POINTER

PROGRAMMING TECHNIQUES

ADVISOR: Trương Toàn Thịnh

CONTENTS

- Using vector<T>
- Pointer
- String datatype
- Memory function
- Operation with pointer
- Pointer techniques
- Exercises

USING VECTOR<T>

- Belong to Standard Template Library (STL)
- Implemented in standard C++
- Being capable of resizing the array-size
- Used with various datatypes
- Being able to declare a nested type for multidimensional array

USING VECTOR<T>

- Example 1

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

USING VECTOR<T>

- Example 1: need to know prior size
- Example 2

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

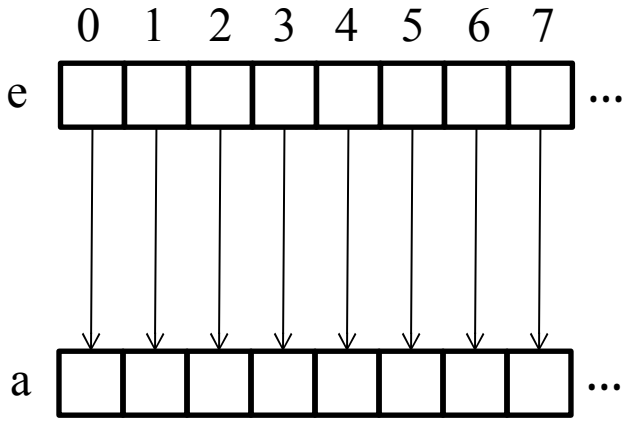
USING VECTOR<T>

- Some basic methods of vector<T>
  - size(): return a current size of array
  - resize(int): change array-size, automatically add/remove the item
  - push_back(T): add an item at the end of array
  - pop_back(): remove the last item of array, array size is decreased by one

USING VECTOR<T>

- Build some basic functions
  - Create a vector from an integer array

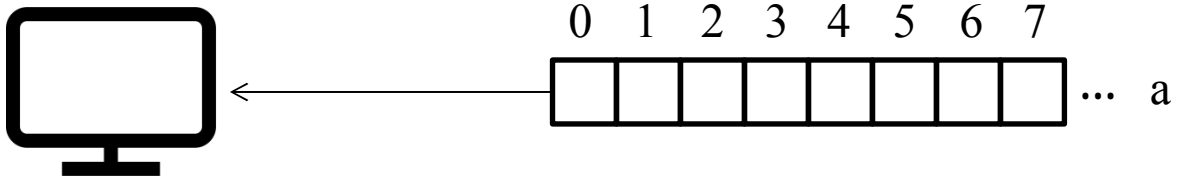
Lines	
1	void intArrayMake(vector<int> &a, int e[], int n){
2	int i = 0;
3	a.resize(0);
4	while(i < n){
5	a.push_back(e[i]);
6	i++;
7	}
8	}



USING VECTOR<T>

- Build some basic functions
  - Print a vector to an output device

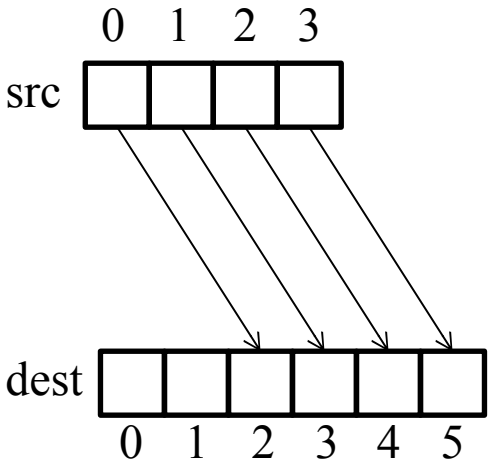
Lines	
1	void intArrayOut(vector<int> &a, ostream& outDev){
2	for(int i = 0; i < a.size(); i++)
3	outDev << a[i] << " ";
4	outDev << endl;
5	}



USING VECTOR<T>

- Build some basic functions
  - Merge two vectors into one

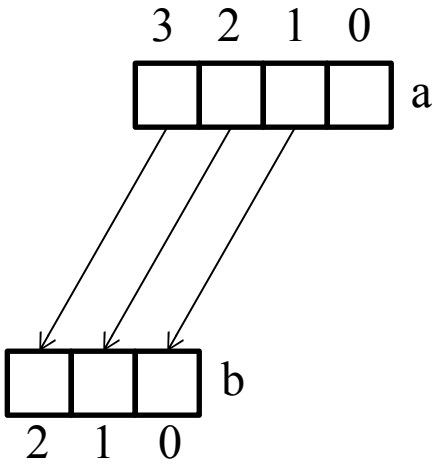
Lines	
1	void intArrayCat(vector<int> &dest, vector<int> &src) {
2	int s1 = dest.size(), s2 = src.size();
3	int idx = s1, newsize = s1 + s2, i = 0;
4	dest.resize(newsize);
5	while(i < s2){
6	dest[idx] = src[i];
7	idx++; i++;
8	}
9	}



USING VECTOR<T>

- Build some basic functions
  - Split a vector from another index and move to another vector

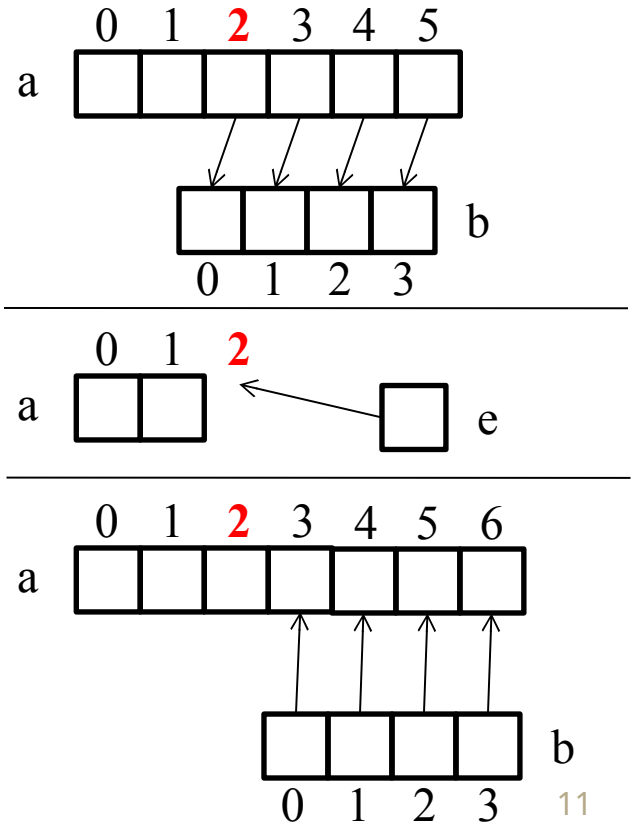
Lines	
1	void intArrayCut(vector<int> &a, int pos, vector<int> &b){
2	int size = a.size(), j = pos;
3	if(j < 0 || j >= size) return;
4	b.resize(0);
5	while(j < size){
6	b.push_back(a[j]);
7	j++;
8	}
9	a.resize(pos);}



USING VECTOR<T>

- Build some basic functions
  - Insert an item into a vector at another index

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



USING VECTOR<T>

- Build some basic functions
  - Function main demonstrate how to use

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

USING VECTOR<T>

- Using struct

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

Operator '>>' leaves character '↵' => Need *cin.ignore* to read '↵' out

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

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

Method *getline* of *cin* reads '↵' out

USING VECTOR<T>

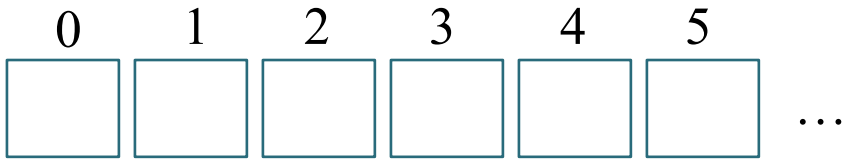
- Using struct

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

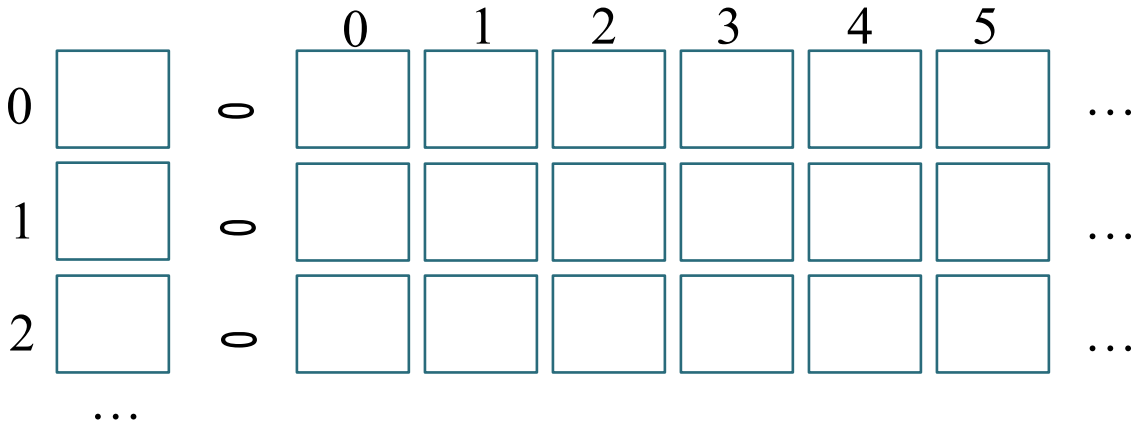
USING VECTOR<T>

- Declare nested type with vector<T> to create 2D array

```
typedef vector<float> Floats;
```




```
typedef vector<Floats> float2D;
```



