STRING AND ALGORITHM

PROGRAMMING TECHNIQUES

ADVISOR: Trương Toàn Thịnh

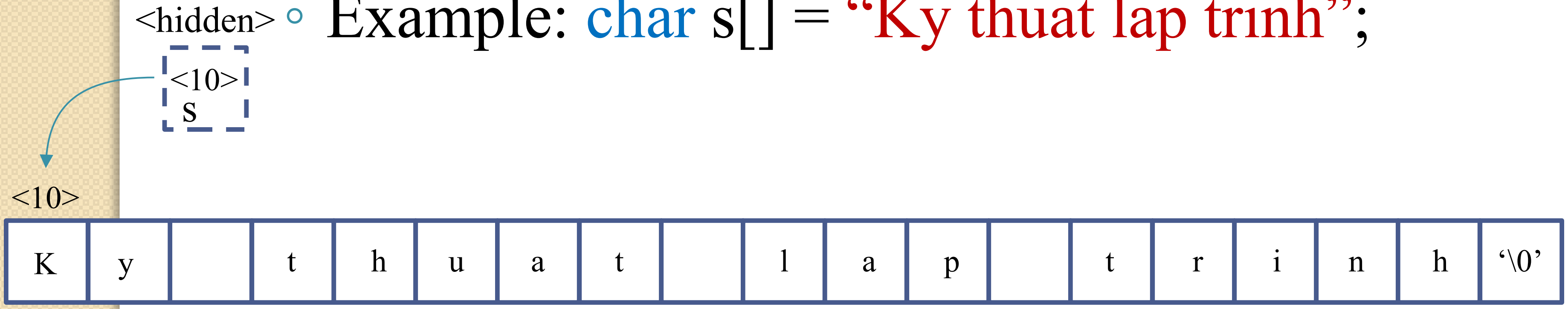
CONTENTS

STRING

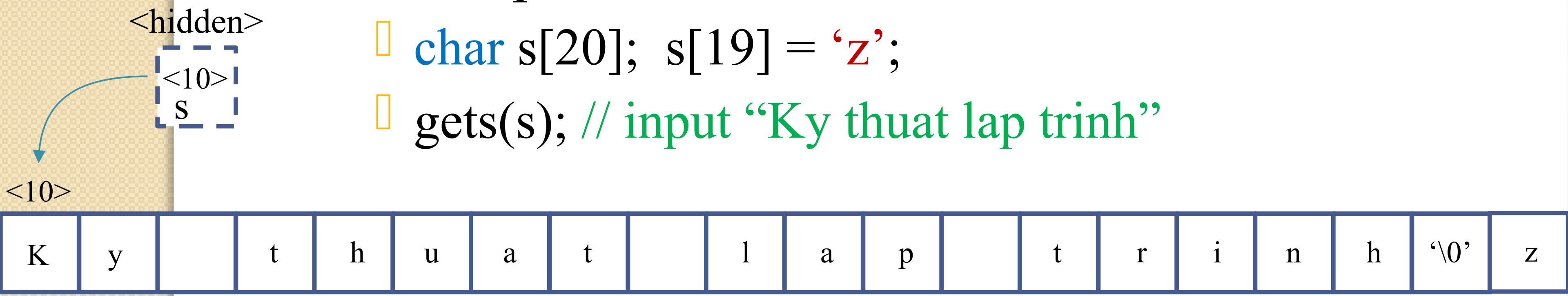
- A basic datatype. For example: email or sms contains the strings
- C/C++ does not have string datatype
- There are 2 ways
  - Implement by using C language
    - Can be used in C++ environment with C-implementation
    - include <string.h> if using more support string functions
    - Array of characters must include '\0' at the end (end-of-string mark)
    - Cannot use operators +, ==, … with character array datatype
  - Using string in STL library of C++
    - Only used in C++
    - Can use operators [], >, < …
    - Include <string> and using namespace std;

SIMPLE OPERATION

- Length of a string
  - Example: char s[] = “Ky thuat lap trinh”;

