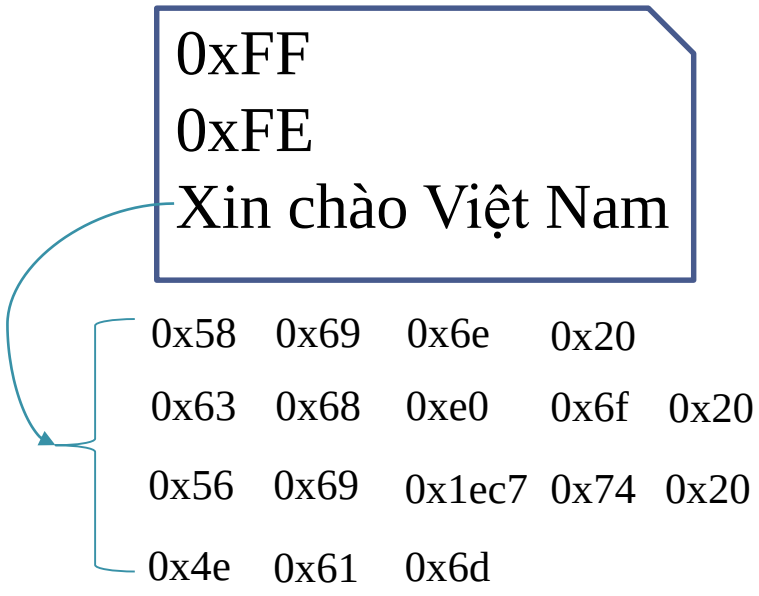
+ value: -12.375375
+ stored dword: 0xC1460189
+ stored words: 0x0189 0xC146
+ stored bytes: 0x89 0x01 0x46 0xC1
+ IEEE stored parts:
  Sign: 1
  Expo: 0x82
  Mantissa: 0x460189



STRUCTURAL MEMORY (16-bit TEXT FILE)

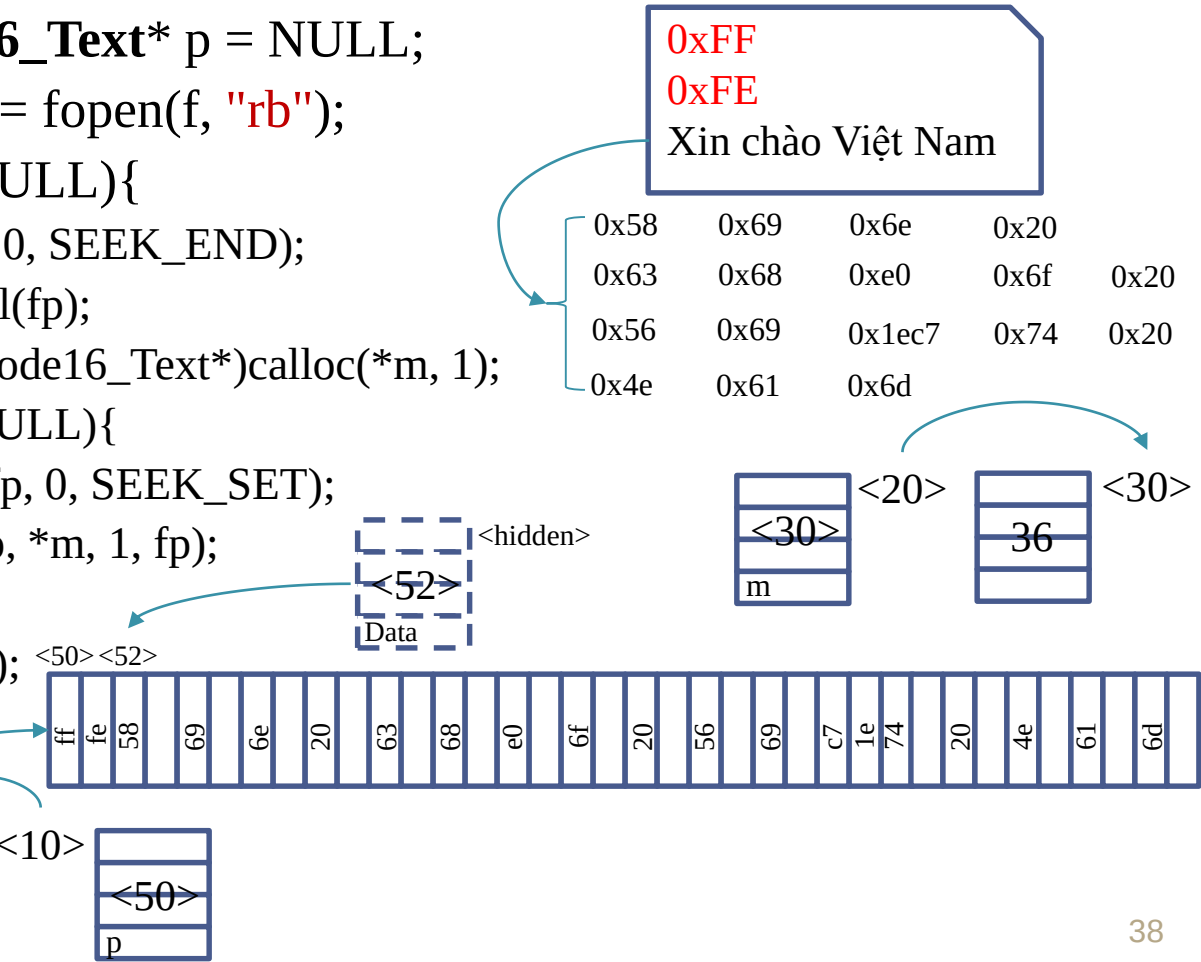
- Example of 16-bit text file

```
typedef struct{
  union{
    unsigned char markBytes[2];
    unsigned short markWord;
  };
  wchar_t Data[1];
}unicode16_Text;
```



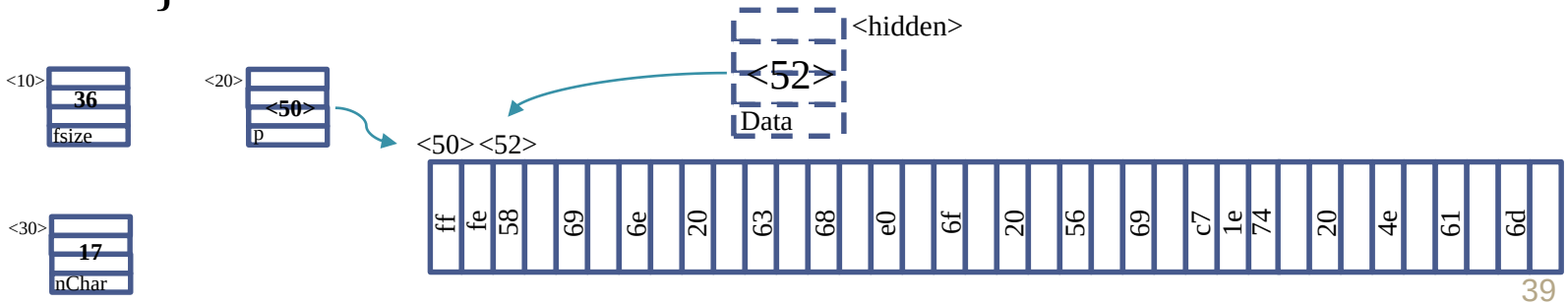
STRUCTURAL MEMORY (16-bit TEXT FILE)

- Example of 16-bit text file
  - **unicode16_Text*** readFileUtf16(char* f, long* m){
    - **unicode16_Text*** p = NULL;
    - FILE* fp = fopen(f, "rb");
    - if(fp != NULL){
      - fseek(fp, 0, SEEK_END);
      - *m = ftell(fp);
      - p = (unicode16_Text*)calloc(*m, 1);
      - if(p != NULL){
        - fseek(fp, 0, SEEK_SET);
        - fread(p, *m, 1, fp);
      - }
      - fclose(fp);
    - }
    - return p;
  - }



STRUCTURAL MEMORY (16-bit TEXT FILE)

- Example of 16-bit text file
  - void main(){
    - long fsize;
    - unicode16_Text* p = readFileUtf16("abc.inp", &fsize);
    - long nChar = (fsize - 2)/2;
    - if(p != NULL) cout << nChar << endl;
    - for(int i = 0; i < nChar; i++)
      - cout << hex << "0x" << (short)p->Data[i] << "|";
    - free(p);
  - }



OPERATOR [] FOR STRUCTURAL VARIABLE

- This operator allows to further define with complex datatypes
- Use this operator help to save memory reserved for pointers (review 2D/3D array)
- Prototype of this operator must return a reference
- This operator should be defined in ‘struct’ block

OPERATOR [] FOR STRUCTURAL VARIABLE

- Example of round array: define a operator [] to freely transmit negative/positive/out-of-bound index
- Example: a[-1] or a[MAX + 1]…
  - template <class T>
  - struct roundArray{
    - int nItem;
    - T* arr, Dummy;
    - //Need '&' because operator '[]' is left-exp of operator assignment '='
    - T& operator[](int i){
      - if(arr == NULL || nItem <= 0) return Dummy;
      - if(i >= nItem) i = i % nItem;
      - else if(i < 0) {
        - int j = (-i) % nItem;
        - if(j == 0) i = 0;
        - else i = nItem - j;
      - }
      - return arr[i];
    - }
  - };

OPERATOR [] FOR STRUCTURAL VARIABLE

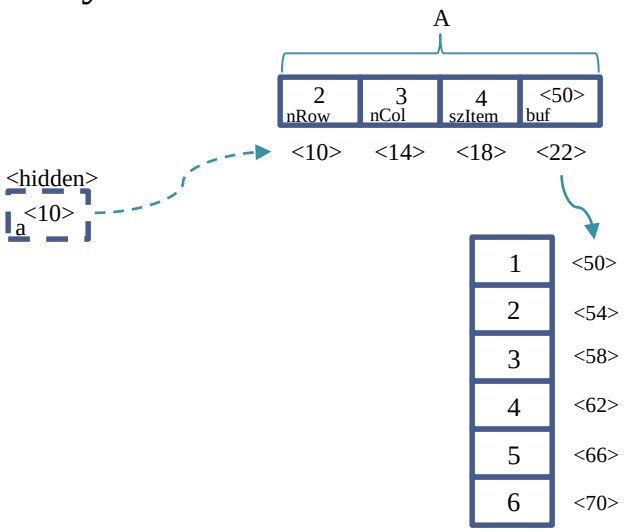
- Example of round array: define a operator [] to freely transmit negative/positive/out-of-bound index
- Example: a[-1] or a[MAX + 1]…
  - template <class T>
  - void roundArrayInit(roundArray<T>& a, int n){
    - a.nItem = 0; a.arr = NULL;
    - if(n <= 0) return;
    - a.nItem = n; a.arr = new T[n];
  - }
  - template <class T>
  - void roundArrayFree(roundArray<T>& a){
    - if(a.arr != NULL){
      - delete []a.arr; a.arr = NULL;
    - }
  - }

OPERATOR [] FOR STRUCTURAL VARIABLE

- Example of round array: define a operator [] to freely transmit negative/positive/out-of-bound index
- Example: a[-1] or a[MAX + 1]…
  - void main(){
    - roundArray<int> A; int n = 5;
    - roundArrayInit<int>(A, n);
    - for(int i = 0; i < A.nItem; i++) A[i] = i;
    - for(int i = -7; i <= 17; i++){
      - cout << " " << A[i];
      - if(i % n == 0) cout << endl;
    - }
    - roundArrayFree(A);
  - }

OPERATOR [] FOR STRUCTURAL VARIABLE

- Example of 2D array: define operator [] in 2D array
  - struct array2D{
    - int nRow, nCol, szItem; void* buf;
    - void* operator[](int i){
      - if(i < 0 || i >= nRow) i = 0;
      - return (char*)buf + i * nCol * szItem;
    - }
  - };
  - bool arr2D_Init(array2D& a, int m, int n, int sz){
    - if(m <= 0 || n <= 0 || sz <= 0) return false;
    - a.nRow = m; a.nCol = n; a.szItem = sz;
    - a.buf = calloc(m * n, sz);
    - if(a.buf == NULL) return false;
    - return true;
  - }
  - void arr2D_free(array2D& a){ if(a.buf != NULL) free(a.buf); }
  - void main(){
    - array2D A;
    - if(**arr2D_Init**(A, 2, 3, sizeof(int))){
      - for(int i = 0; i < 2; i++)
        - for(int j = 0; j < 3; j++)
          - cin >> **((int*)A[i])**[j]; // Example of inputing 1 2 3 4 5 6
      - **arr2D_free**(A);
    - }
  - }