USING VECTOR<T>

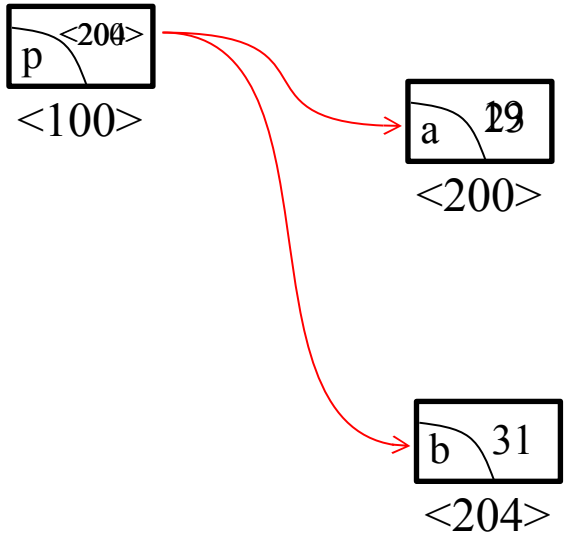
- Declare nested type with vector<T> to create 2D array

Lines		Lines	
1	#include <iostream>	10	void float2DInput(float2D &a){
2	#include <vector>	11	for(int i = 0; i < a.size(); i++)
3	using namespace std;	12	for(int j = 0; j < a[i].size(); j++)
4	typedef vector<float> Floats;	13	cin >> a[i][j];
5	typedef vector<Floats> float2D;	14	}
6		15	void float2DOutput(float2D &a){
7	void float2DInit(float2D &a, int n){	16	for(int i = 0; i < a.size(); i++)
8	a.resize(n);	17	for(int j = 0; j < a[i].size(); j++){
9	for(int i = 0; i < n; i++) a[i].resize(n);	18	cout << a[i][j] << "\t";}
10	}	19	cout << endl;}

POINTER

- Used to store valid memory address
- Also have datatype as normal variables

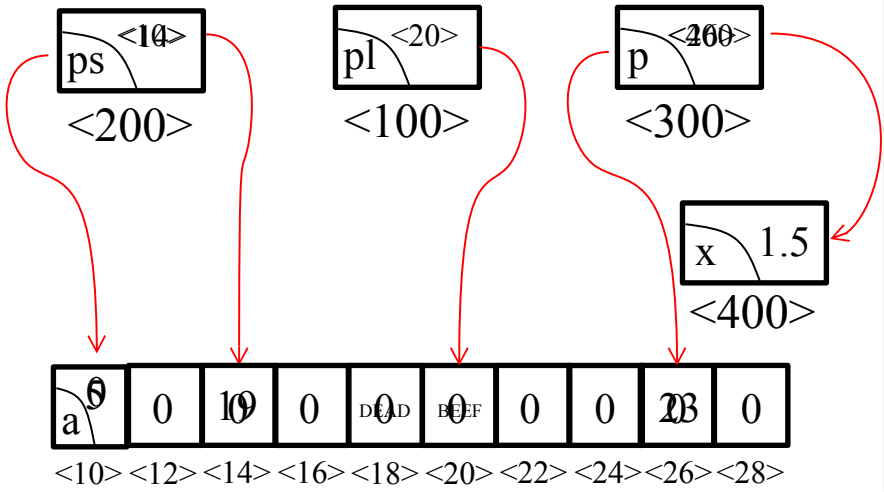
Lines	
1	void main(){
2	int *p;
3	int a = 19, b;
4	p = &a;
5	*p = 23;
6	p = &b;
7	*p = 31;
8	}



POINTER

- Store address of the items of an array
- Can declare a typeless pointer (void*)

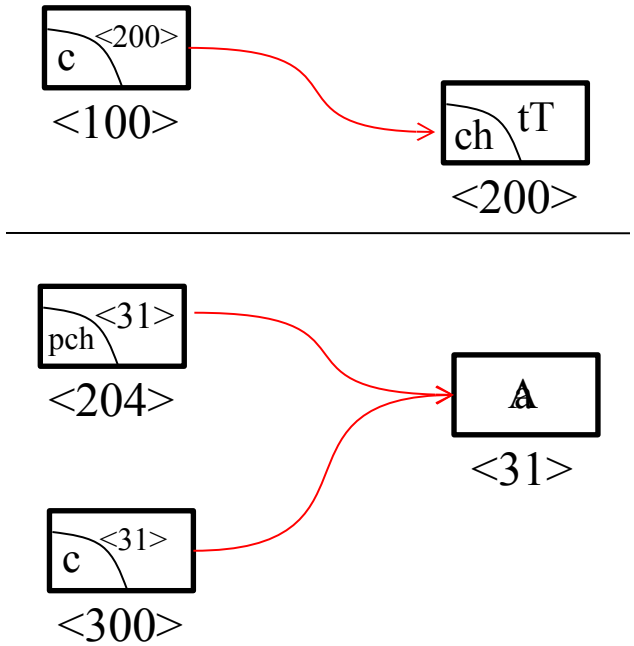
Lines	
1	void main(){
2	float x; unsigned short a[10] = {0};
3	unsigned short *pshort;
4	unsigned long *plong; void* p
5	pshort = a; *pshort = 5;
6	pshort = &a[2]; *pshort = 19;
7	p = &x; *(float*)p = 1.5F;
8	p = &a[8]; *(unsigned short*)p = 23;
9	plong = (unsigned long*)&a[5];
10	*plong = 0xDEADBEEF;}



POINTER

- Function with pointer param has 2 cases:
  - Value-type pointer
  - Reference-type pointer

Lines	
1	void upperCase(unsigned char* c){
2	if(*c >= ‘a’ && *c <= ‘z’)
3	*c = (*c) – 32;
4	}
5	void main(){
6	unsigned char ch, *pch = new unsigned char;
7	scanf(“%c”, &ch); scanf(“%c”, pch);
8	upperCase(&ch); upperCase(pch);
9	printf(“%c”, ch); printf(“%c”, pch)}



POINTER

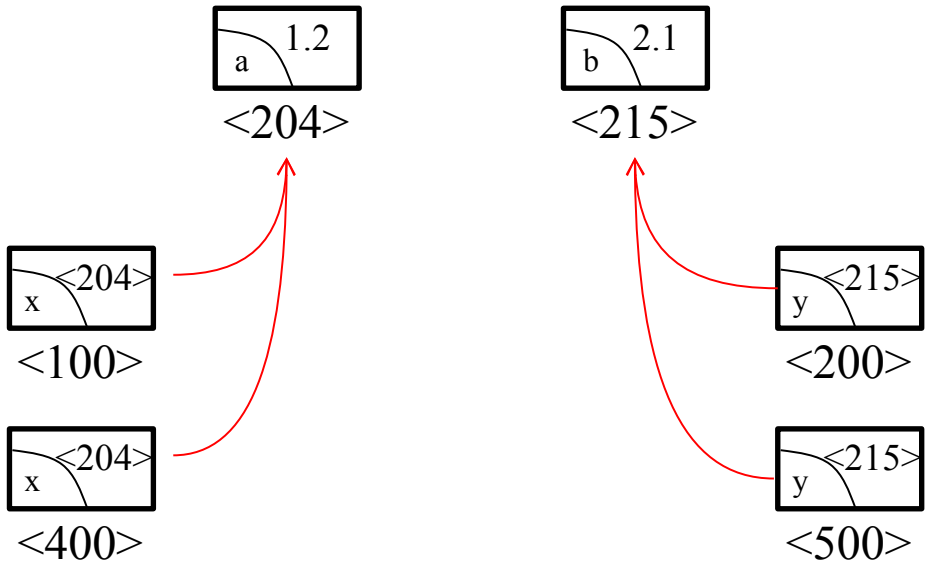
- Function with pointer param has 2 cases:
  - Value-type pointer
  - Reference-type pointer