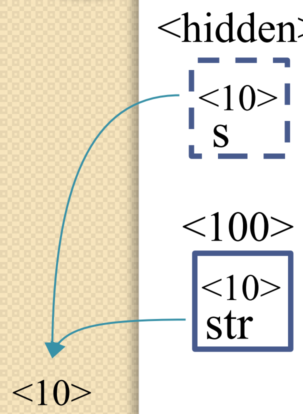


  - Example:
    - char s[20]; s[19] = ‘z’;
    - gets(s); // input “Ky thuat lap trinh”

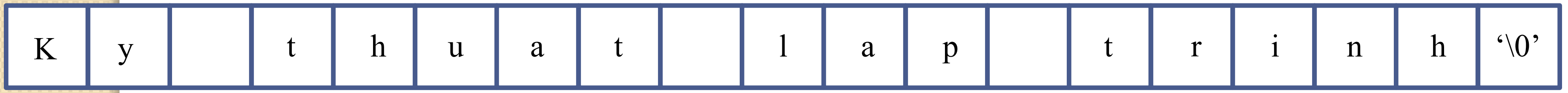


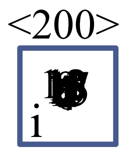
SIMPLE OPERATION

- Length of a string
  - Example:



```
void main(){
    char s[] = "Ky thuat lap trinh";
    cout << StringLength(s);
}
```





```
int StringLength(char str[]){
    int i = 0;
    while(*(str + i) != '\0') i++;
    return i;
}
```

SIMPLE OPERATION

- Alphabetical order

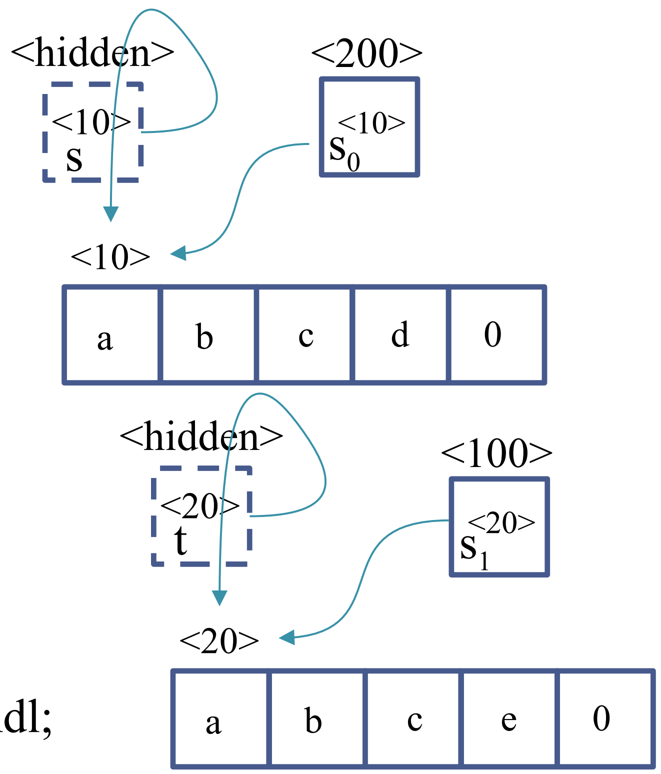
Examples	Explanation
s_0 = "abc" & s_1 = "abd" $s_0 < s_1$	3rd character of s_1 > 3rd character of s_0
s_0 = "abc" & s_1 = "abcd" $s_0 < s_1$	String s_0 and string s_1 are the same at the first 3 characters, string $s_1 > s_0$ due to longer than s_0
s_0 = "abc" & s_1 = "d" $s_0 < s_1$	Due to 1st character of s_1 > 1st character of s_0 so $s_1 > s_0$ although shorter

- String comparison algorithm s_0 & s_1
  - Step 0: $n_0 \leftarrow |s_0|$ & $n_1 \leftarrow |s_1|$
  - Step 1: $n \leftarrow \min\{n_0, n_1\}$
  - Step 2: $i \in \{0, 1, \ldots, n - 1\}$
    - If $s_0[i] > s_1[i]$ then $s_0 > s_1$ & stop
    - If $s_0[i] < s_1[i]$ then $s_0 < s_1$ & stop
  - Step 3:
    - If $n_0 > n$ then $s_0 > s_1$ & stop
    - If $n_1 > n$ then $s_0 < s_1$ & stop