OPERATOR [] FOR STRUCTURAL VARIABLE

- Ex of 2D array: structure with template

		template <class T>
1	struct array2D{	struct array2D{
2	int nRow, nCol, szItem; void* buf;	int nRow, nCol; T* buf;
3	void* operator[](int i){	T* operator[](int i){
4	if(i < 0 || i >= nRow) i = 0;	if(i < 0 || i >= nRow) i = 0;
5	return (char*)buf + i * nCol * szItem;	return buf + i * nCol;
6	}};	}};
7		template <class T>
8	bool arr2D_Init(array2D& a, int m, int n, int sz){	bool arr2D_Init(array2D<T> &a, int m, int n){
9	if(m <= 0 || n <= 0 || sz <= 0) return false;	if(m <= 0 || n <= 0) return false;
10	a.nRow = m; a.nCol = n; a.szItem = sz;	a.nRow = m; a.nCol = n;
11	a.buf = calloc(m * n, sz);	a.buf = (T*)calloc(m * n, sizeof(T));
12	if(a.buf == NULL) return false;	if(a.buf == NULL) return false;
13	return true;	return true;
14	}	}

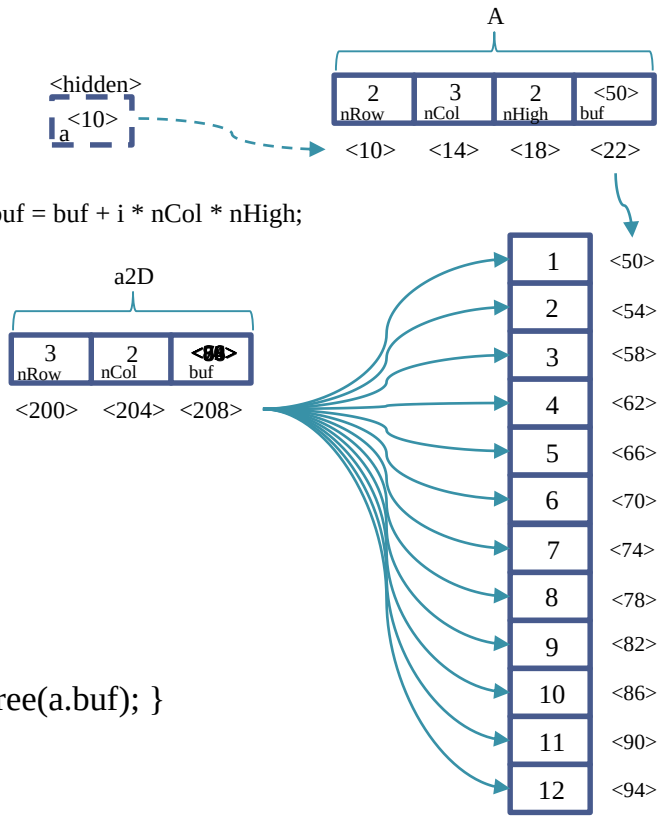
OPERATOR [] FOR STRUCTURAL VARIABLE

- Ex of 2D array: structure with template

		template <class T>
15	void arr2D_free(array2D& a){	void arr2D_free(array2D<T> &a){
16	if(a.buf != NULL) free(a.buf);	if(a.buf != NULL) free(a.buf);
17	}	}
18	void main(){	void main(){
19	array2D A;	array2D<int> A;
20	if(**arr2D_Init**(A, 2, 3, sizeof(int))){	if(**arr2D_Init**<int>(A, 2, 3)){
21	for(int i = 0; i < 2; i++)	for(int i = 0; i < 2; i++)
22	for(int j = 0; j < 3; j++)	for(int j = 0; j < 3; j++)
23	cin >> **((int*)A[i])**[j]; // input 1 2 3 4 5 6	cin >> A[i][j]; // input 1 2 3 4 5 6
24	**arr2D_free**(A);	**arr2D_free**<int>(A);
25	}	}

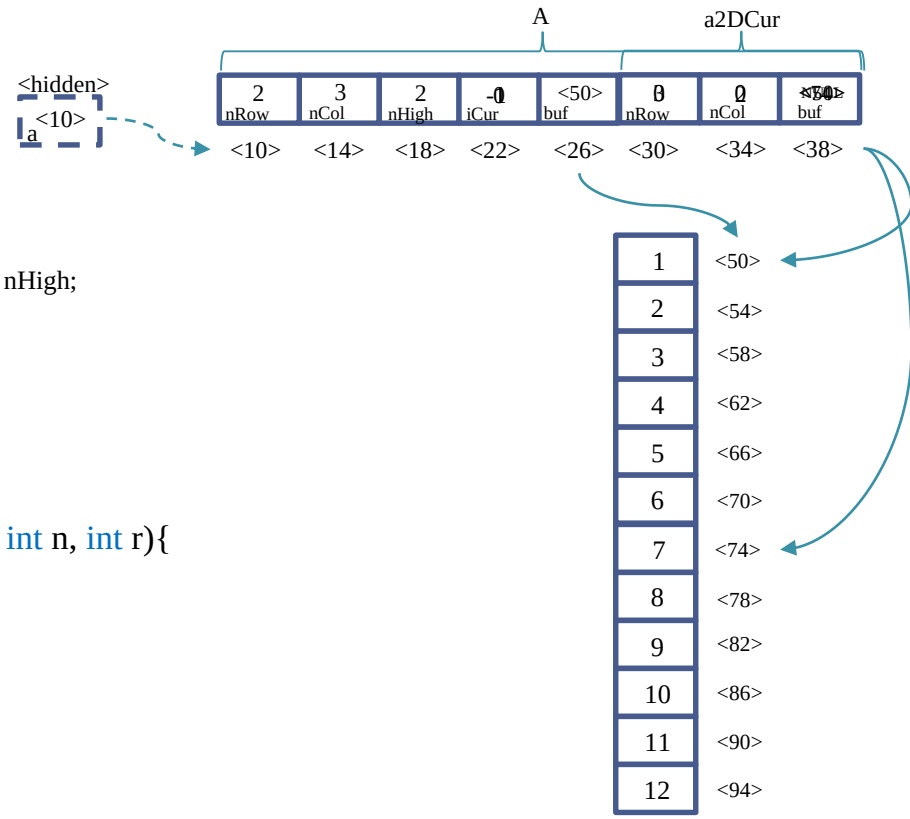
OPERATOR [] FOR STRUCTURAL VARIABLE

- Example of 3D array: define operator [] in 3D structure
  - template <class T>
  - struct array3D{
    - int nRow, nCol, nHigh; T* buf;
    - array2D<T> operator[](int i){
      - if(i < 0 || i >= nRow) i = 0;
      - array2D<T> a2D; a2D.nRow = nCol; a2D.nCol = nHigh; a2D.buf = buf + i * nCol * nHigh;
      - return a2D;
    - }};
  - template <class T>
  - bool arr3D_Init(array3D<T>& a, int m, int n, int r){
    - if(m <= 0 || n <= 0 || r <= 0) return false;
    - a.nRow = m; a.nCol = n; a.nHigh = r;
    - a.buf = (T*)calloc(m * n * r, sizeof(T));
    - if(a.buf == NULL) return false;
    - return true;
  - }
  - template <class T>
  - void arr3D_free(array3D<T>& a){ if(a.buf != NULL) free(a.buf); }
  - void main(){
    - array3D<int> A;
    - if(**arr3D_Init**<int>(A, 2, 3, 2)){
      - for(int i = 0; i < 2; i++)
        - for(int j = 0; j < 3; j++)
          - for(int k = 0; k < 2; k++)
            - cin >> A[i][j][k]; // input 1 2 3 4 5 6 7 8 9 10 11 12
      - **arr3D_free**<int>(A);
    - }
  - }



OPERATOR [] FOR STRUCTURAL VARIABLE

- Example of 3D array: define operator [] in 3D structure (improved)
  - template <class T>
  - struct array3D{
    - int nRow, nCol, nHigh, iCur; T* buf;
    - array2D<T> a2DCur;
    - array2D<T> operator[](int i){
      - if(i < 0 || i >= nRow) i = 0;
      - if(i != iCur){
        - a2DCur.nRow = nCol; a2DCur.nCol = nHigh;
        - a2DCur.buf = buf + i * nCol * nHigh;
        - iCur = i;
      - }
      - return a2DCur;
    - }};
  - template <class T>
  - bool arr3D_Init(array3D<T>& a, int m, int n, int r){
    - if(m <= 0 || n <= 0 || r <= 0) return false;
    - a.nRow = m; a.nCol = n; a.nHigh = r;
    - a.iCur = -1; a.a2DCur = {0};
    - a.buf = (T*)calloc(m * n * r, sizeof(T));
    - if(a.buf == NULL) return false;
    - return true;
  - }
  - template <class T>
  - void arr3D_free(array3D<T>& a){ //… }
  - void main(){
    - //…
  - }

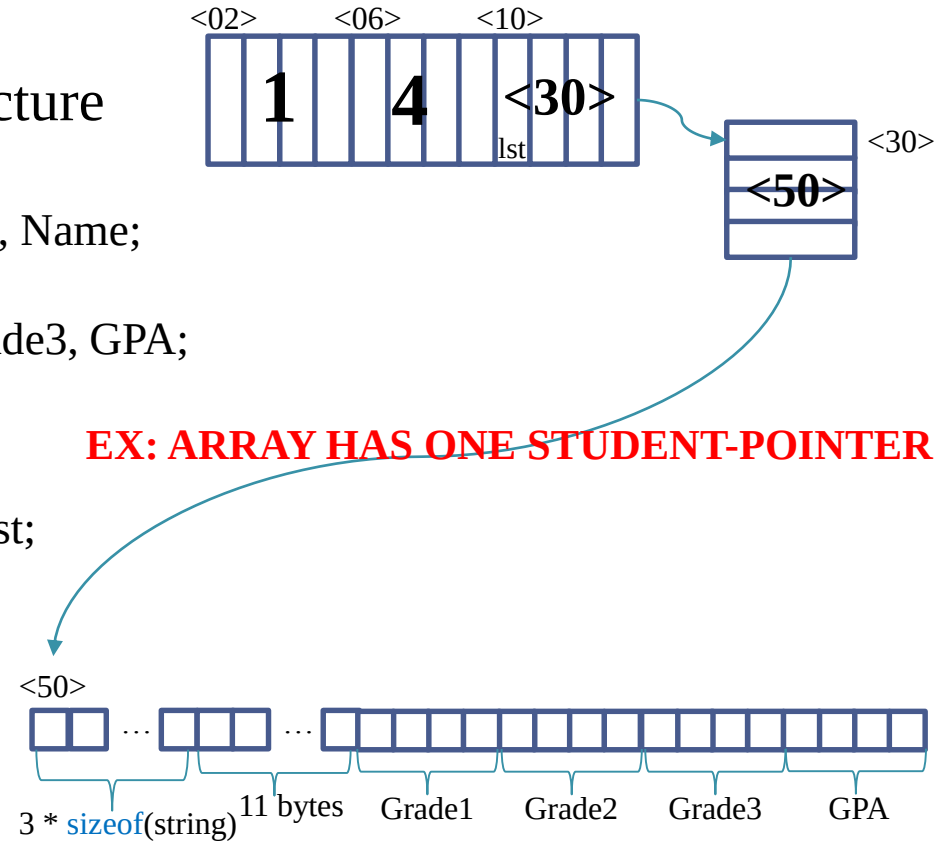


STRUCTURAL ARRAY

- Be an array, in which each element has a user-defined datatype (using struct)
- New datatype may have fixed or dynamic members
- With fixed member, we can directly allocate an amount of bytes (May use the method of creating 1D array as previous examples)
- With dynamic member, we need to specially process

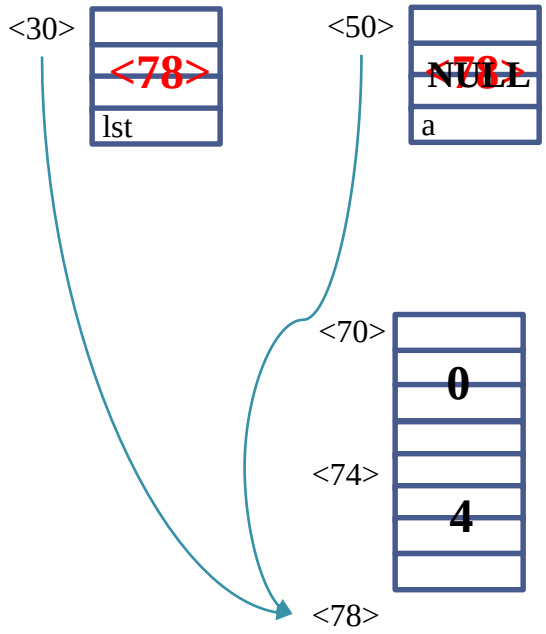
STRUCTURAL ARRAY

- Example of Student structure
  - typedef struct{
    - string Code, FamilyName, Name;
    - char BirthDate[11];
    - float Grade1, Grade2, Grade3, GPA;
  - } Student;
  - void main(){
    - Student** lst = **stInit**(0), st;
    - string stcode = "nocode";
    - while(1){
      - cin >> stcode;
      - if(stcode == "---") break;
      - st.Code = stcode;
      - **getStudent**(st);
      - if(**StPush**(&lst, st) == 0) break;
    - }
    - int n = **arrSize**((void*)lst);
    - for(int i = 0; i < n; i++) {//cout << lst[i]->Code …}
    - **StFree**(lst);
  - }