Lines	
1	void upperCase(unsigned char*& c){
2	if(*c >= ‘a’ && *c <= ‘z’)
3	*c = (*c) – 32;
4	}
5	void main(){
6	unsigned char *pch = new unsigned char;
7	scanf(“%c”, pch);
8	upperCase(pch);
9	printf(“%c”, pch)}

POINTER

- Transmit pointer param to many functions

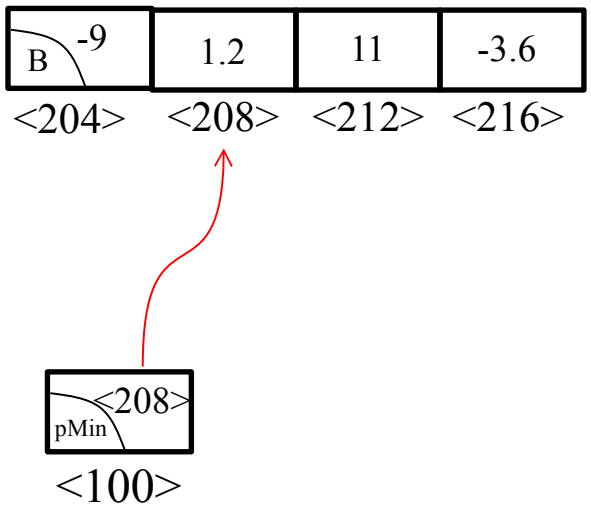
Lines	
1	void swap (float* x, float* y){
2	float u = *x; *x = *y; *y = u;
3	}
4	void adjust(float* x, float* y){
5	if(fabs(*x) > fabs(*y))
6	swap(&(*x), &(*y));
7	}
8	void main(){
9	float a = 1.2F, b = 2.1F;
10	adjust(&a, &b);
11	cout<<*a << *b << endl;
12	}



POINTER

- Return pointer value

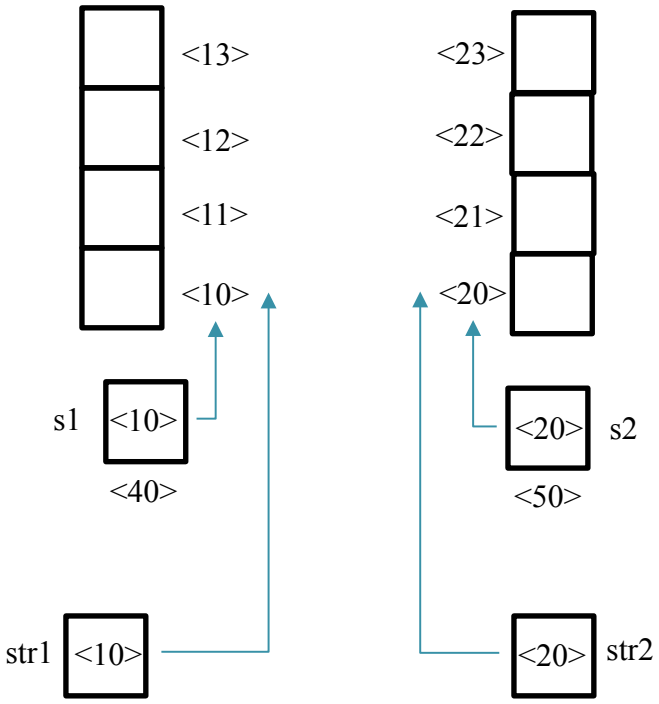
Lines	
1	float* pointerMin (float a[], int n){
2	int i = 1, idx = 0;
3	while(i < n){
4	if(fabs(a[i]) < fabs(a[idx])) idx = i;
5	i++;
6	}
7	return &a[idx];
8	}
9	void main(){
10	float B[] = {-9, 1.2F, 11, -3.6F}; int n = sizeof(B)/sizeof(B[0]);
11	float* pMin = pointerMin(B, n);
12	cout << *pMin << endl;}



POINTER

- Return pointer value: use to interact with the functions in <string>

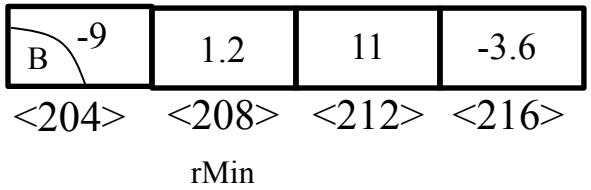
Lines	
1	char* strmax(char* str1, char* str2){
2	if(strcmp(str1, str2) > 0) return str1;
3	return str2;
4	}
5	
6	void main{
7	char* s1 = new char[256], *s2 = new char[256];
8	cin.getline(s1, 256); cin.getline(s2, 256);
9	cout << strupr(strmax(s1, s2));
10	}



POINTER

- Return a reference type

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



STRING DATATYPE

- A 1-D array of items of char
- May use pointer to allocate a string
- Comparison of string datatype uses alphabetical order
- Function can return a pointer (string address)
- Many functions support string manipulation
  - Copy substring of a string
  - Merge two strings
  - Detect substring
  - Count a number of substring
  - ...

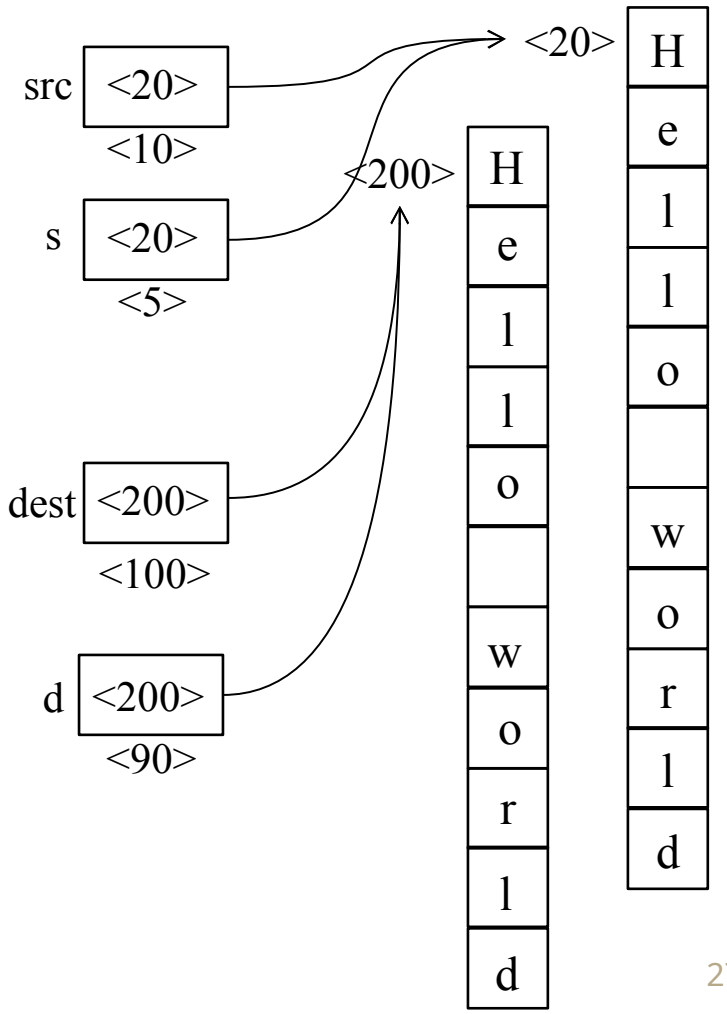
STRING DATATYPE

- String copy:
  - char* d: address of destination string
  - char* s: address of source string
  - char*: address of destination string
- Consider 2 cases
  - The first case: pointer d has an address before coming into function
  - The second case: pointer d does not have an address before coming into function