SIMPLE OPERATION

- Example of string comparison s_0 & s_1

```
int CompareString(char* s0, char* s1){
    int n0 = strlen(s0), n1 = strlen(s1);
    int n = (n0 < n1) ? n0 : n1;
    for(int i = 0; i < n; i++){
        if(s0[i] > s1[i]) return 1;
        else if(s0[i] < s1[i]) return -1;
    }
    if(n0 > n) return 1;
    if(n1 > n) return -1;
    return 0;
}
void main(){
    char s[] = "abcd", t[] = "abce";
    cout << CompareString(s, t) << endl;
}
```



SIMPLE OPERATION

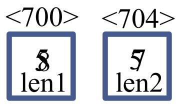
- Remind of const string
  - Const string is a string with fixed value, unchangeable value
    - Example: “abcd” is a const string
  - Const pointer contains an address of const string (const pointer ㄴㅎ pointer const)
  - Const pointer is used to const a data or point to a data with constant nature
    - Example: const char* s = “abcd”; // Right

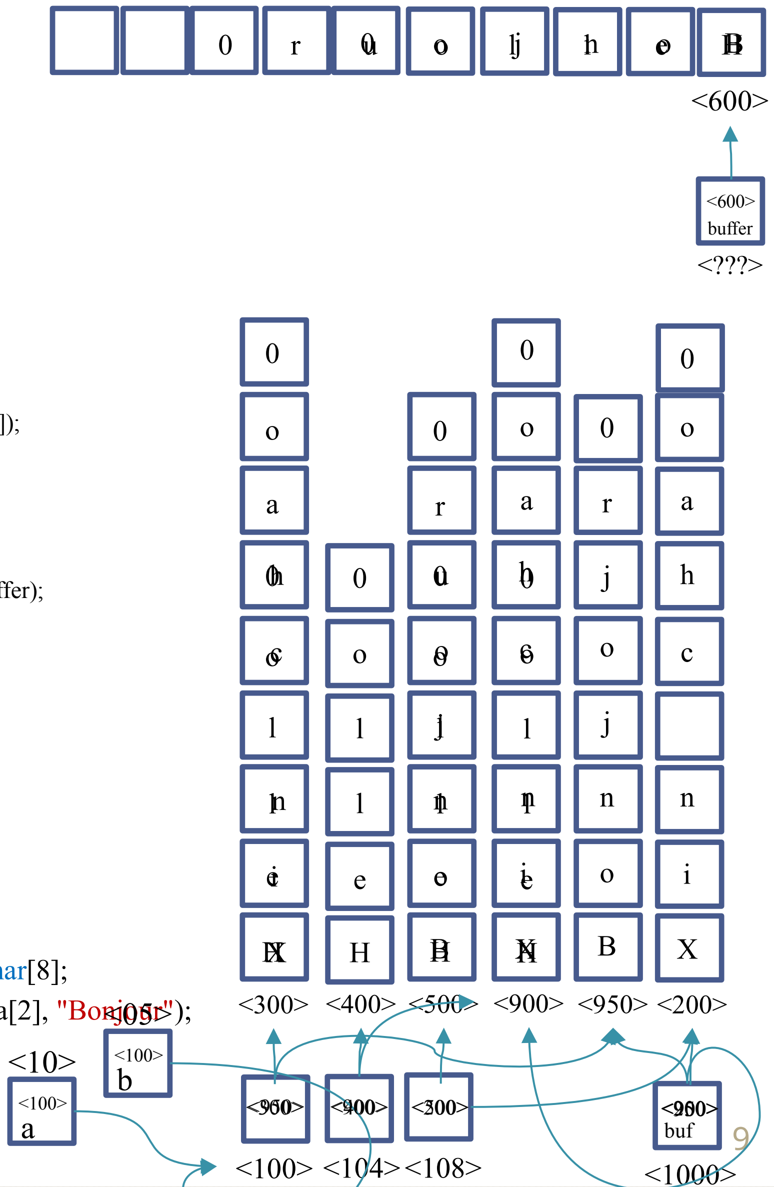
      char* s = “abcd”; // Wrong
  - Changing a const string with const pointer is illegal
    - Example: s[0] = ‘A’; // Wrong