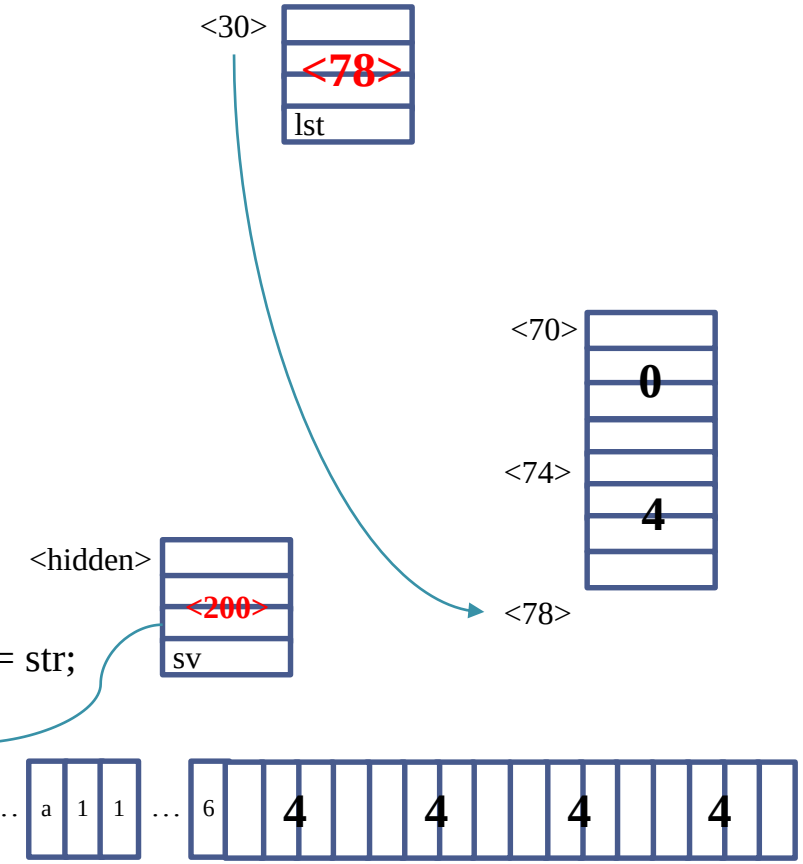
STRUCTURAL ARRAY

- Example of Student structure
  - typedef struct{
    - string Code, FamilyName, Name;
    - char BirthDate[11];
    - float Grade1, Grade2, Grade3, GPA;
  - } Student;
  - void main(){
    - Student** lst = **stInit**(0), st;
    - //…
  - }
  - Student** StInit(int n){
    - Student** a = NULL;
    - if(n < 0) n = 0;
    - a = (Student**)arrInit(n, sizeof(Student*));
    - return a;
  - }



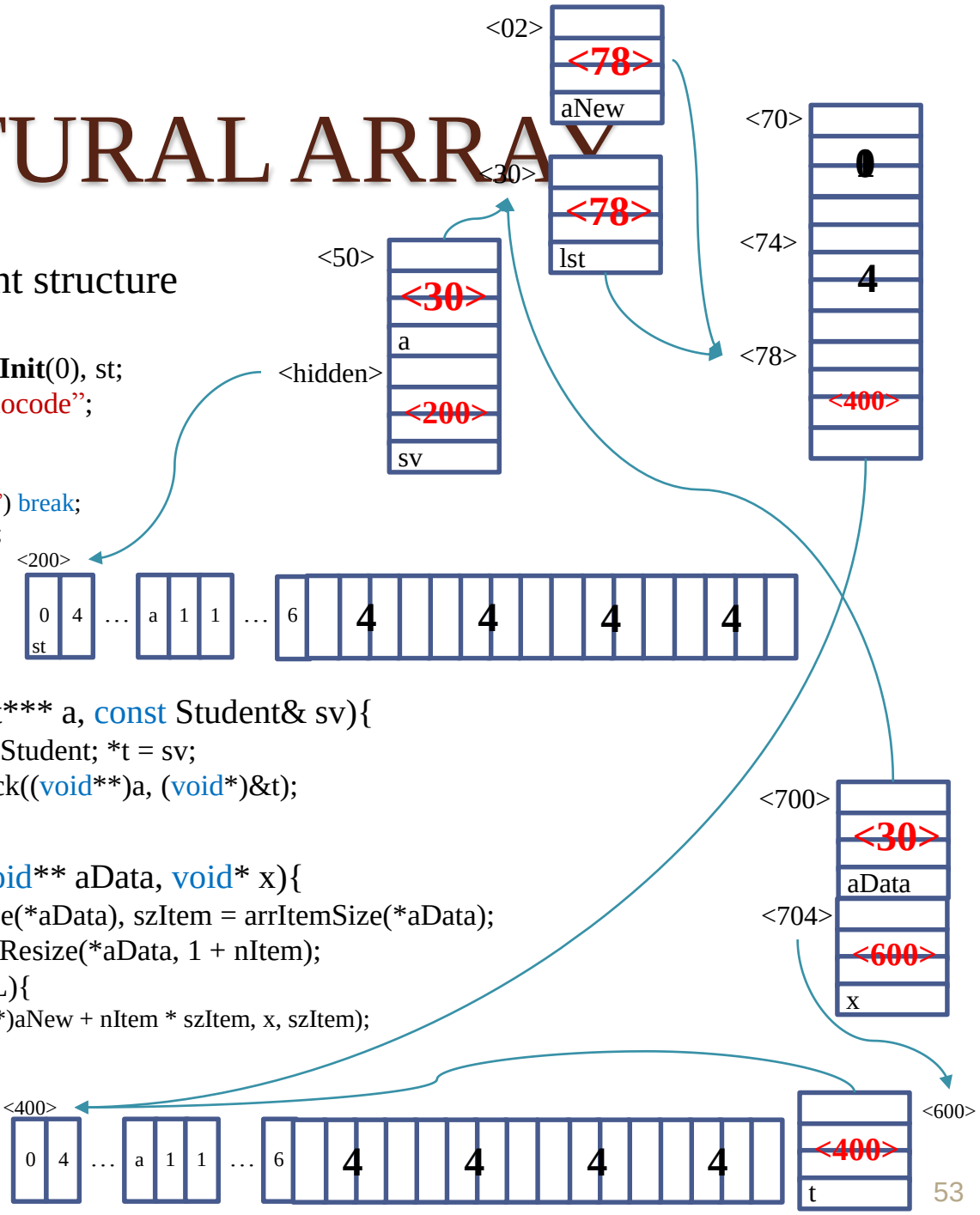
STRUCTURAL ARRAY

- Example of Student structure
  - void main(){
    - Student** lst = **stInit**(0), st;
    - string stcode = "nocode";
    - while(1){
      - cin >> stcode;
      - if(stcode == "---") break;
      - st.Code = stcode;
      - **getStudent**(st);
      - //…
  - }
  - void getStudent(Student& sv){
    - cin.ignore();
    - char str[256]; float g1, g2, g3;
    - cin.getline(str, 256); sv.FamilyName = str;
    - cin.getline(str, 256); sv.Name = str;
    - cin.getline(sv.BirthDate, 256);
    - cin >> g1 >> g2 >> g3;
    - sv.GPA = (g1 + g2 + g3)/3;
    - sv.Grade1 = g1; sv.Grade2 = g2; sv.Grade3 = g3;
  - }



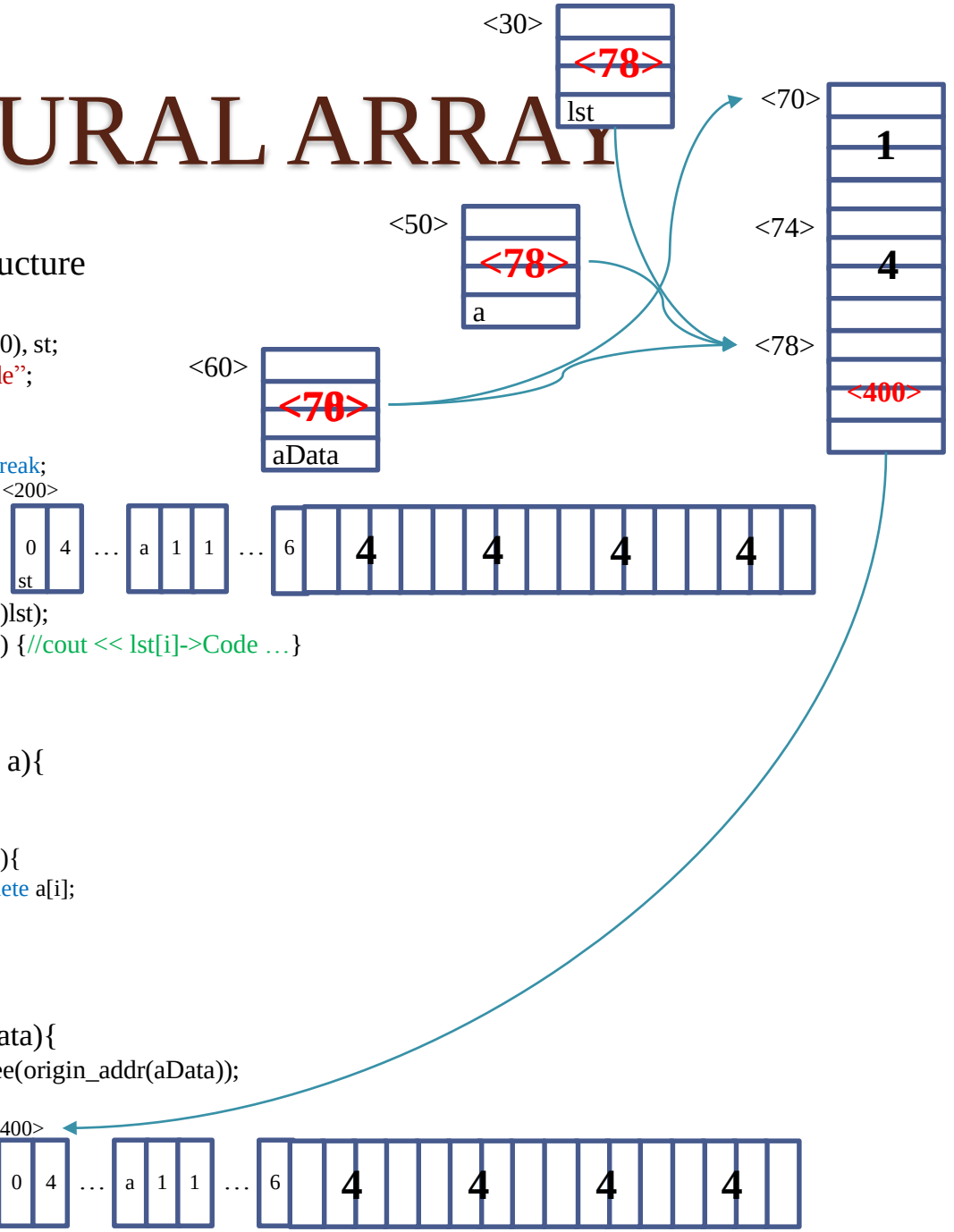
STRUCTURAL ARRAY

- Example of Student structure
  - void main(){
    - Student** lst = **stInit**(0), st;
    - string stcode = "nocode";
    - while(1){
      - cin >> stcode;
      - if(stcode == "---") break;
      - st.Code = stcode;
      - **getStudent**(st);
      - **StPush**(&lst, st);
      - //…
  - }
  - int StPush(Student*** a, const Student& sv){
    - Student* t = new Student; *t = sv;
    - return arrPushback((void**)a, (void*)&t);
  - }
  - 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 * szItem, x, szItem);
      - *aData = aNew;
      - return 1;
    - }
    - return 0;
  - }