STRING DATATYPE

- The first case

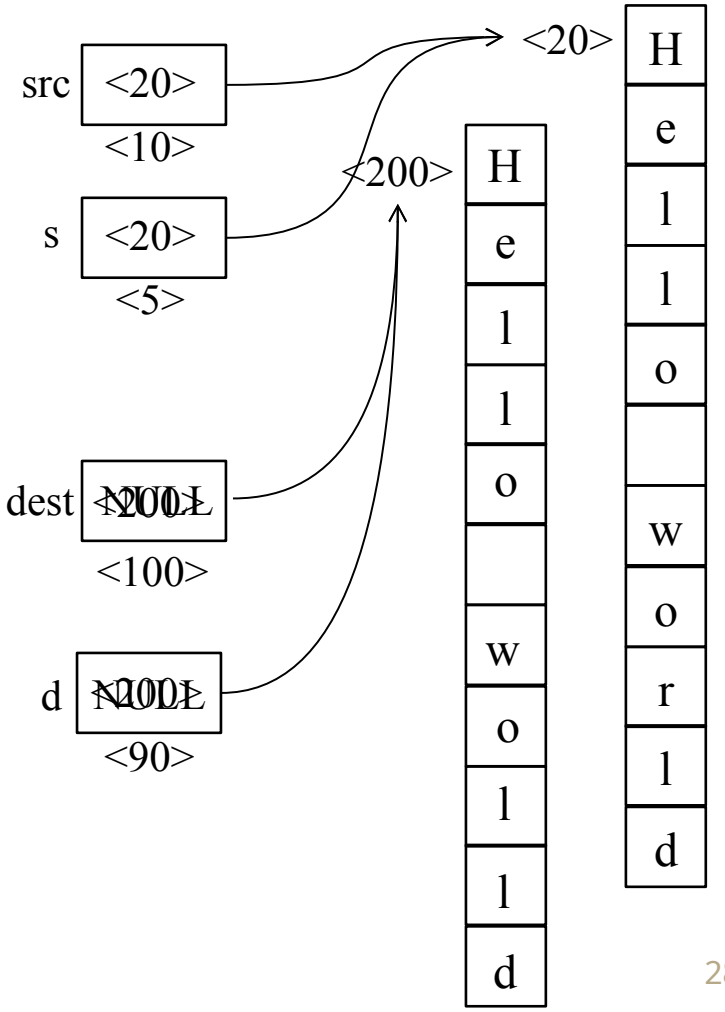
Lines	Description
1	char* strCopyPB1(char* d, char* s){
2	int i, n = strlen(s);
3	for(i = 0; i < n; i++) d[i] = s[i];
4	d[i] = '\0';
5	return d;
6	}
7	void main(){
8	char* src = "Hello world", *dest = new char[12];
9	strCopyPB1(dest, src);
10	cout << dest << endl;
11	delete[] dest;
12	}



STRING DATATYPE

- The second case

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



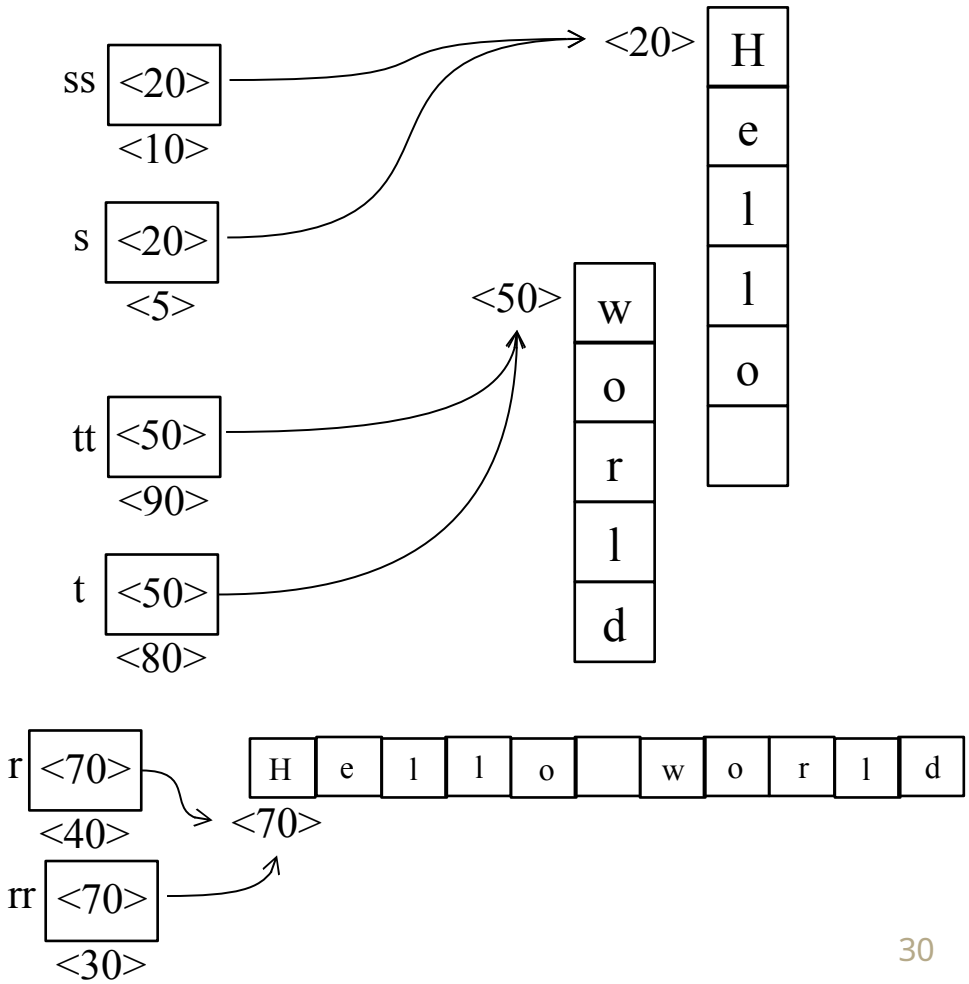
STRING DATATYPE

- Merge 2 strings into one
  - char* s: address of the first string
  - char* t: address of the second string
  - char*: address of destination string
- Note:
  - Length of destination string is equal to the sum of two substrings
  - Add '\0' into the last byte of destination string
  - Need a pointer in main to receive address returned by merge-function

STRING DATATYPE

- Code

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



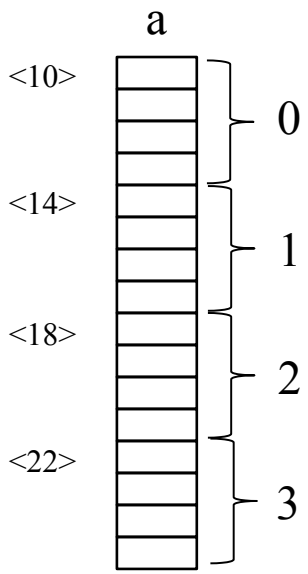
MEMORY FUNCTION

- C/C++ supports some allocation functions
  - void* malloc(int n): allocate n bytes
  - void* calloc(int nItem, int n): allocate nItem item continuously, each item has n bytes
  - void* realloc(void* pmem, int size): re-allocate memory with size either bigger or smaller than old size
  - void free(void* pmem): free memory

MEMORY FUNCTION

- Allocate 1D array

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



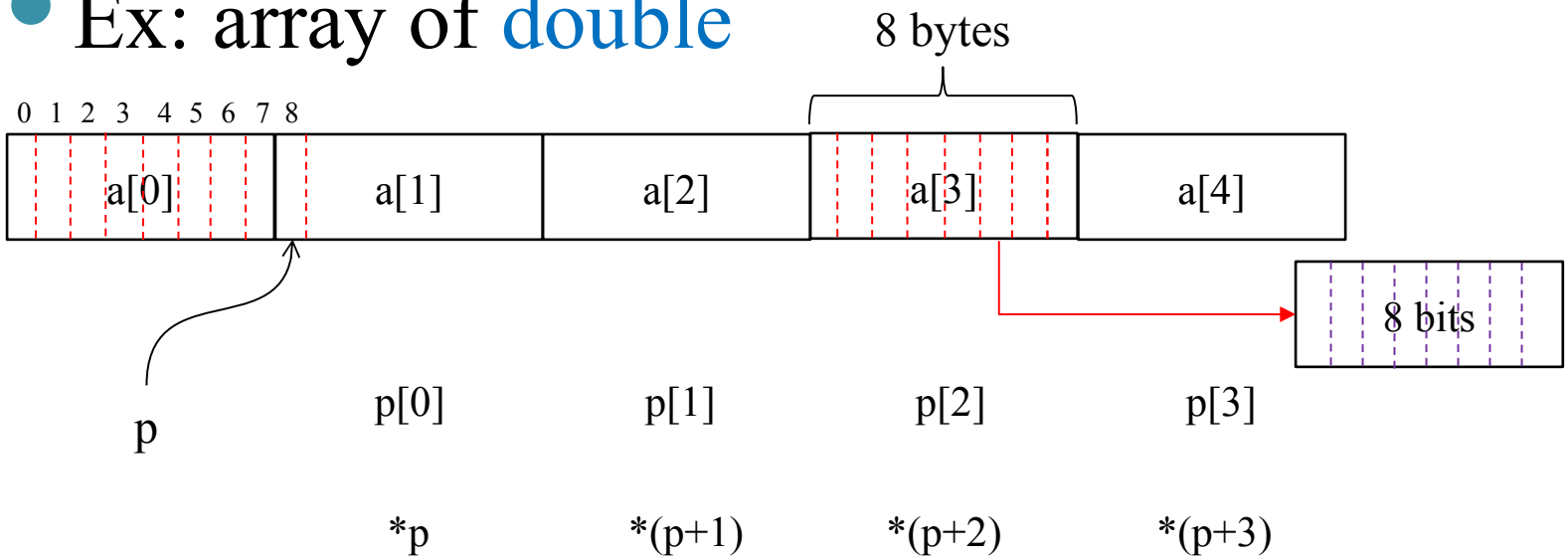
Note:
a = &a[0] and (a + i) = &a[i]
*a = a[0] and *(a + i) = a[i]
Add/subtract address: a + i ≙ sizeof(float)