SIMPLE OPERATION

- Sort an array of strings

```
void SortStringArray(char** b, int n){
    char buffer[10]; int len1, len2;
    for(int i = 0; i < n - 1; i++){
        for(int j = i + 1; j < n; j++){
            if(strcmp(b[i], b[j]) > 0){
                len1 = strlen(b[i]); len2 = strlen(b[j]);
                strcpy(buffer, b[j]);
                if(len2 < len1){
                    char* buf = new char[len1 + 1]; strcpy(buf, b[i]);
                    delete[] b[j]; b[j] = buf;
                }
                else strcpy(b[j], b[i]);
                if(len1 < len2){
                    char* buf = new char[len2 + 1]; strcpy(buf, buffer);
                    delete[] b[i]; b[i] = buf;
                }
                else strcpy(b[i], buffer);
            }
        }
    }
}
void main(){
    char** a = new char*[3];
    a[0] = new char[9]; a[1] = new char[6]; a[2] = new char[8];
    strcpy(a[0], "Xin chao");strcpy(a[1], "Hello");strcpy(a[2], "Bonjour");
    SortStringArray(a, 3);
    for(int i = 0; i < 3; i++) delete[] a[i];
    delete[] a;
}
```





SIMPLE OPERATION

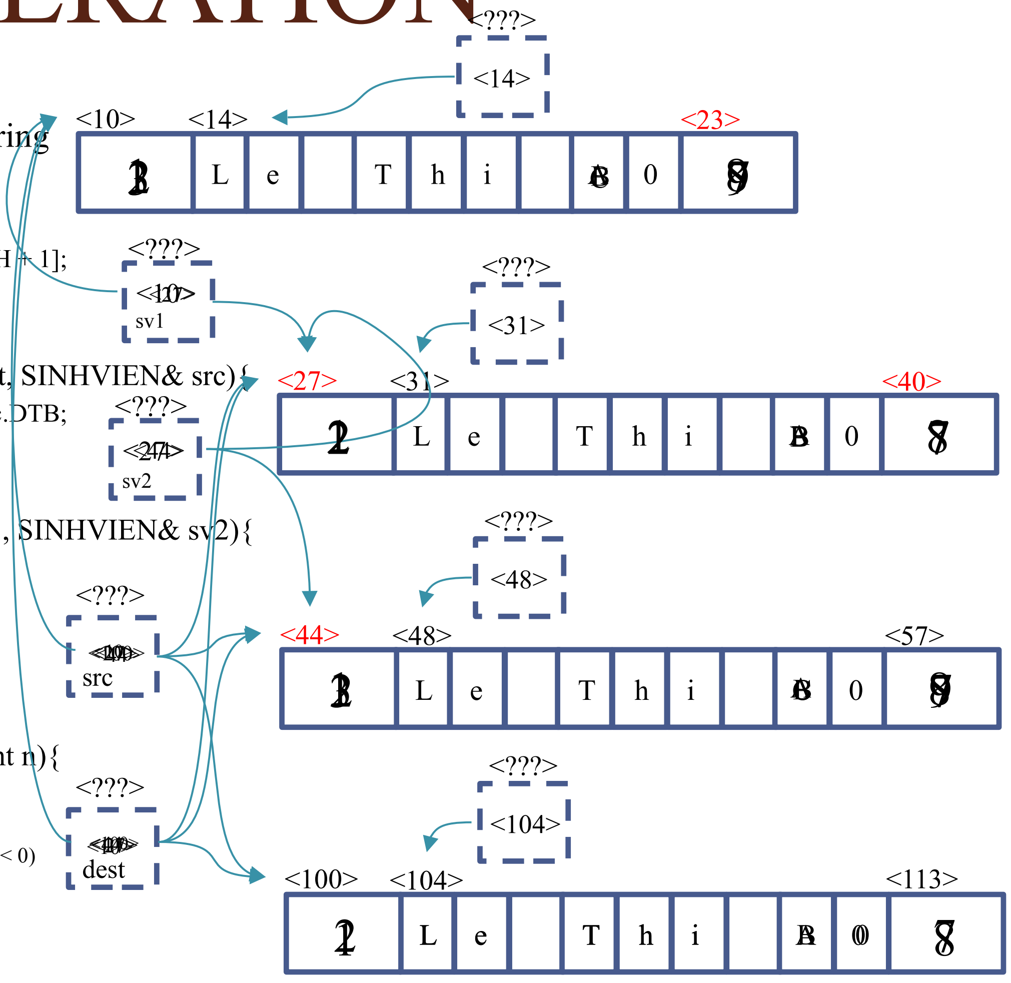
- Sort an array of strings
  - May use <string> of C++
  - Example:

```
void main(){
    string a[] = {“Xin chao”, “Hello”, “Bonjour”};
    SortStringArray(a, 3);
    for(int i = 0; i < 3; i++) cout << a[i] << endl;
}
void SortStringArray(string strArr[], int n){
    for(int i = 0; i < n - 1; i++){
        for(int j = i + 1; j < n; j++){
            if(strArr[i] > strArr[j]){
                string tmp = strArr[i];
                strArr[i] = strArr[j];
                strArr[j] = tmp;
            }
        }
    }
}
```