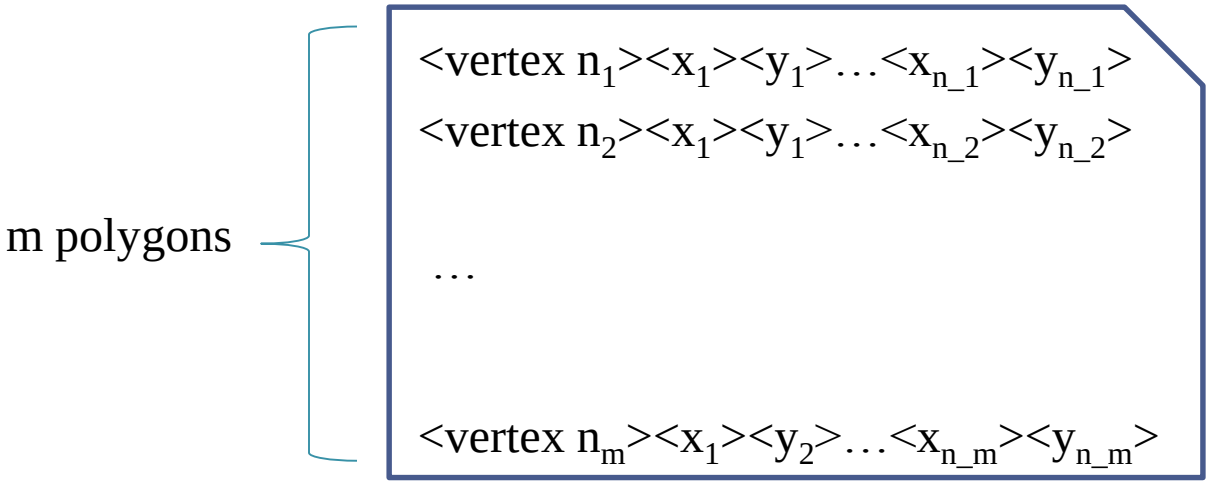
STRUCTURAL ARRAY

- Example of Student structure
  - void main(){
    - Student** lst = **stInit**(0), st;
    - string stcode = "nocode";
    - while(1){
      - cin >> stcode;
      - if(stcode == "---") break;
      - st.Code = stcode;
      - **getStudent**(st);
      - **StPush**(&lst, st);
    - }
    - int n = **arrSize**((void*)lst);
    - for(int i = 0; i < n; i++) {//cout << lst[i]->Code …}
    - **StFree**(lst);
  - }
  - void StFree(Student** a){
    - if(a == NULL) return;
    - int n = arrSize(a);
    - for(int i = 0; i < n; i++){
      - if(a[i] != NULL) delete a[i];
    - }
    - arrFree(a);
  - }
  - void arrFree(void* aData){
    - if(aData != NULL) free(origin_addr(aData));
  - }



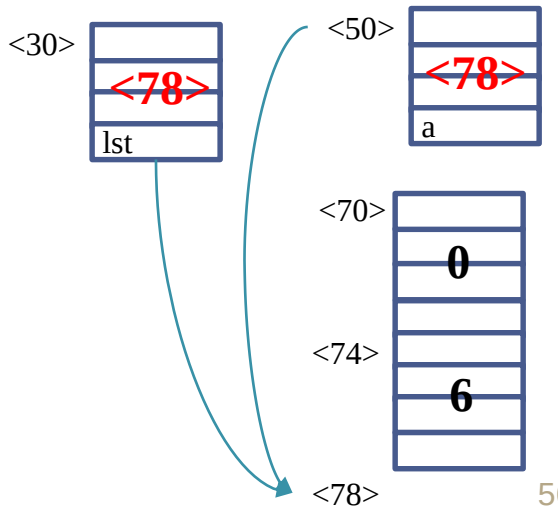
STRUCTURAL ARRAY

- Build structural array of polygons
- Use read/write binary files
- Reuse the functions of 1D Array
- Structure of file read/written is as follows:



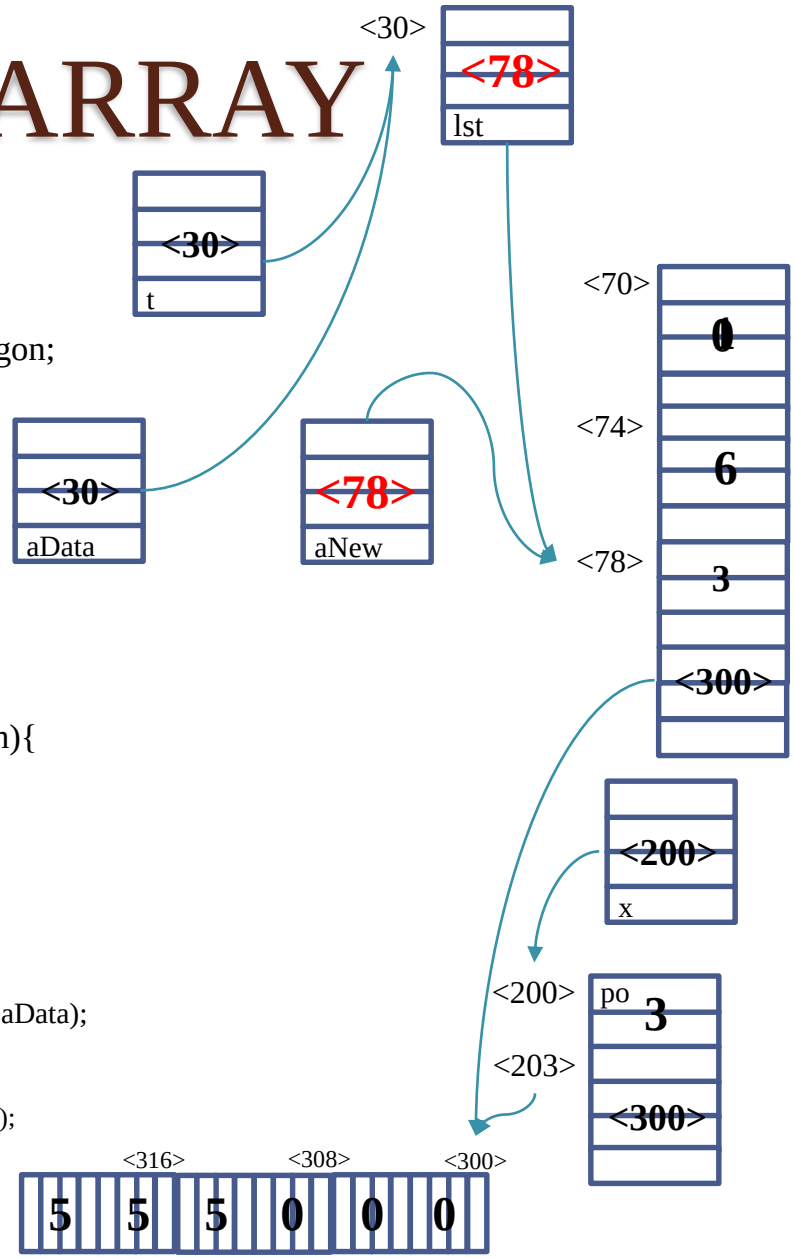
STRUCTURAL ARRAY

- Some struct declarations
  - typedef struct{ int x, y; } Point;
  - typedef struct { short nVer; Point* Vers; } Polygon;
  - void main(){
    - Point dg1[] = {{0, 0}, {0, 5}, {5, 5}}; int n1 = 3;
    - Point dg2[] = {{0, 0}, {0, 5}, {5, 5}, {5, 0}}; int n2 = 4;
    - Point dg3[] = {{0, 0}, {0, 5}, {1, 1}, {5, 5}, {5, 0}}; int n3 = 5;
    - Polygon* lst = **PolyListInit**();
    - if(lst != NULL){
      - PolyListPush(&lst, dg1, n1); PolyListPush(&lst, dg2, n2);
      - PolyListPush(&lst, dg3, n3);
      - PolyListSave(lst, "Polygons.dat");
      - PolyListFree(lst);
    - }
  - }
  - Polygon* **PolyListInit**(){
    - void* a = arrInit(0, sizeof(Polygon));
    - return (Polygon*)a;
  - }



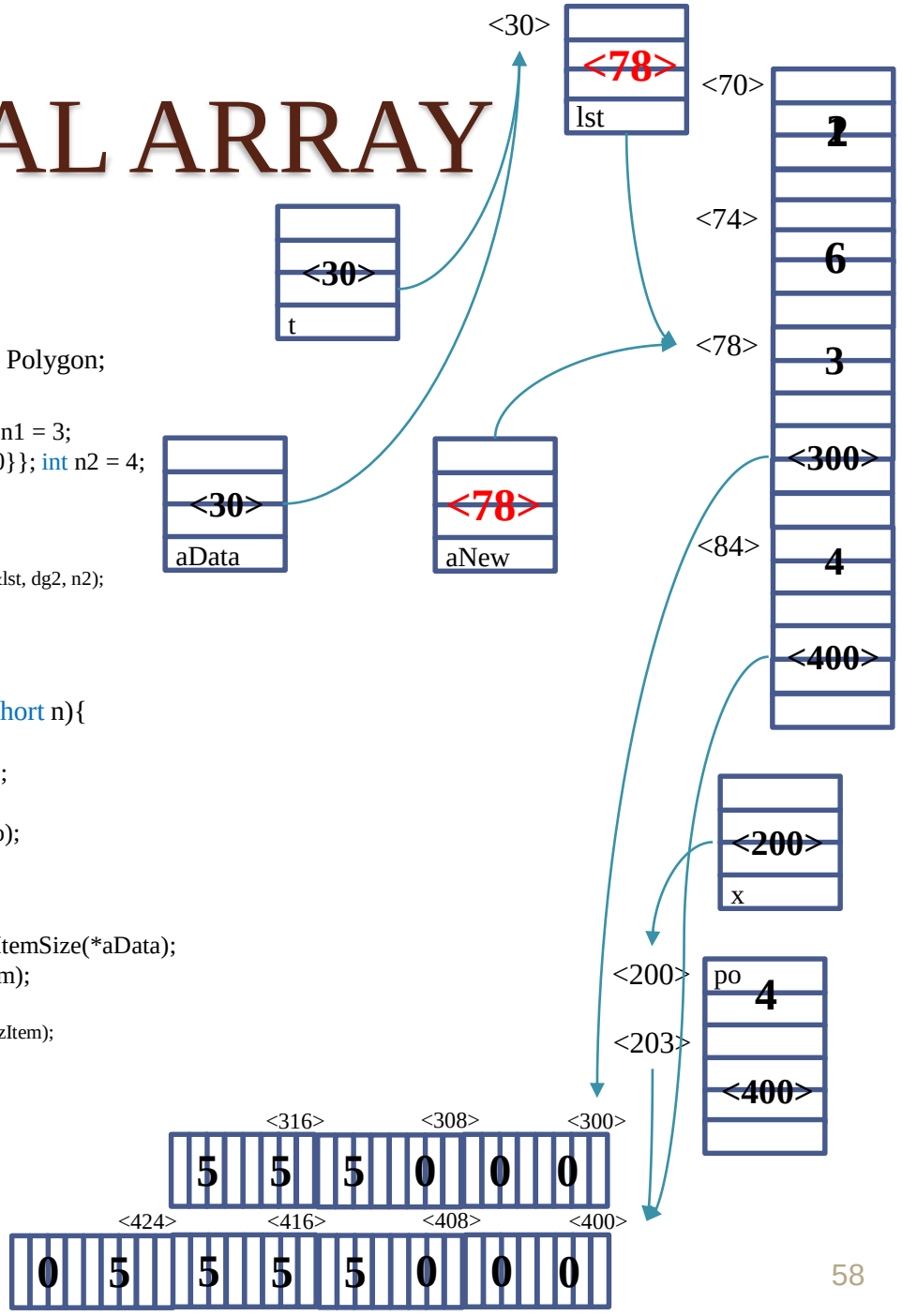
STRUCTURAL ARRAY

- Some struct declarations
  - typedef struct{ int x, y; } Point;
  - typedef struct { short nVer; Point* Vers; } Polygon;
  - void main(){
    - Point dg1[] = {{0, 0}, {0, 5}, {5, 5}}; int n1 = 3;
    - //…
    - Polygon* lst = PolyListInit();
    - if(lst != NULL){
      - **PolyListPush**(&lst, dg1, n1);
      - //…
    - }
  - }
  - int **PolyListPush**(Polygon** t, Point* P, short n){
    - Polygon po = {n};
    - po.Vers = (Point*)calloc(n, sizeof(Point));
    - for(int i = 0; i < n; i++) po.Vers[i] = P[i];
    - return arrPushback((void**)t, (void*)&po);
  - }
  - 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 * szItem, x, szItem);
      - *aData = aNew;
      - return 1;
    - }
    - return 0;
  - }



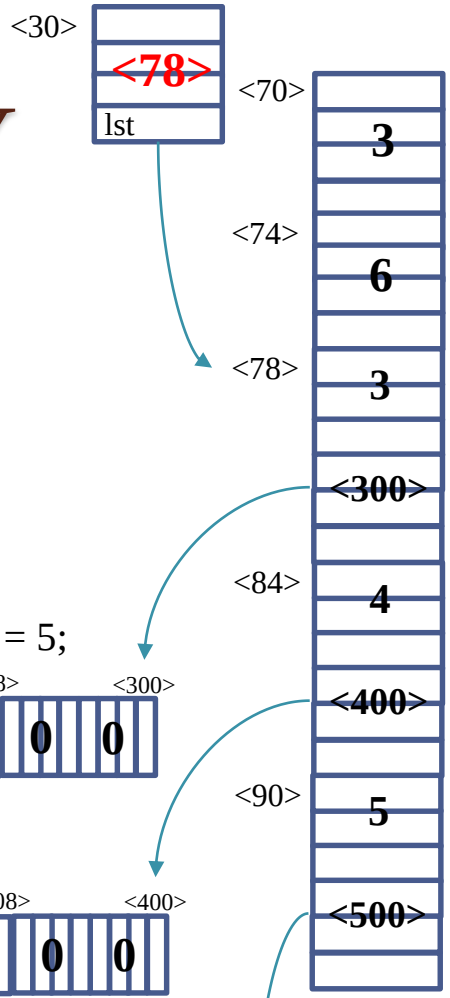
STRUCTURAL ARRAY

- Some struct declarations
  - typedef struct{ int x, y; } Point;
  - typedef struct { short nVer; Point* Vers; } Polygon;
  - void main(){
    - Point dg1[] = {{0, 0}, {0, 5}, {5, 5}}; int n1 = 3;
    - Point dg2[] = {{0, 0}, {0, 5}, {5, 5}, {5, 0}}; int n2 = 4;
    - //…
    - Polygon* lst = PolyListInit();
    - if(lst != NULL){
      - **PolyListPush**(&lst, dg1, n1); **PolyListPush**(&lst, dg2, n2);
      - //…
    - }
  - }
  - int **PolyListPush**(Polygon** t, Point* P, short n){
    - Polygon po = {n};
    - po.Vers = (Point*)calloc(n, sizeof(Point));
    - for(int i = 0; i < n; i++) po.Vers[i] = P[i];
    - return arrPushback((void**)t, (void*)&po);
  - }
  - 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 * szItem, x, szItem);
      - *aData = aNew;
      - return 1;
    - }
    - return 0;
  - }