OPERATION WITH POINTER

- Using + and – for address
  - Addition: address += integer creates a new address different from old address about some bytes depending on datatype and integer
  - Formula: new address = old address += integer *= sizeof(pointer datatype)
  - Subtraction: Pointer 1 – pointer 2 returns deviations depending on datatype
  - Formula: pointer 1 – pointer 2 = $\frac{address1 - address2}{sizeof(type)}$

OPERATION WITH POINTER

- Ex: array of double



  - p + 1 = &a[2]
  - (float*)p + 2 = &a[2], (float*)p + 4 = &a[3]
  - *((float*)p + 4) = the first ½ bytes of a[3]

OPERATION WITH POINTER

- Code

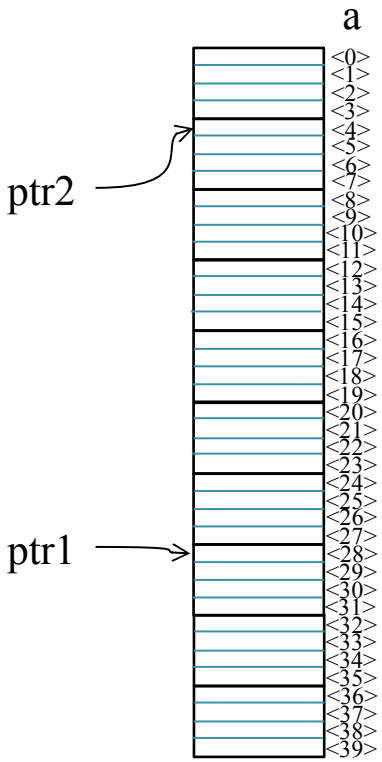
Lines	Description
1	void main (){
2	long a[10];
3	void* ptr1 = &a[7], *ptr2 = &a[1];
4	long d1 = (long*)ptr1 – (long*)ptr2;
5	long d2 = (char*)ptr1 – (char*)ptr2;
6	long d3 = (short*)ptr2 – (short*)ptr1;
7	cout<<“d1 = ” << d1 << endl;
8	cout<<“d2 = ” << d2 << endl;
9	cout<<“d3 = ” << d3 << endl;
10	cout<<“Distance = ” << (long)ptr1 – (long)ptr2 << endl;
11	}

(<28> - <4>)/4

(<28> - <4>)/1

(<4> - <28>)/2

28 - 4



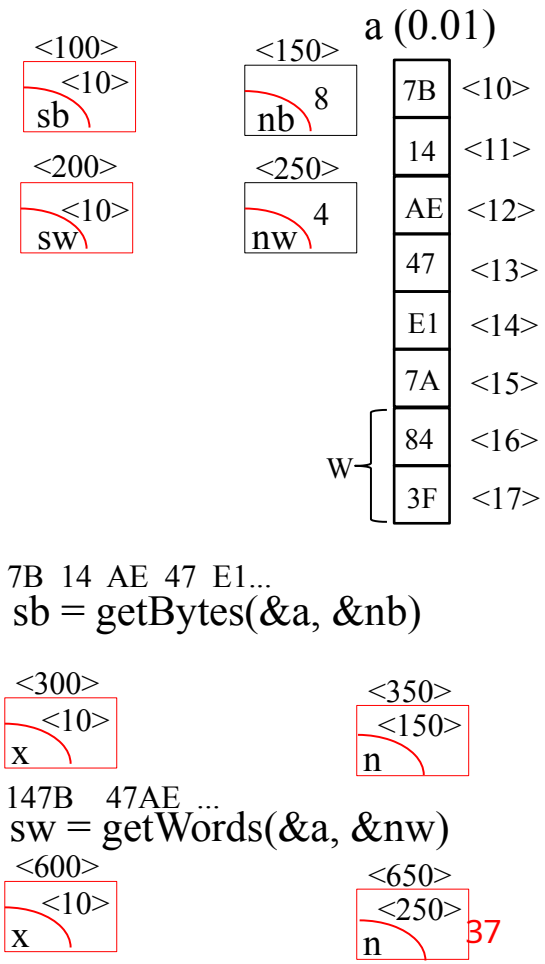
POINTER TECHNIQUES

- Take data physically stored
  - Take each byte/word of a variable
  - May use char*/short* to access each byte/word of basic datatype
  - Syntax
    - (char*)*x* in C or (char*)(&*x*) in C++
    - (short*)*x* in C or (short*)(&*x*) in C++
  - Use "%X" in C or hex in C++ to print hex numbers
  - Technique is similar to dynamic bytes

POINTER TECHNIQUES

- Take data physically stored

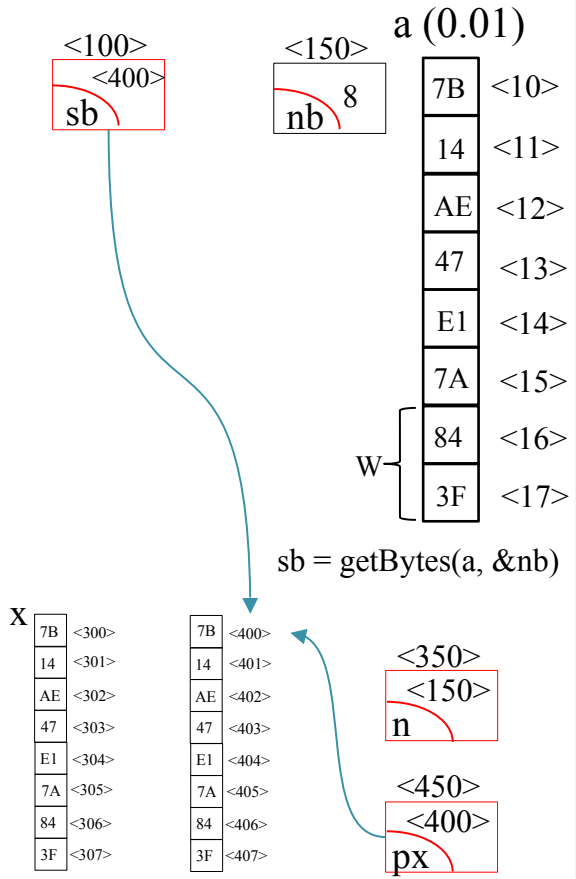
Lines	Description
1	char* getBytes(double* x, int* n){*n = 8; return (char*)x;}
2	short* getWords(double* x, int* n){*n = 4; return (short*)x;}
3	
4	void listBytes(char b[], int nb){
5	for(int i = 0; i < nb; i++) cout << hex << (unsigned char)b[i];
6	cout << endl;
7	}
8	
9	void listWords(short w[], int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



POINTER TECHNIQUES

- Take data physically stored (modifying getBytes)

Lines	Description
1	char* getBytes(double x, int* n){
2	double* px = (double*)malloc(sizeof(double));
3	*n = 8;
4	if(px != NULL) *px = x;
5	return (char*)px;
6	}
9	void main(){
10	double a = 0.01; char* sb; int nb;
11	sb = getBytes(a, &nb);
12	if(sb != NULL){//…; free(sb);}
	}