SIMPLE OPERATION

- Sort a structural array with a static string

```
#define MAX_LENGTH 8
typedef struct {
    int MaSo; char HoTen[MAX_LENGTH + 1];
    float DTB;
} SINHVIEN;
void copySinhVien(SINHVIEN& dest, SINHVIEN& src){
    dest.MaSo = src.MaSo; dest.DTB = src.DTB;
    strcpy(dest.HoTen, src.HoTen);
}
void swapSinhVien(SINHVIEN& sv1, SINHVIEN& sv2){
    SINHVIEN tmp;
    copySinhVien(tmp, sv1);
    copySinhVien(sv1, sv2);
    copySinhVien(sv2, tmp);
}
void sortSinhVien(SINHVIEN sv[], int n){
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(strcmp(sv[i].HoTen, sv[j].HoTen) < 0)
                swapSinhVien(sv[i], sv[j]);
}
void main(){
    SINHVIEN a[3] = {{1, "Le Thi A", 8},
  {2, "Le Thi B", 7},
                        {3, "Le Thi C", 9}};
    sortSinhVien(a, 3);
}
```



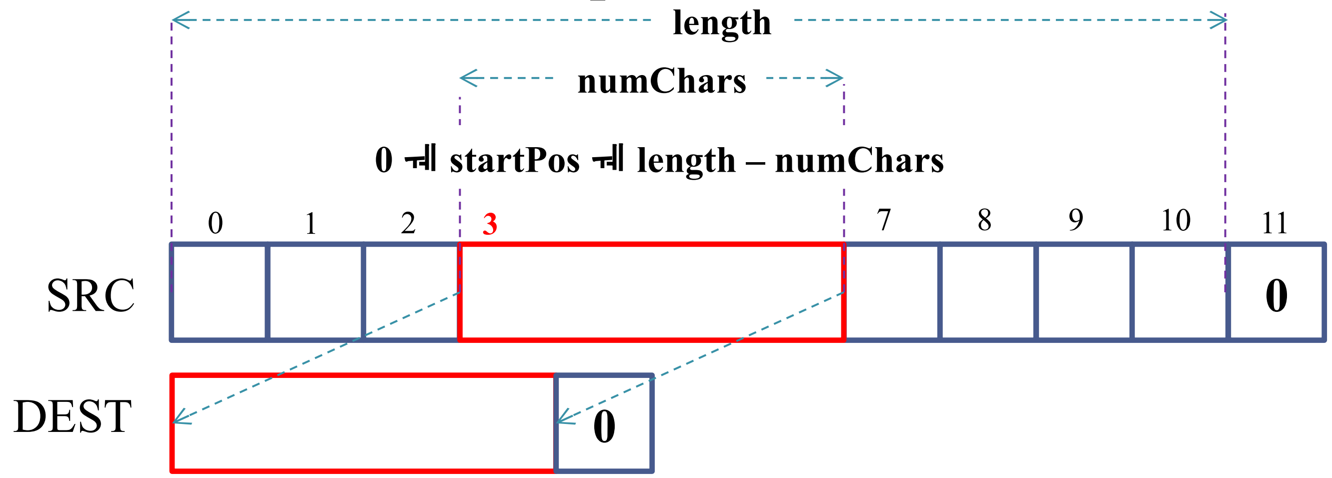
SIMPLE OPERATION

- Sort a structural array with a static string (use string)

C	C++
#define MAX_LENGTH 10	
typedef struct {	typedef struct {
int MaSo; char HoTen[MAX_LENGTH + 1];	int MaSo; string HoTen;
double DTB; }SVIEN;	double DTB; }SVIEN;
void ***copySinhVien***(SVIEN& d, SVIEN& s){	
d.MaSo = s.MaSo; d.DTB = s.DTB;	
strcpy(d.HoTen, s.HoTen);}	
void ***swapSinhVien***(SVIEN& sv1, SVIEN& sv2){	void ***swapSinhVien***(SVIEN& sv1, SVIEN& sv2){
SVIEN tmp; copySinhVien(tmp, sv1);	SVIEN tmp = sv1;
copySinhVien(sv1, sv2); copySinhVien(sv2, tmp);}	sv1 = sv2; sv2 = tmp;}
void ***sortSinhVien***(SVIEN sv[], int n){	void ***sortSinhVien***(SVIEN sv[], int n){
for(int i = 0; i < n - 1; i++)	for(int i = 0; i < n - 1; i++)
for(int j = i + 1; j < n; j++)	for(int j = i + 1; j < n; j++)
if(**strcmp**(sv[i].HoTen, sv[j].HoTen) < 0)	if(sv[i].HoTen < sv[j].HoTen)
swapSinhVien(sv[i], sv[j]); }	swapSinhVien(sv[i], sv[j]); }