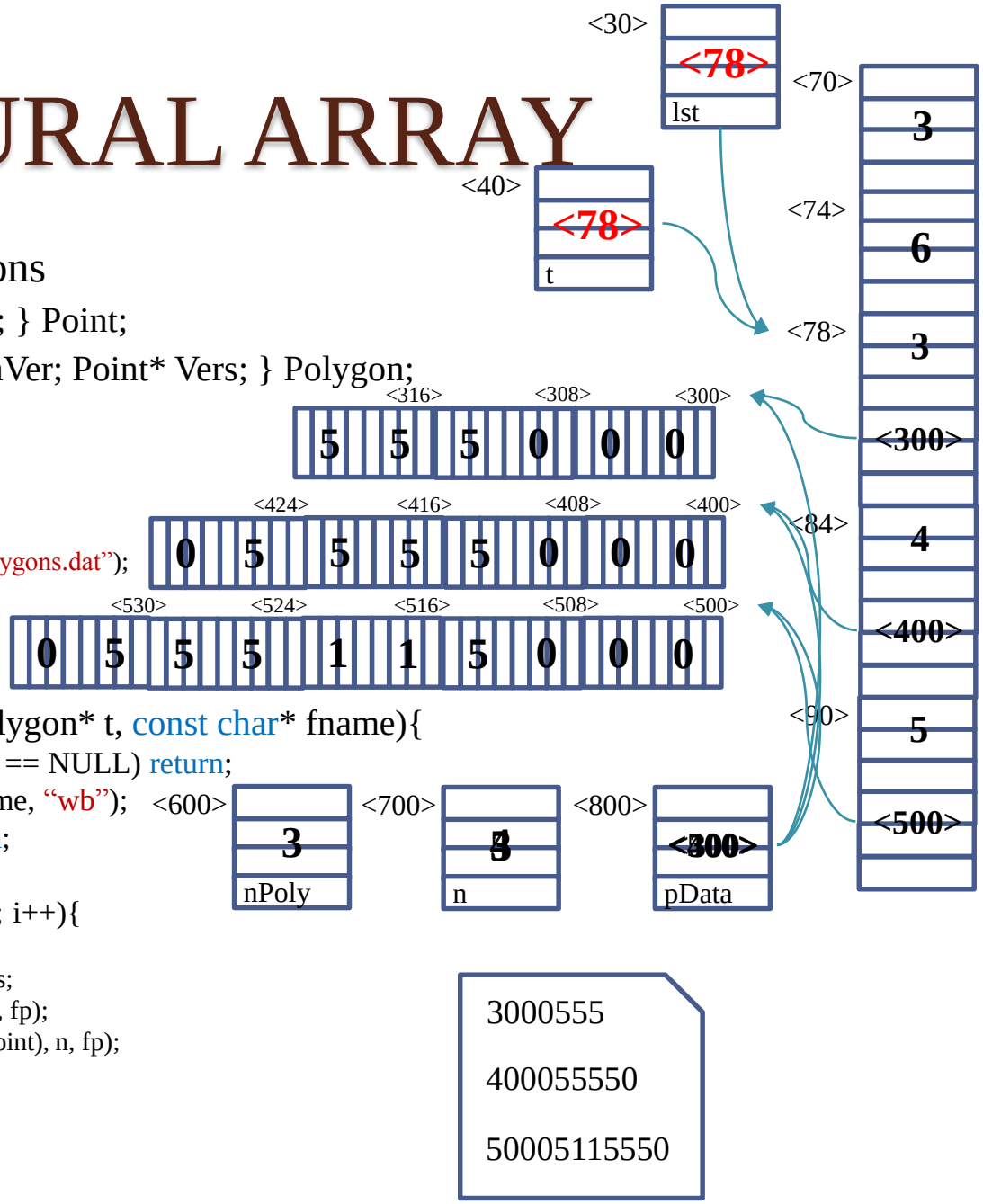
STRUCTURAL ARRAY

- Some struct declarations
  - typedef struct{ int x, y; } Point;
  - typedef struct { short nVer; Point* Vers; } Polygon;
  - void main(){
    - Point dg1[] = {{0, 0}, {0, 5}, {5, 5}}; int n1 = 3;
    - Point dg2[] = {{0, 0}, {0, 5}, {5, 5}, {5, 0}}; int n2 = 4;
    - Point dg3[] = {{0, 0}, {0, 5}, {1, 1}, {5, 5}, {5, 0}}; int n3 = 5;
    - //…
    - Polygon* lst = PolyListInit();
    - if(lst != NULL){
      - **PolyListPush**(&lst, dg1, n1); **PolyListPush**(&lst, dg2, n2);
      - **PolyListPush**(&lst, dg3, n3);
      - //…
    - }
  - }
  - int **PolyListPush**(Polygon** t, Point* P, short n){
    - Polygon po = {n};
    - po.Vers = (Point*)calloc(n, sizeof(Point));
    - for(int i = 0; i < n; i++) po.Vers[i] = P[i];
    - return arrPushback((void**)t, (void*)&po);
  - }



STRUCTURAL ARRAY

- Some struct declarations
  - typedef struct{ int x, y; } Point;
  - typedef struct { short nVer; Point* Vers; } Polygon;
  - void main(){
    - //…
    - if(lst != NULL){
      - //…
      - **PolyListSave**(lst, "Polygons.dat");
      - PolyListFree(lst);
    - }
  - }
  - void **PolyListSave**(Polygon* t, const char* fname){
    - if(t == NULL || fname == NULL) return;
    - FILE* fp = fopen(fname, "wb");
    - if(fp == NULL) return;
    - int nPoly = arrSize(t);
    - for(int i = 0; i < nPoly; i++){
      - short n = t[i].nVer;
      - void* pData = t[i].Vers;
      - fwrite(&n, sizeof(n), 1, fp);
      - fwrite(pData, sizeof(Point), n, fp);
    - }
    - fclose(fp);
  - }

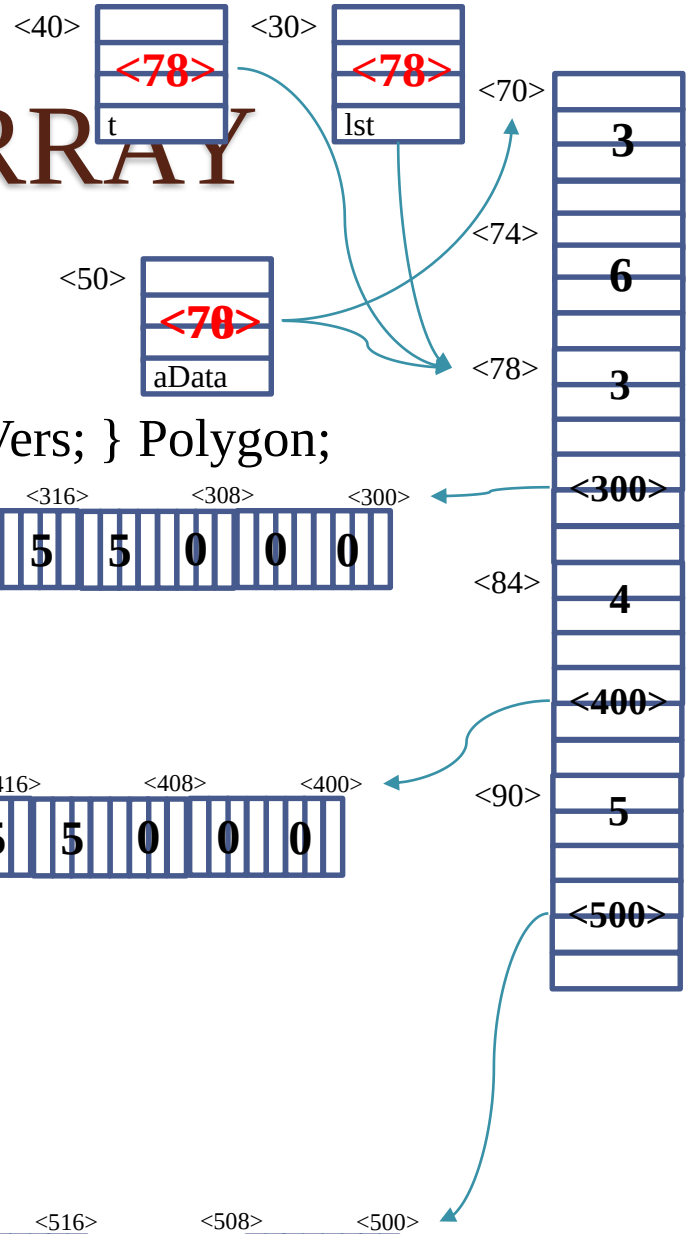


3000555

400055550

50005115550

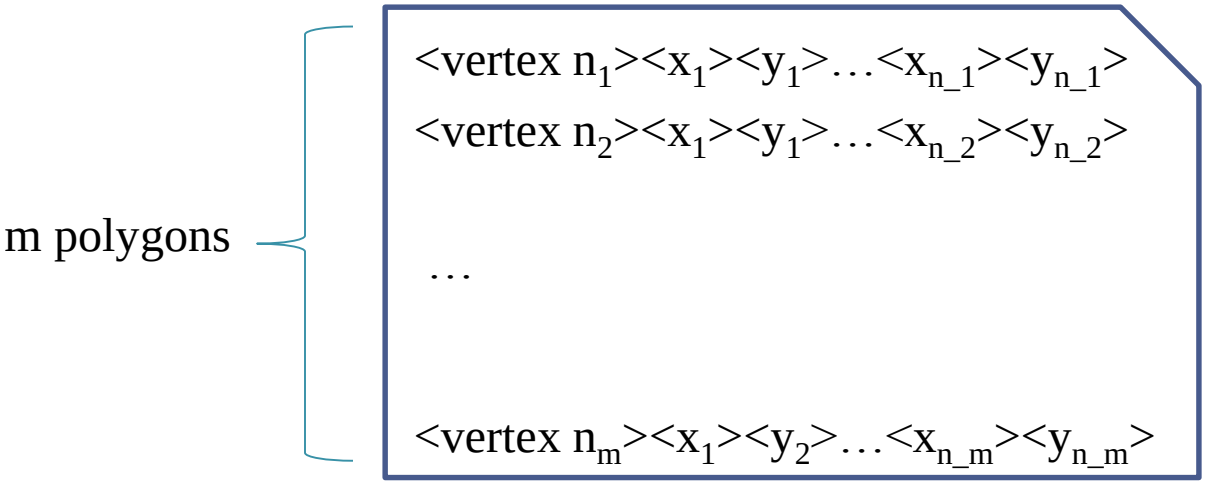
STRUCTURAL ARRAY



- Some struct declarations
  - typedef struct{ int x, y; } Point;
  - typedef struct { short nVer; Point* Vers; } Polygon;
  - void main(){
    - //…
    - if(lst != NULL){
      - //…
      - PolyListSave(lst, "Polygons.dat");
      - **PolyListFree**(lst);
    - }
  - }
  - void **PolyListFree**(Polygon* t){
    - if(t == NULL) return;
    - int n = arrSize(t);
    - for(int i = 0; i < n; i++)
      - if(t[i].Vers != NULL) free(t[i].Vers);
    - arrFree(t);
  - }

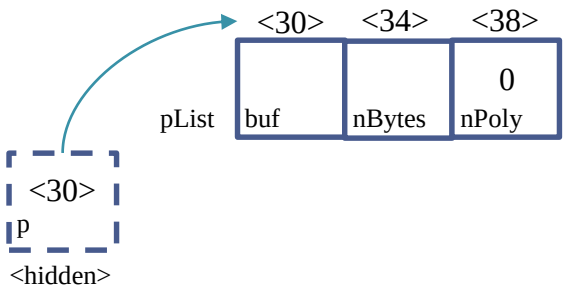
STRUCTURAL ARRAY

- Build a structural array of polygons to **reread binary file just created**
- There are 2 ways:
  - Create new struct with one-member technique
  - Reuse Polygon*