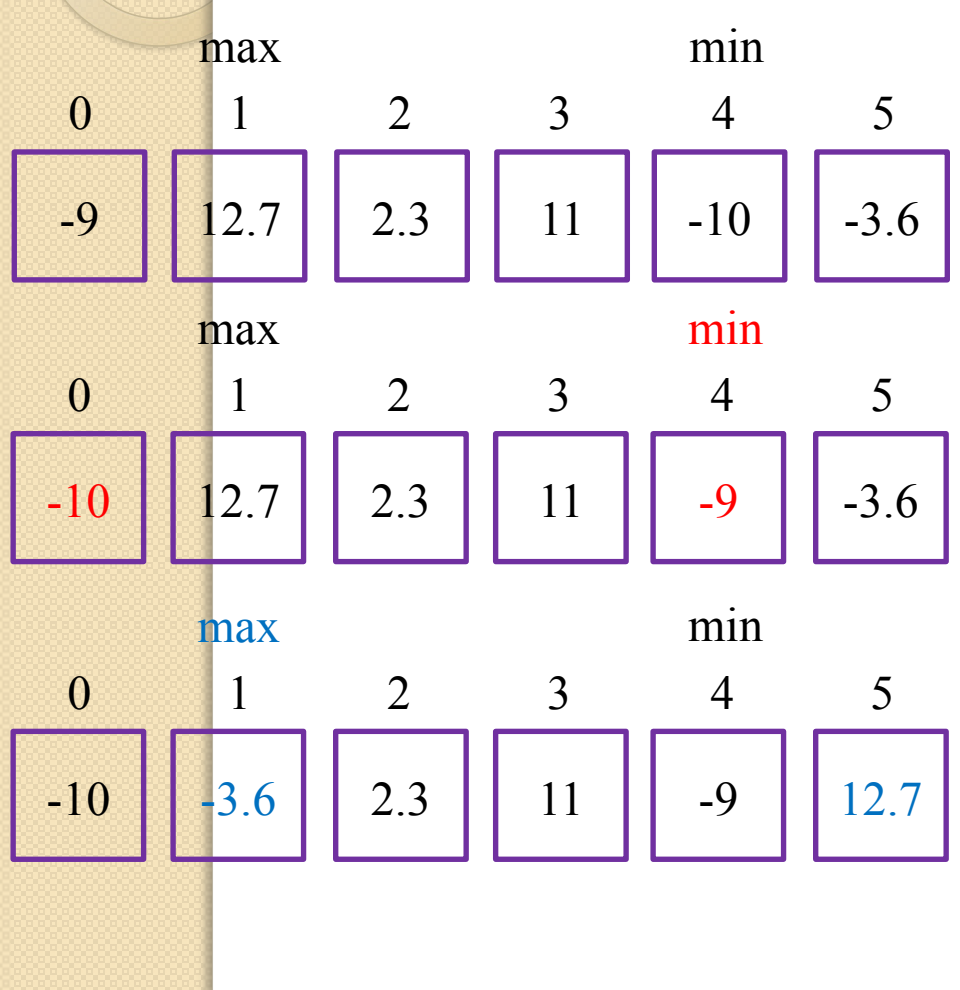
POINTER TECHNIQUES

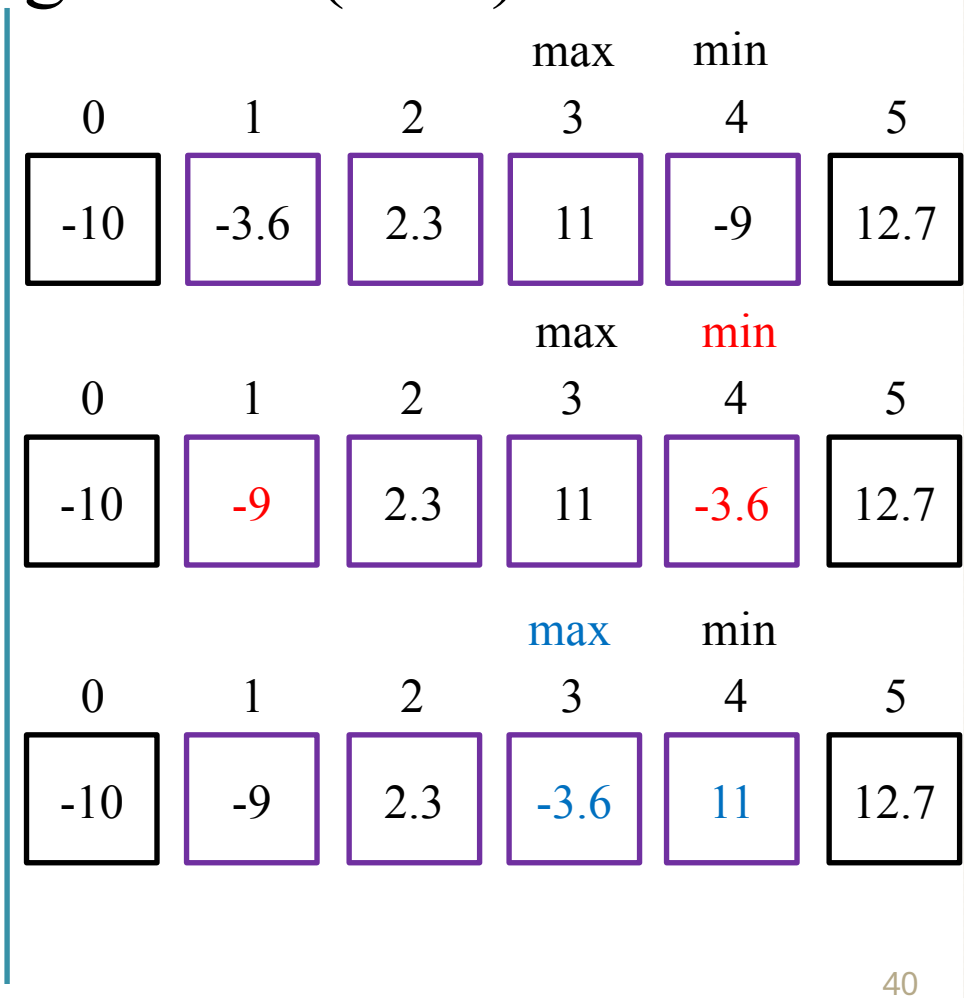
- Directly take sub-array
  - May use a partial array
  - Ex: int a[15] = {…}, *sub_a = &a[5]
  - Easy to see: sub_a[0] = a[5], …, sub_a[9] = a[14]
- Consider recursive sort function

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

POINTER TECHNIQUES

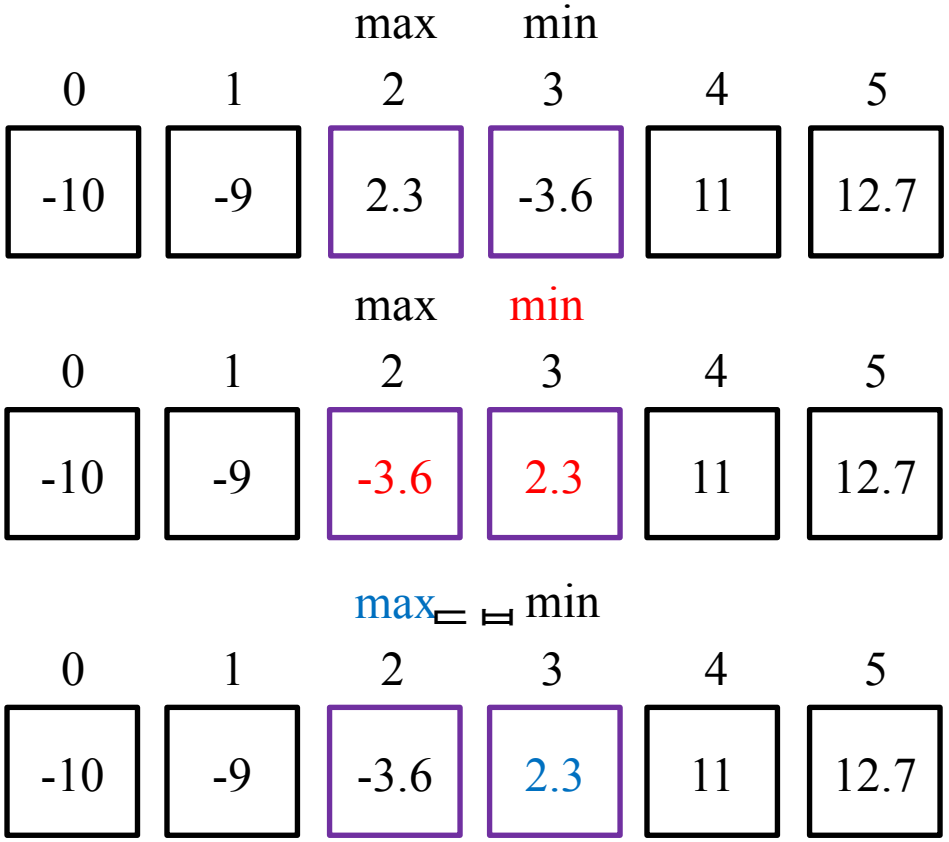
- Recursive sort algorithm (Idea)