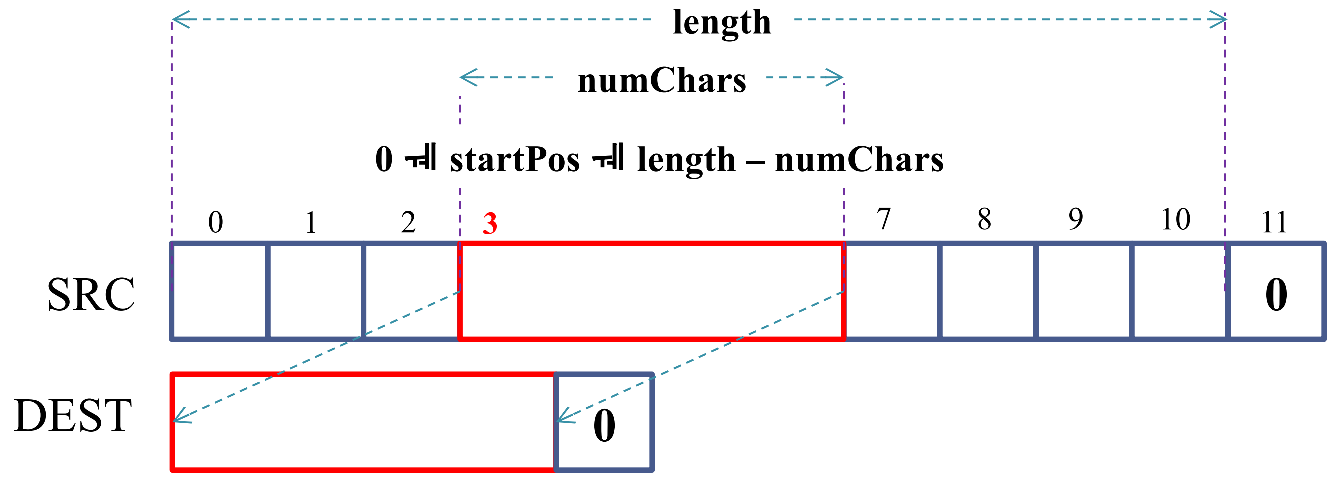
SIMPLE OPERATION

- String copy: there are many cases of extracting sub-string from main-string
  - Example:
    - Registration number XXXYZZZZZ (school-code, ordinal numbers)
    - Telephone number 098XXXXXXX (The first three numbers indicate operator)



SIMPLE OPERATION

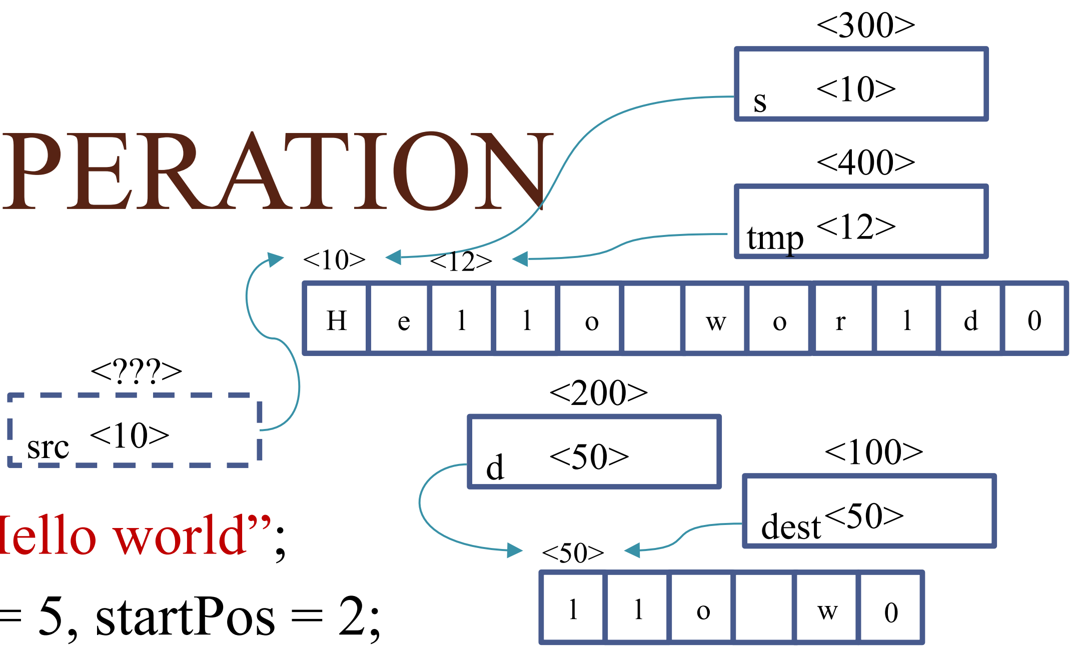
- String copy:
  - The parameters length, numChars and startPos must satisfy the condition
  - The length of main-string does not include ‘\0’
  - Length of string dest = numChars + 1