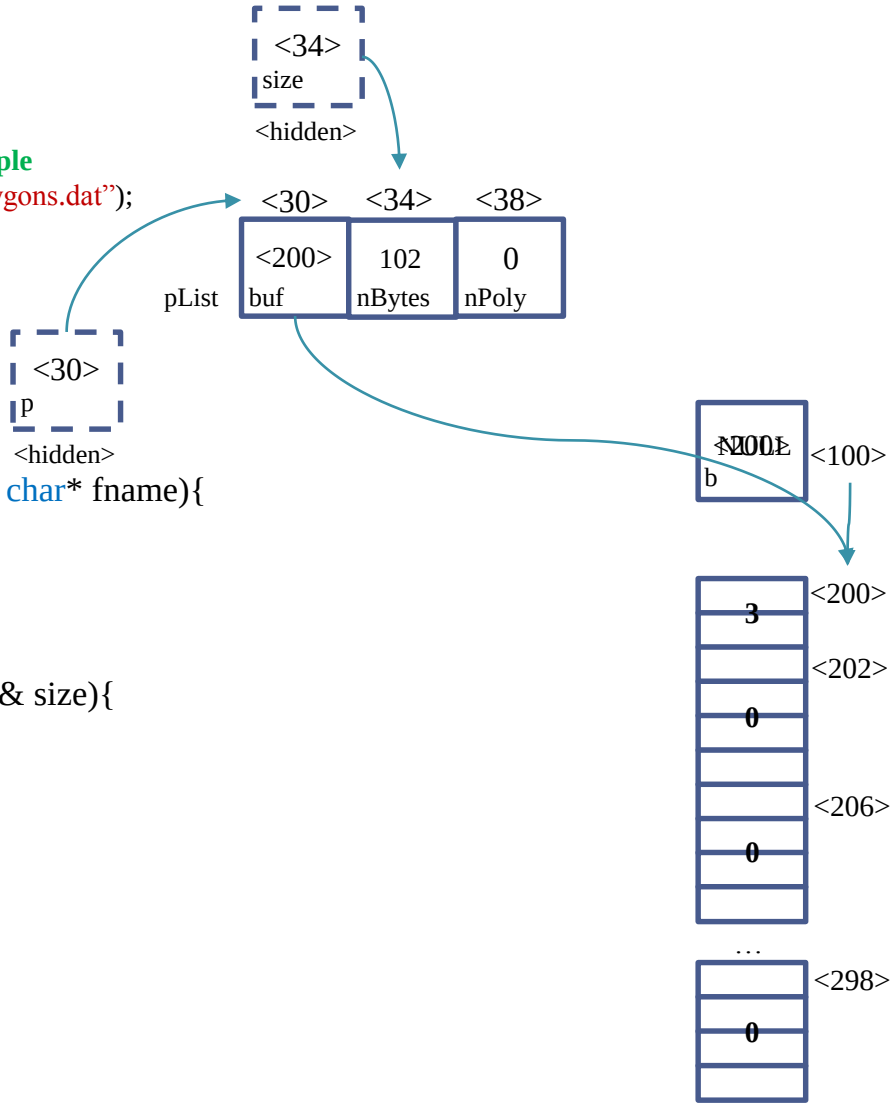
STRUCTURAL ARRAY

- Some struct declarations (method 1)
  - typedef struct { short nVer; Point Vers[1]; } PolygonDat;
  - typedef struct {
    - char* buf; // Data
    - long nBytes; // Size of file
    - int nPoly; // Amount of vertices
    - PolygonDat* operator[](int i){ //… }
  - } PolygonList;
  - void main(){
    - **//portion of creating file as previous example**
    - PolygonList pList; **InitPolyList**(pList, "Polygons.dat");
    - if(pList.buf != NULL){
      - for(int i = 0; i < pList.nPoly; i++){
        - PolygonDat* pg = pList[i];
        - ShowPoly(pg); cout << endl;
      - }
      - free(pList.buf);
    - }
  - }
  - void **InitPolyList**(PolygonList& p, const char* fname){
    - p.nPoly = 0;
    - p.buf = PolyRead(fname, p.nBytes);
    - CountPoly(p);
  - }



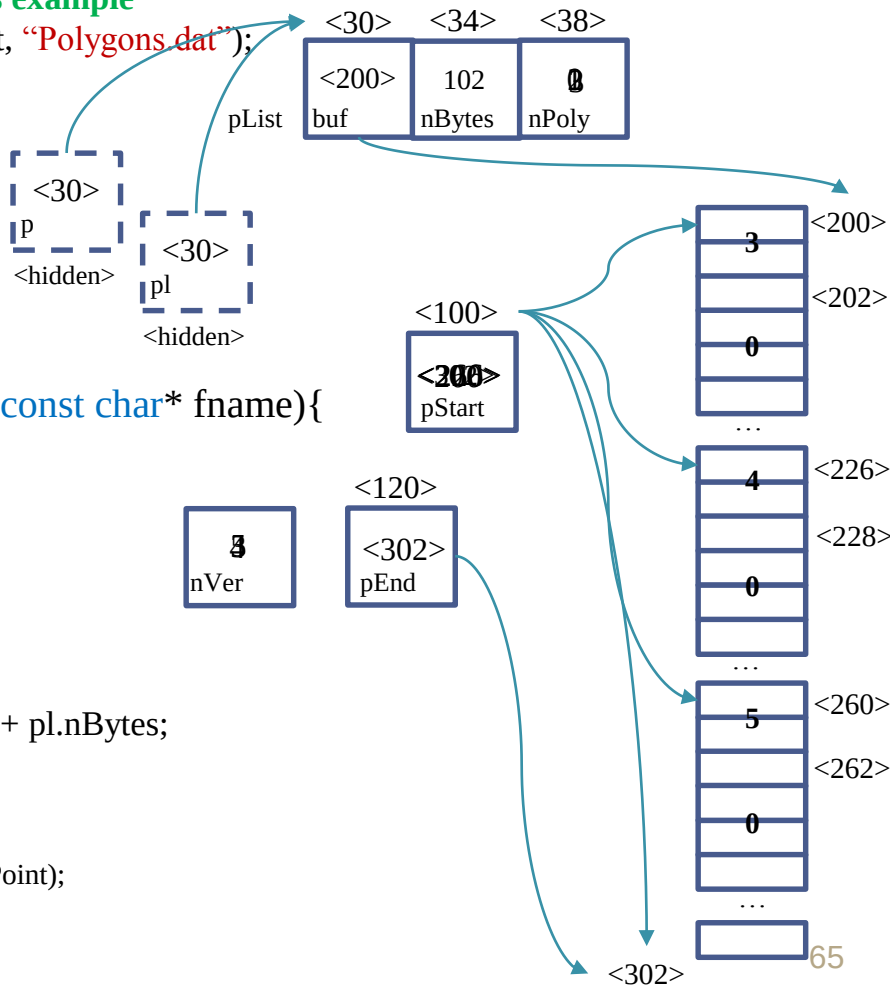
STRUCTURAL ARRAY

- Some struct declarations (method 1)
  - void main(){
    - //portion of creating file as previous example
    - PolygonList pList; **InitPolyList**(pList, "Polygons.dat");
    - if(pList.buf != NULL){
      - for(int i = 0; i < pList.nPoly; i++){
        - PolygonDat* pg = pList[i];
        - ShowPoly(pg); cout << endl;
      - } free(pList.buf);
    - }
  - }
  - void **InitPolyList**(PolygonList& p, const char* fname){
    - p.nPoly = 0;
    - p.buf = **PolyRead**(fname, p.nBytes);
    - CountPoly(p);
  - }
  - char* **PolyRead**(const char* fname, long& size){
    - char* b = NULL;
    - FILE* fp = fopen(fname, "rb");
    - if(fp != NULL){
      - fseek(fp, 0, SEEK_END); size = ftell(fp);
      - fseek(fp, 0, SEEK_SET);
      - b = (char*)malloc(size);
      - if(b != NULL) fread(b, size, 1, fp);
      - fclose(fp);
    - }
    - return b;
  - }



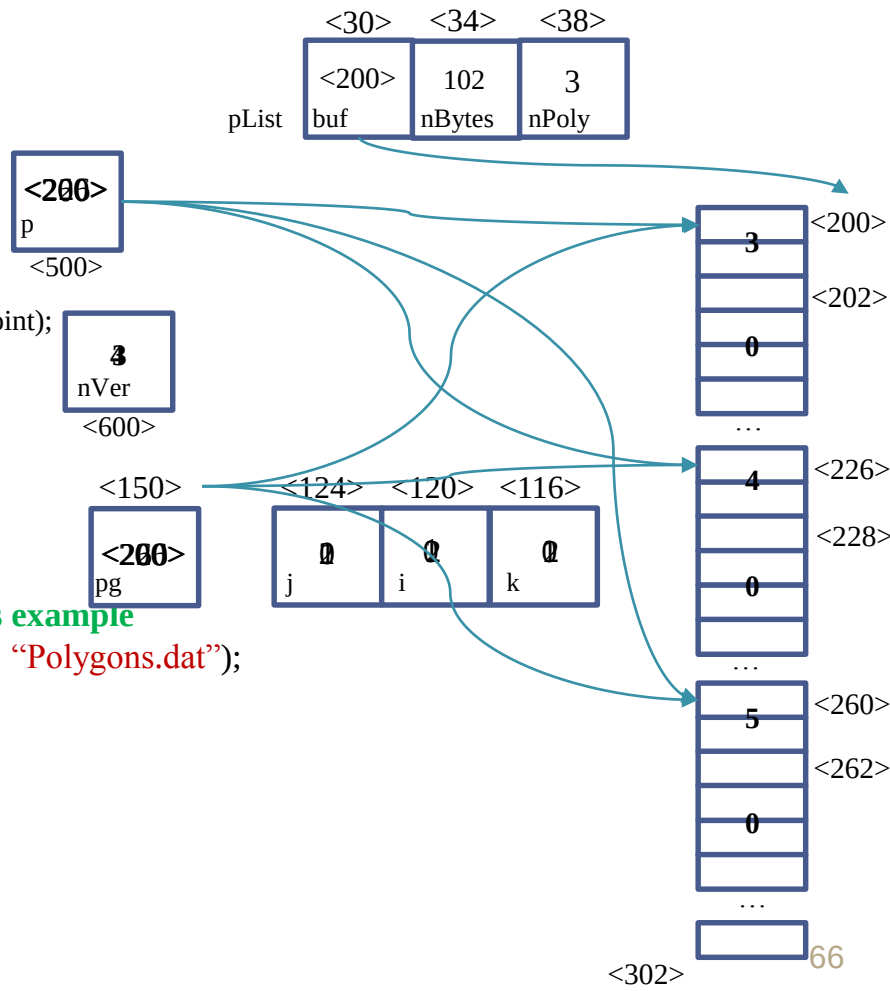
STRUCTURAL ARRAY

- Some struct declarations (method 1)
  - void main(){
    - **//portion of creating file as previous example**
    - PolygonList pList; **InitPolyList**(pList, "Polygons.dat");
    - if(pList.buf != NULL){
      - for(int i = 0; i < pList.nPoly; i++){
        - PolygonDat* pg = pList[i];
        - ShowPoly(pg); cout << endl;
      - }
      - free(pList.buf);
    - }
  - }
  - void **InitPolyList**(PolygonList& p, const char* fname){
    - p.nPoly = 0;
    - p.buf = PolyRead(fname, p.nBytes);
    - **CountPoly**(p);
  - }
  - void **CountPoly**(PolygonList& pl){
    - if(pl.buf == NULL) return;
    - char* pStart = pl.buf, *pEnd = pl.buf + pl.nBytes;
    - while(pStart < pEnd){
      - short nVer = *(short*)pStart;
      - if(nVer > 0) pl.nPoly++;
      - pStart += sizeof(short) + nVer * sizeof(Point);
    - }
  - }