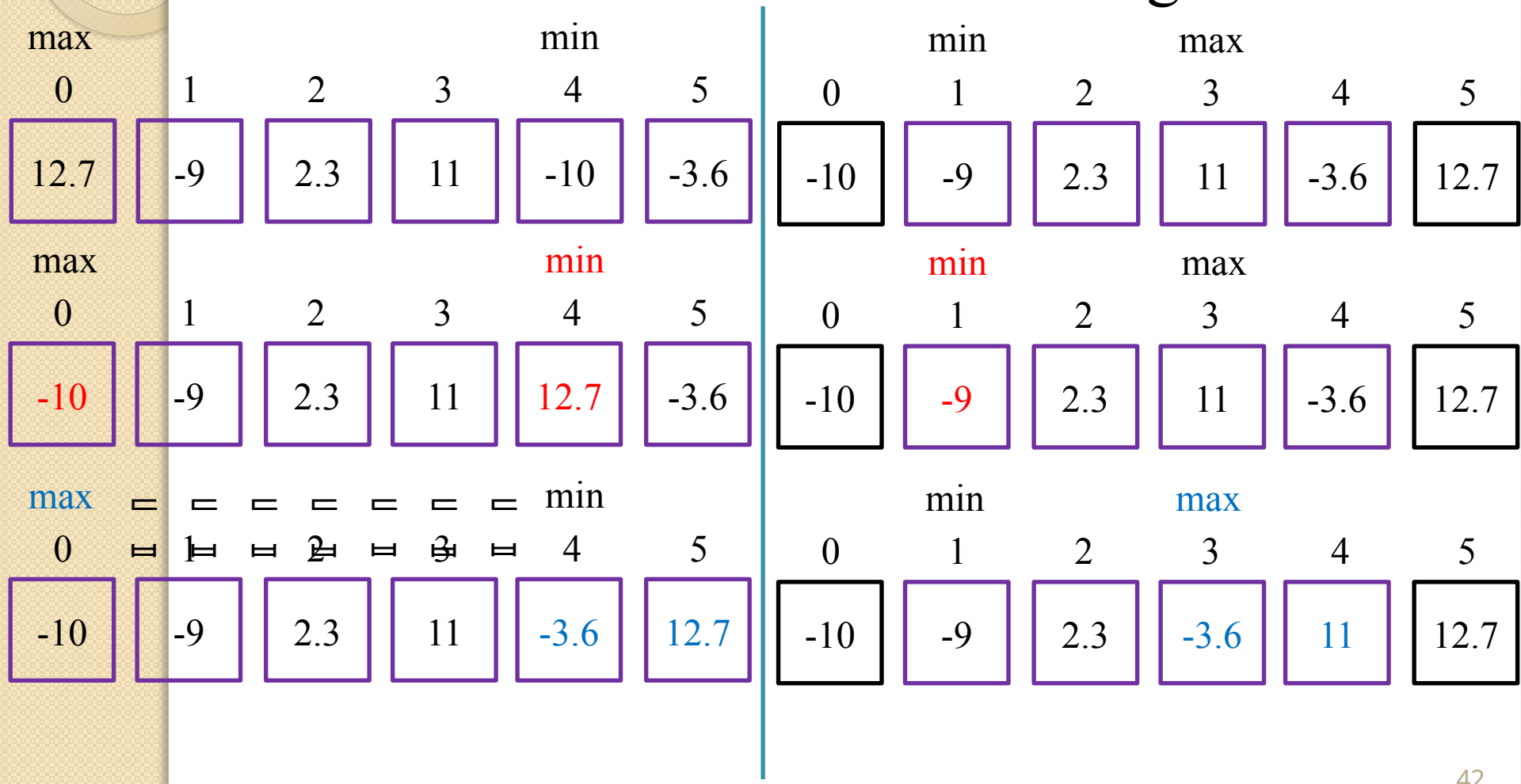
POINTER TECHNIQUES

- Recursive sort algorithm (Idea)



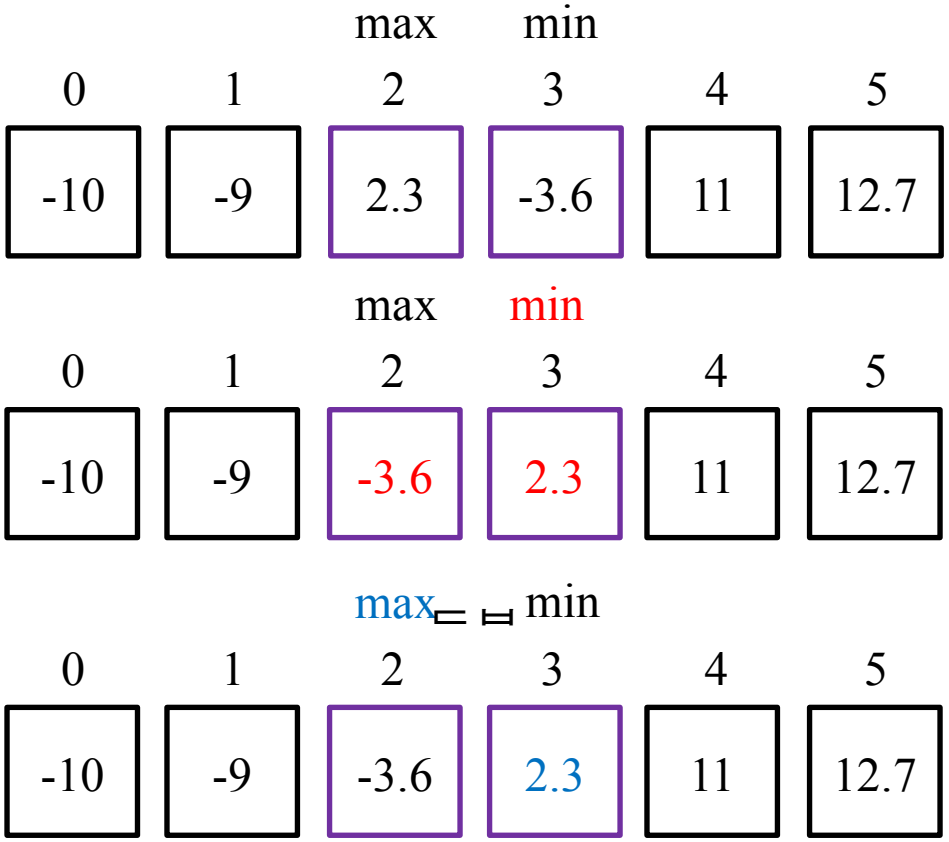
POINTER TECHNIQUES

- Another ex of recursive sort algorithm



POINTER TECHNIQUES

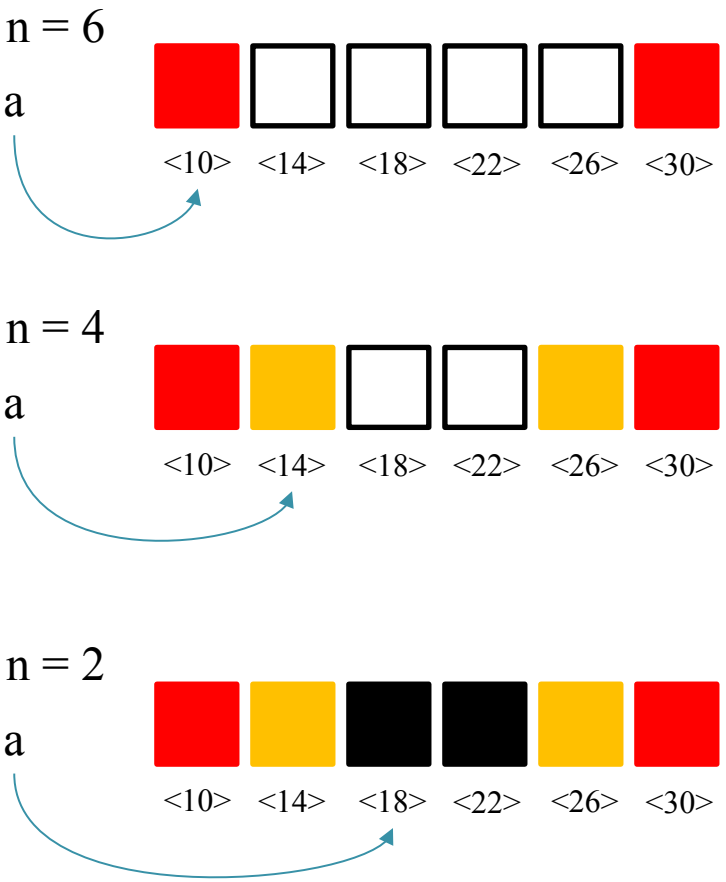
- Another ex of recursive sort algorithm