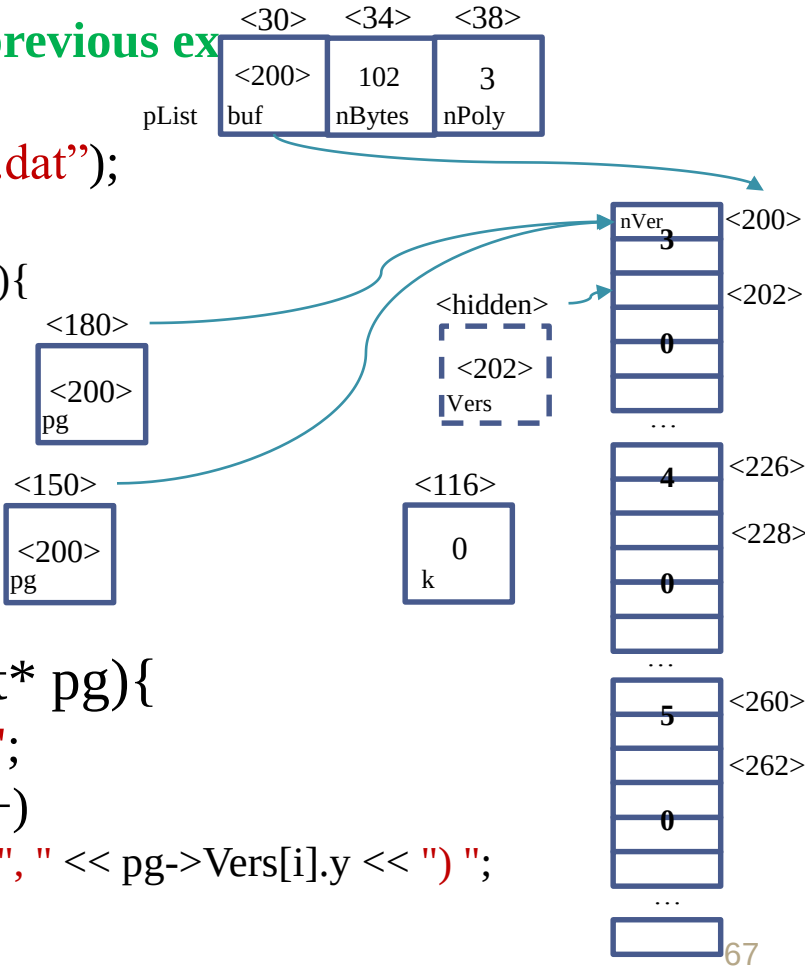
STRUCTURAL ARRAY

- Some struct declarations (method 1)
  - typedef struct { short nVer; Point Vers[1]; } PolygonDat;
  - typedef struct {
    - char* buf; // Data
    - long nBytes; // Size of file
    - int nPoly; // Amount of vertices
    - PolygonDat* operator[](int i){
      - int j = 0; char* p = buf;
      - while(j < i){
        - short nVer = *(short*)p;
        - p += sizeof(short) + nVer * sizeof(Point);
        - j++;
      - }
      - return (PolygonDat*)p;
    - }
  - } PolygonList;
  - void main(){
    - **//portion of creating file as previous example**
    - PolygonList pList; InitPolyList(pList, "Polygons.dat");
    - if(pList.buf != NULL){
      - for(int k = 0; k < pList.nPoly; k++){
        - PolygonDat* pg = pList[k];
        - ShowPoly(pg); cout << endl;
      - }
      - free(pList.buf);
    - }
  - }



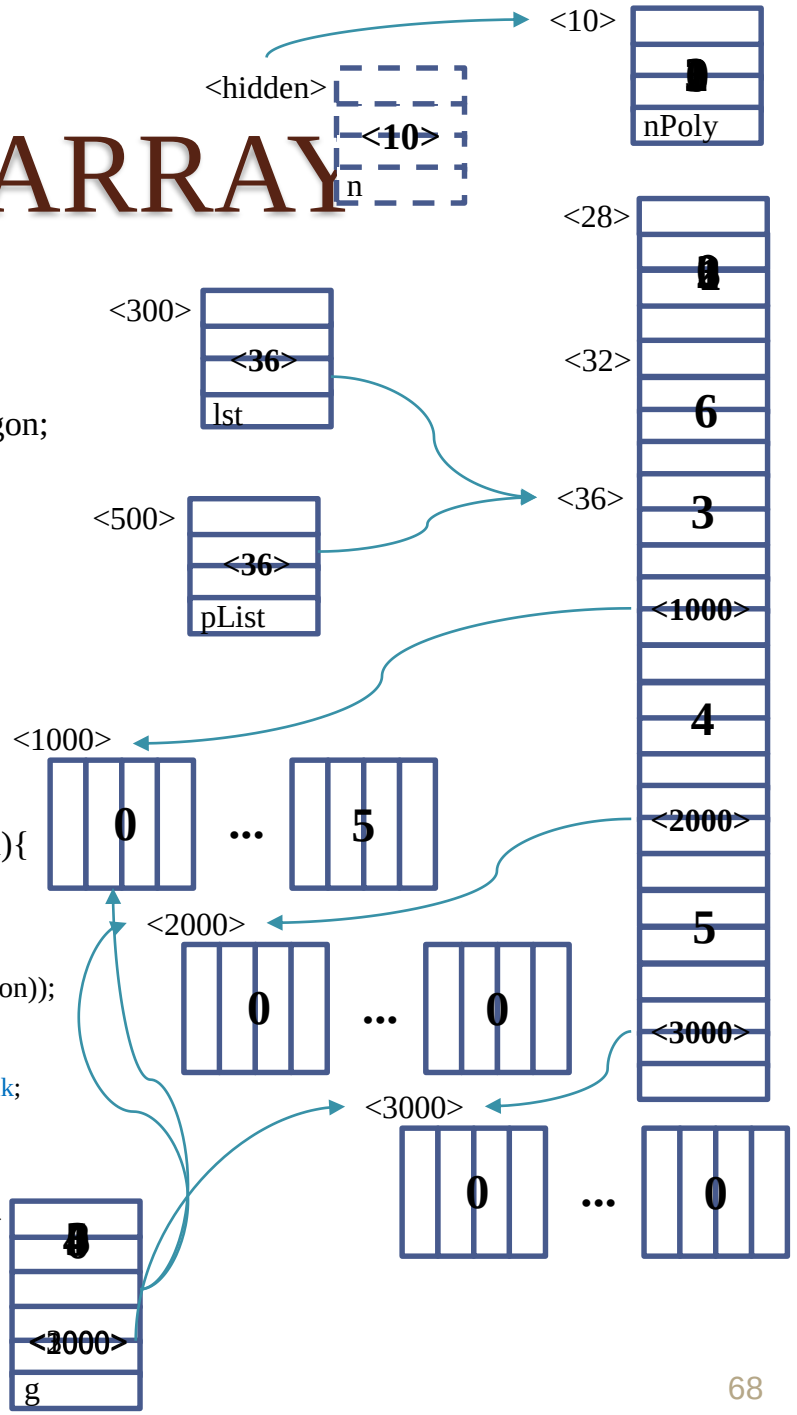
STRUCTURAL ARRAY

- Some struct declarations (method 1)
  - void main(){
    - **//portion of creating file as previous ex**
    - PolygonList pList;
    - InitPolyList(pList, "Polygons.dat");
    - if(pList.buf != NULL){
      - for(int i = 0; i < pList.nPoly; i++){
        - PolygonDat* pg = pList[i];
        - **ShowPoly**(pg); cout << endl;
      - }
      - free(pList.buf);
    - }
  - }
  - void **ShowPoly**(PolygonDat* pg){
    - cout << pg->nVer << " dinh: ";
    - for(int i = 0; i < pg->nVer; i++)
      - cout << "(" << pg->Vers[i].x << ", " << pg->Vers[i].y << ") ";
  - }



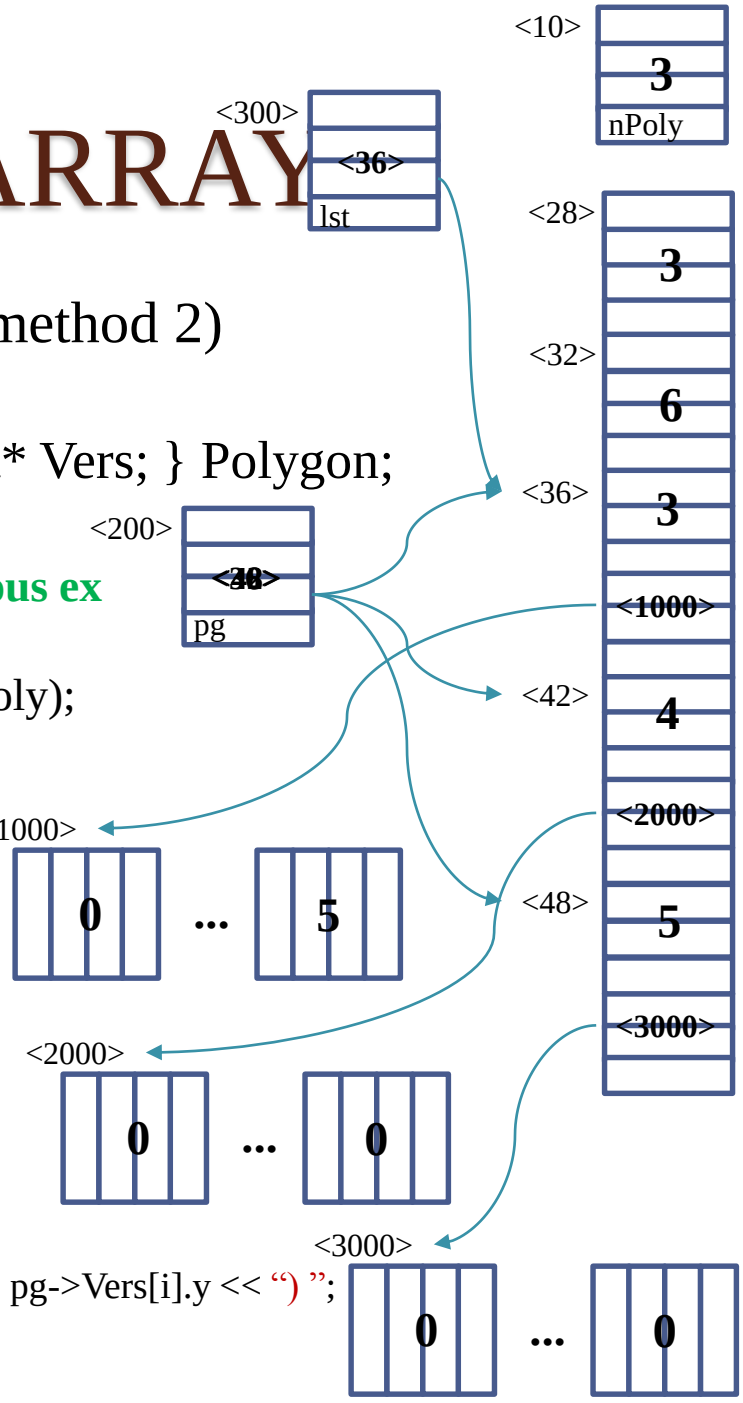
STRUCTURAL ARRAY

- Reuse struct Point & Polygon (method 2)
  - typedef struct{ int x, y; } Point;
  - typedef struct {short nVer; Point* Vers; } Polygon;
  - void main(){
    - **//portion of creating file as previous example**
    - int nPoly;
    - lst = **PolyRead**("Polygons.dat", nPoly);
    - if(lst){
      - for(int i = 0; i < nPoly; i++){
        - ShowPoly(&lst[i]); cout << endl;
      - }
    - } PolyListFree(lst);
  - }
  - Polygon* **PolyRead**(const char* fname, int& n){
    - FILE* fp = fopen(fname, "rb");
    - if(fp == NULL) return NULL;
    - n = 0;
    - Polygon* pList = (Polygon*)arrInit(0, sizeof(Polygon));
    - while(!feof(fp)){
      - Polygon g = {0};
      - if(fread(&(g.nVer), sizeof(g.nVer), 1, fp) <= 0) break;
      - g.Vers = (Point*)malloc(g.nVer * sizeof(Point));
      - if(g.Vers != NULL){
        - n++;
        - fread(g.Vers, g.nVer, sizeof(Point), fp);
        - arrPushback((void**)&pList, (void*)&g);
      - }
    - } fclose(fp);
    - return pList;
  - }