POINTER TECHNIQUES

- Directly take sub-array

Lines	Description
1	void minMaxSort(float a[], int n){
2	int idmin = 0, idmax = 0;
3	for(int i = 1; i < n; i++){
4	if(a[idmin] > a[i]) idmin = i;
5	if (a[idmax] < a[i]) idmax = i;
6	}
7	swap(&a[0], &a[idmin]);
8	if(idmax == 0) idmax = idmin;
9	swap(&a[n - 1], &a[idmax]);
10	if(n > 3) minMaxSort(&a[1], n – 2);
11	}



POINTER TECHNIQUES

- Initialization with struct variable

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

Note: main() function

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

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

POINTER TECHNIQUES

- Scanning in string
- Pointer datatype must be char

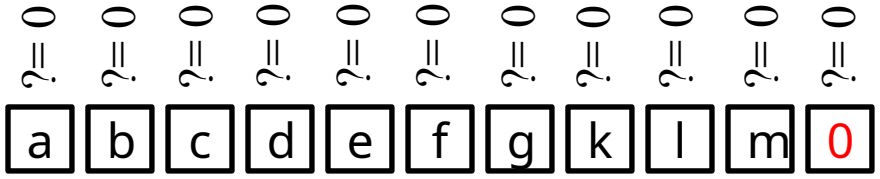
Lines	
1	int strLen(char *s){
2	int len = 0;
3	while(s[len] != '\0') len++;
4	return len;
5	}

len = 10

POINTER TECHNIQUES

- Scanning in string
- Directly increase variable ‘len’ in operator ‘[]’

Lines	
1	int strLen(char *s){
2	int len = 0;
3	while(s[len++] != ‘\0’);
4	return len – 1;
5	}

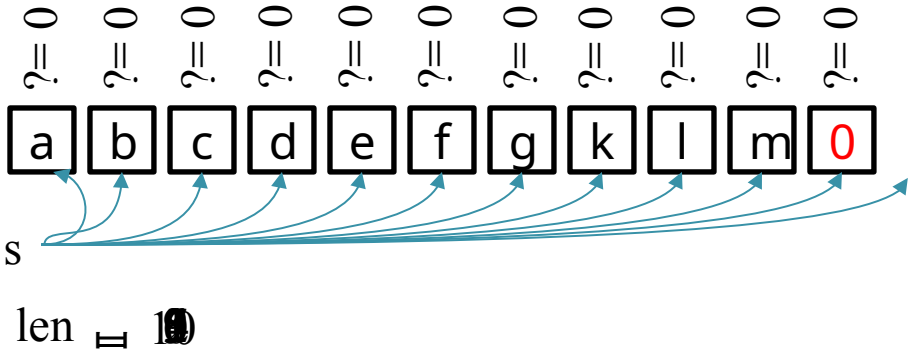


- Note:
  - Line 3 takes s[len], then increase len
  - Line 3 compares s[len] ?= ‘\0’ with “**old len**”

POINTER TECHNIQUES

- Scanning in string
- Using operator * with ++

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

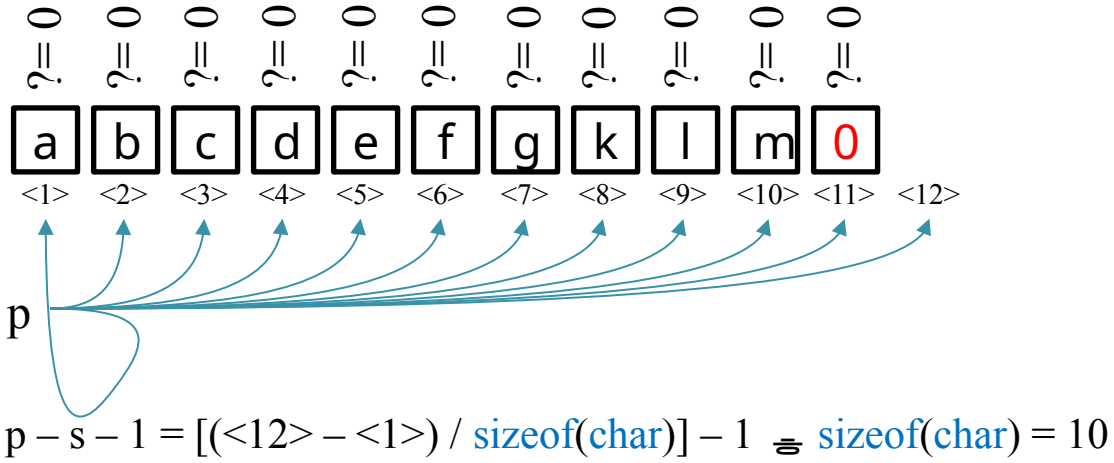


- Note:
  - Line 3 takes *s, then increase address
  - Line 3 compares *s ?= '\0' (**old *s**)

POINTER TECHNIQUES

- Scanning in string
- Utilize address to speed computation

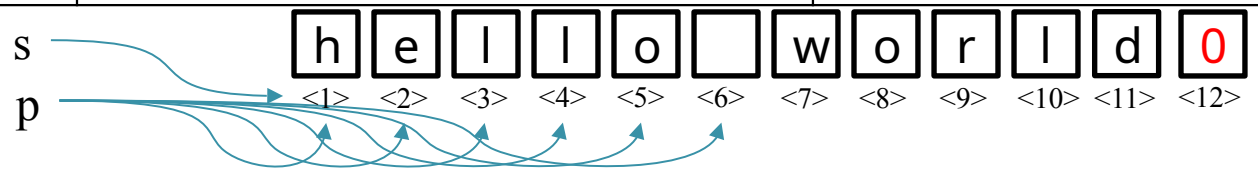
Lines	Description
1	int strLen(char* s){
2	char* p = s;
3	while(*p++);
4	return (p – s – 1);
5	}



POINTER TECHNIQUES

- Searching termination character
- Termination characters may be: space, comma, tab…

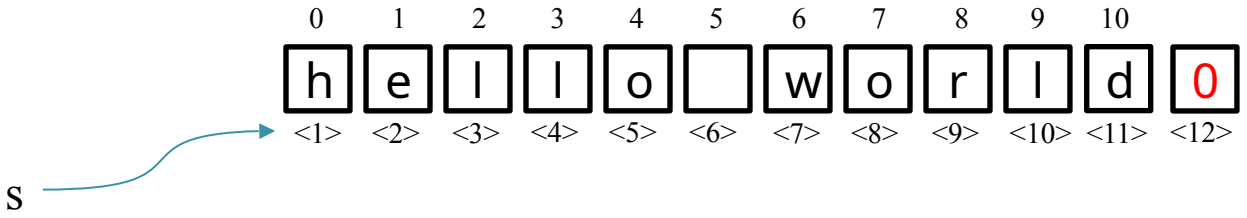
Lines		
1	int isDelimeter(char s){return (s == ' ' || s == ',' || s== '\t' || s == '\n');}	
2	char* delim(char* s){	char* delim(char* s){
3	int i = 0, n = strlen(s);	char* p = s;
4	while(i < n && !isDelimeter(s[i]))	while(*p && !isDelimeter(*p))
5	{i++;}	{p++;}
6	return s + i;	return p;
7	}	}



POINTER TECHNIQUES

- Searching termination character
- Improvement

Lines		
2	char* delim(char* s){	char* delim(char* s){
3	int i = 0;	int i = 0; char c;
4	while(s[i] !=0 && !isDelimeter(s[i]))	while((c = s[i]) != 0 && !isDelimeter(c))
5	{i++;}	{i++;}
6	return s + i;	return s + i;
7	}	}



POINTER TECHNIQUES

- Some rules of using pointer
  - When declaring a pointer, its value is rubbish (address of previous program) ⟹ assign NULL, for example int* p = NULL;
  - Should check pointer value before using, for example if(p != NULL) {…}
  - Pointer at parameter place of another function, this means it had an address at a place of calling function, for example:
    - int abc(int* p){…}; void main(){ abc(new int);}
  - When moving pointer, this should be in its area where pointer manages
  - When moving pointer with addition '+' or subtraction '-' , remember its datatype, for example: pointer int '+' one unit will move 4 bytes, pointer short '+' 2 units will move 4 bytes