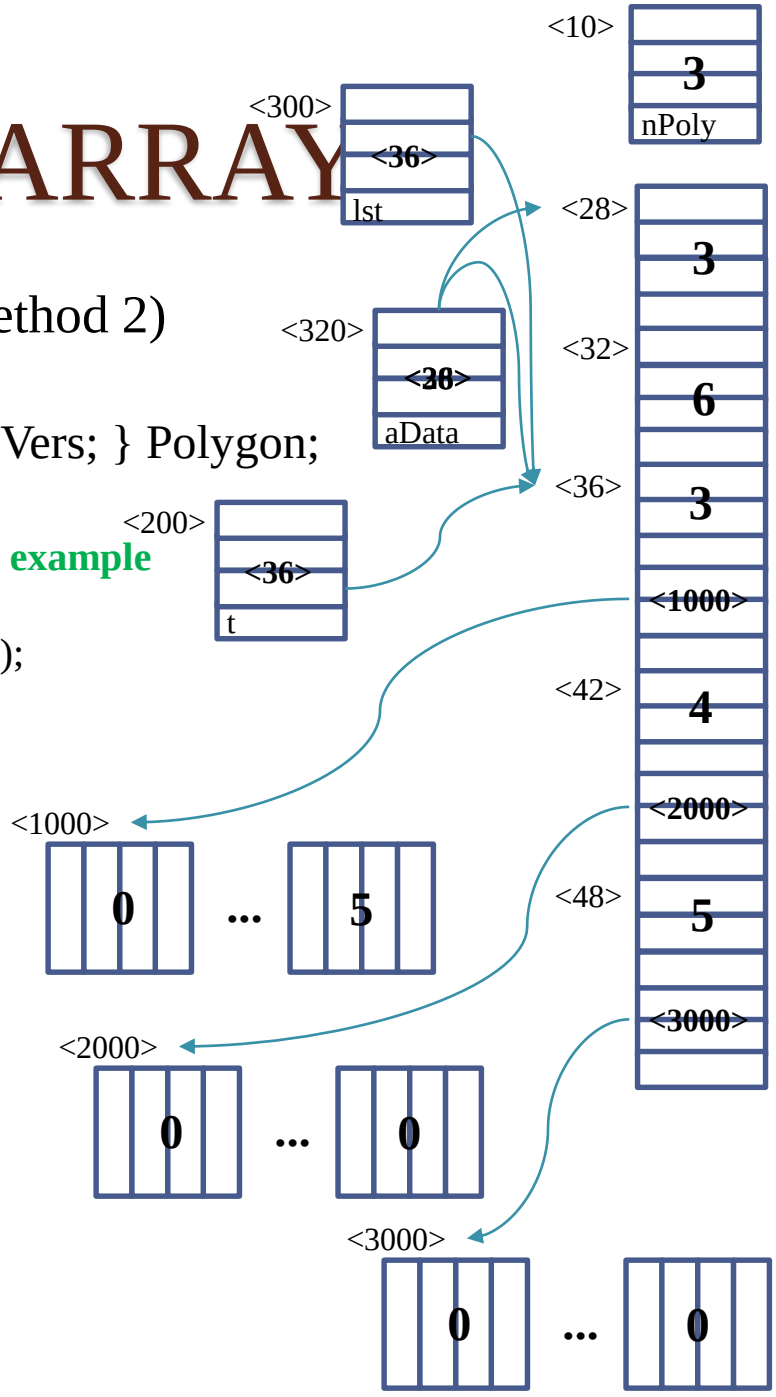
STRUCTURAL ARRAY

- Reuse struct Point & Polygon (method 2)
  - typedef struct{ int x, y; } Point;
  - typedef struct {short nVer; Point* Vers; } Polygon;
  - void main(){
    - **//portion of creating file as previous ex**
    - int nPoly;
    - lst = PolyRead("Polygons.dat", nPoly);
    - if(lst){
      - for(int i = 0; i < nPoly; i++){
        - **ShowPoly**(&lst[i]); cout << endl;
      - }
    - }
    - PolyListFree(lst);
  - }
  - void **ShowPoly**(Polygon* pg){
    - cout << pg->nVer << " vertex: ";
    - for(int i = 0; i < pg->nVer; i++)
      - cout << "(" << pg->Vers[i].x << ", " << pg->Vers[i].y << ") ";
  - }



STRUCTURAL ARRAY

- Reuse struct Point & Polygon (method 2)
  - typedef struct{ int x, y; } Point;
  - typedef struct {short nVer; Point* Vers; } Polygon;
  - void main(){
    - **//portion of creating file as previous example**
    - int nPoly;
    - lst = PolyRead("Polygons.dat", nPoly);
    - if(lst){
      - for(int i = 0; i < nPoly; i++){
        - ShowPoly(&lst[i]); cout << endl;
      - }
    - }
    - **PolyListFree**(lst);
  - }
  - void **PolyListFree**(Polygon* t){
    - if(t == NULL) return;
    - int n = arrSize(t);
    - for(int i = 0; i < n; i++)
      - if(t[i].Vers != NULL) free(t[i].Vers);
    - arrFree(t);
  - }

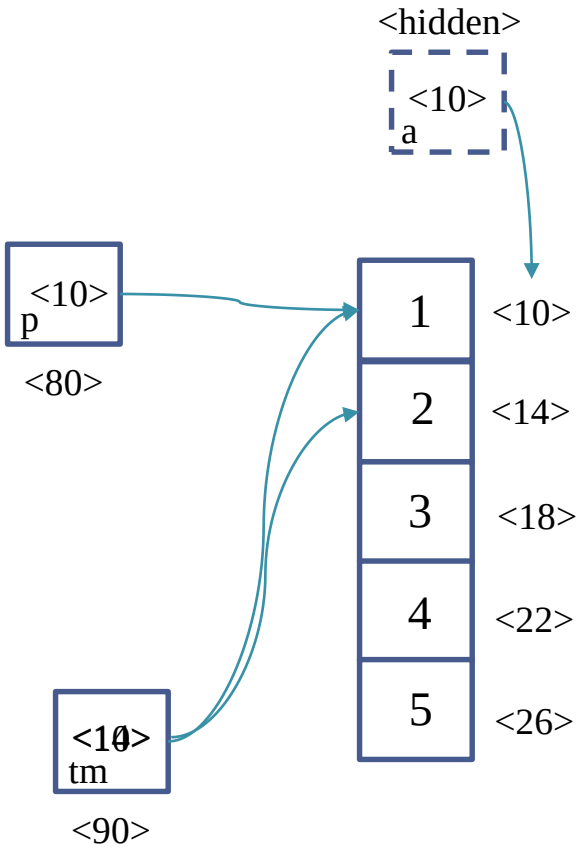


ENHANCED TECNIQUES (VOID* POINTER)

- Be a pointer with datatype of void, for example void* p
- Use in case of unknown datatype in practice
- Disadvantage of this pointer is that we cannot compute with memory address. For example: cannot code p + 1 or *(p – 1) because p with void* cannot ‘jump’
- May use template in C++ to replace

ENHANCED TECNIQUES (VOID* POINTER)

- Example:
  - void main(){
    - int a[] = {1, 2, 3, 4, 5};
    - void* p = a;
    - cout << p[1] << endl; // Wrong
    - cout << *((int*)p + 1) << endl;
    - cout << ((int*)p)[1] << endl;
  - }

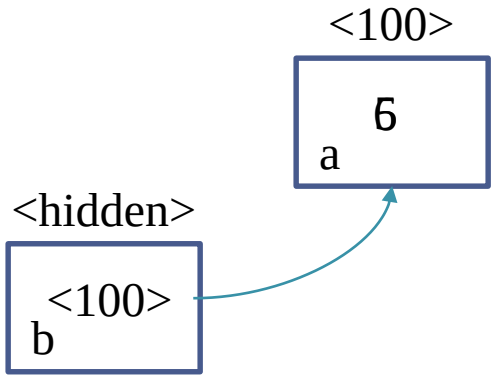


ENHANCED TECNIQUES (REFERENCE)

- A variable referring to another one
- Reference variable needs original variable, for it cannot exist independently
- There are 3 ways to use reference
  - Declare reference variable referring (with same datatype) to original one in the same scope, for example: int a = 5; int& b = a;
  - Be a reference parameter of another function, for example: Func(<datatype>& x)
  - Returned value of another function is a reference variable, for example <datatype>& Func(). Note: original variable must exist when returning a reference (not refer to local variable in function)
- May use reference variable for pointer

ENHANCED TECNIQUES (REFERENCE)

- Declare a reference variable (same datatype) referring to original one in the same scope
- Example
  - void main(){
    - int a = 5;
    - int& b = a;
    - a = 6;
    - cout << “Value of a: ” << a << endl;
    - cout << “Value of b: ” << b << endl;
    - cout << “Address of a: ” << &a << endl;
    - cout << “Address of b: ” << &b << endl;
  - }



ENHANCED TECNIQUES (DATA & CONST POINTER)

- Pointer can “const” the memory of data
  - void main(){
    - char a[] = “Hello world!!!”; const char* pstr;
    - a[0] = ‘h’; // Right
    - pstr = a;
    - pstr[0] = ‘H’; // Wrong
    - a[0] = ‘H’ // Right
    - pstr = “a another string”; // Right
  - }
- pstr can point to another different address
- Note:
  - pstr = (char*)pstr runs OK but nonsense because data cannot be changed with this pointer. Example: pstr[0] = ‘H’ still produce error message.
  - Can cast datatype but must return another different pointer. Example char* tmp = (char*)pstr; This time, we can use tmp to edit string

ENHANCED TECNIQUES (DATA & CONST POINTER)

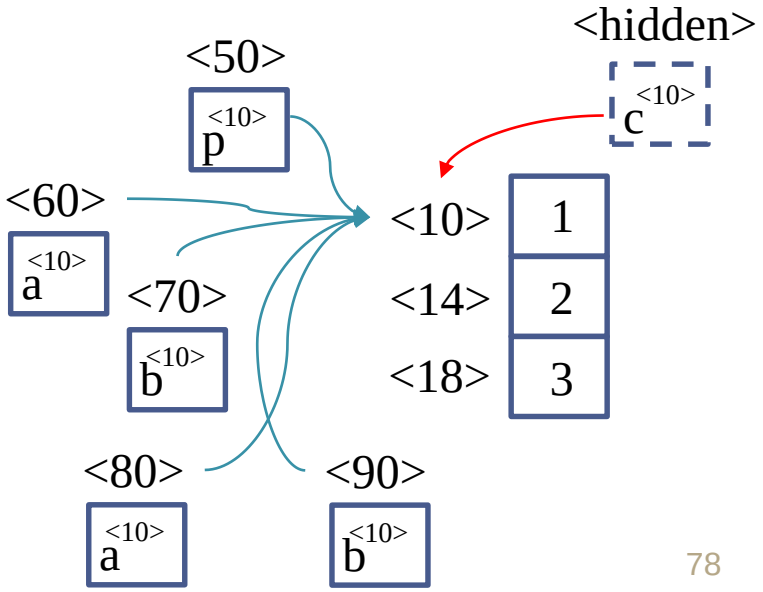
- Const pointer is a pointer only pointing to **one** address when it is initialized and defined
  - void main(){
    - char a[] = "Hello world!!!";
    - char* const pstr = a; //Declare and define simultaneously
    - pstr[0] = 'h'; // Right
    - *(pstr + 1) = 'E'; // Right
    - pstr = pstr + 1: // Wrong
  - }
- Can **change value** of const pointer by using type-casting combined with reference
  - void main(){
    - char a[] = "Hello world!!!"; char* const pstr = a;
    - pstr[0] = 'h'; // Right
    - *(pstr + 1) = 'E'; // Right
    - char* &tmp = (char*&)pstr; tmp++;
    - cout << pstr[0] << endl; // print 'E'
  - }
- Note: directly using pstr **still produce error message**. Example: pstr++ and pstr = (char*)pstr are WRONG

ENHANCED TECNIQUES (DATA & CONST POINTER)

- We may use const for structural/base non-pointer datatype to "const" data
- Example:
  - Point const P = {0, 1}; const Point P = {0, 1};
  - int const a[] = {1, 2}; const int a[] = {1, 2};
- Note: with 2 ways, **a** and **&P** respectively have const int* and const Point*
- Changing values of P.x, P.y and the elements of array a is illegal
- May use type-casting to return a different variable to **change value** of the variables

ENHANCED TECNIQUES (STATIC ARRAY)

- Example of static 1D array
  - int c[] = {1, 2, 3}, *p = c;
- Note:
  - p and c point to one address.
  - The statement "c = new int" is wrong because c is **const pointer**
  - &p ≠ p = c = &c. It can be said that c is a special pointer with hidden address
  - sizeof(p) = sizeof(int*) = 4 ≠ sizeof(c) = 12
- Passing parameter to function
  - void main(){
    - int c[] = {1, 2, 3}, *p = c;
    - funcA(c); funcB(c);
    - funcA(p); funcB(p);
  - }
  - void funcA(int a[]) {//…}
  - void funcB(int* b) {//…}



ENHANCED TECNIQUES (LEFT-VALUE & RIGHT-VALUE)

- Left-value is the left expressions of assignment "=". Example: single var, pointer var, structural var (access by using '.' and '->'), array-element variable (access by using '[]'), pointer var combined with operator '*'
- Returned value can play a role of left-value
  - Returned value is a reference
  - Returned value is base/structural pointer
- Right-value is the right expressions of assignment "=". All left-value expressions can play the roles of right-value expressions, The reverse direction is not sure. For example "&x" is only a right-value expression

ENHANCED TECNIQUES (LEFT-VALUE & RIGHT-VALUE)

- Returned value is reference / base pointer
  - int& maxRef(int& tx, int& ty){
    - if(tx > ty) return tx;
    - return ty;
  - }
  - int* minPtr(int* px, int* py){
    - if(*px > *py) return py;
    - return px;
  - }
  - int x = 99, y = 88;
  - void main(){
    - maxRef(x, y) = 9988;
    - cout << x; // x = 9988
    - *(minPtr(&x, &y)) = 7766;
    - cout << y; // y = 7766
  - }

ENHANCED TECNIQUES (LEFT-VALUE & RIGHT-VALUE)

- Returned value is structural pointer
  - typedef struct { int x, y; } Point;
  - Point P = {16, 10}, Q = {-8, 25};
  - Point* PointMinX(Point* A, Point* B){
    - if(A->x > B->x) return B;
    - return A;
  - }
  - void main(){
    - PointMinX(&P, &Q)->x /= 2; //Q.x = -4
    - *PointMinX(&P, &Q) = P; //Value Q is equal to P's
  - }