SIMPLE OPERATION

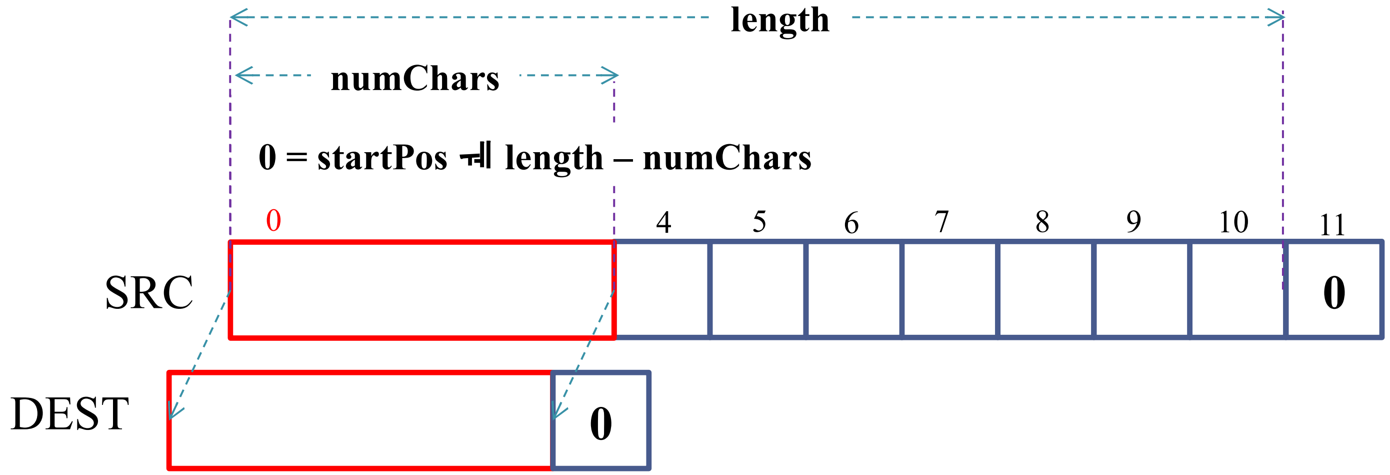
- String copy:

```
void main(){
    char src[] = “Hello world”;
    int numChars = 5, startPos = 2;
    char* dest = new char[numChars + 1];
    CopySubStr(dest, src, startPos, numChars);
    cout << dest << endl;
    delete[] dest;
}
void CopySubStr(char* d, char* s, int sp, int nc){
    strncpy(d, s + sp, nc);
    d[nc] = ‘\0’;
}
```



SIMPLE OPERATION

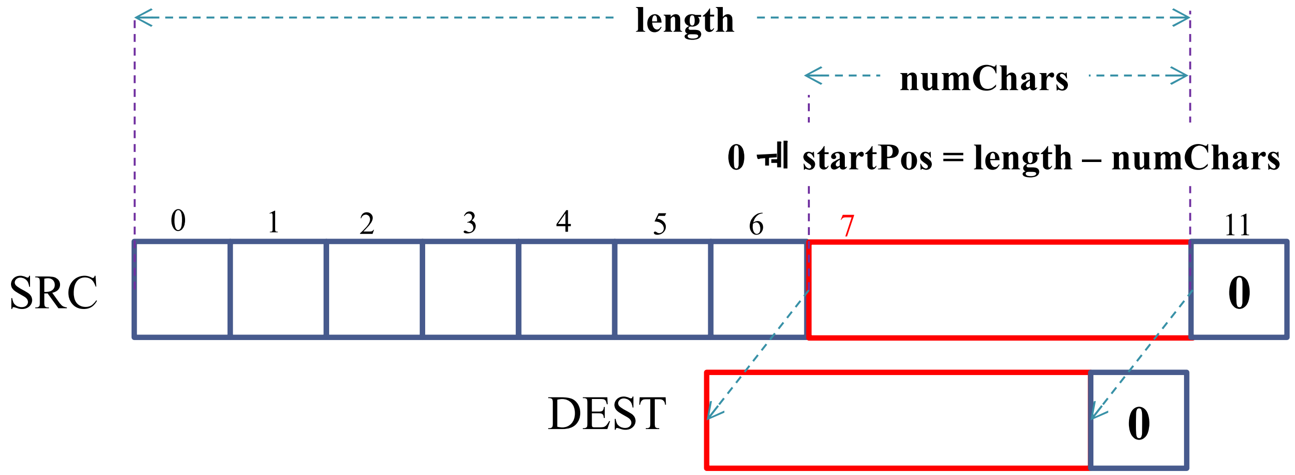
- String copy
  - Copy substring with startPos = 0



  - Example:
    - void GetLeftSubStr(char* d, char* s, int numChars){
      - int len = strlen(s);
      - if(numChars > len) numChars = len;
      - **CopySubStr**(d, s, 0, numChars);
  - }

SIMPLE OPERATION

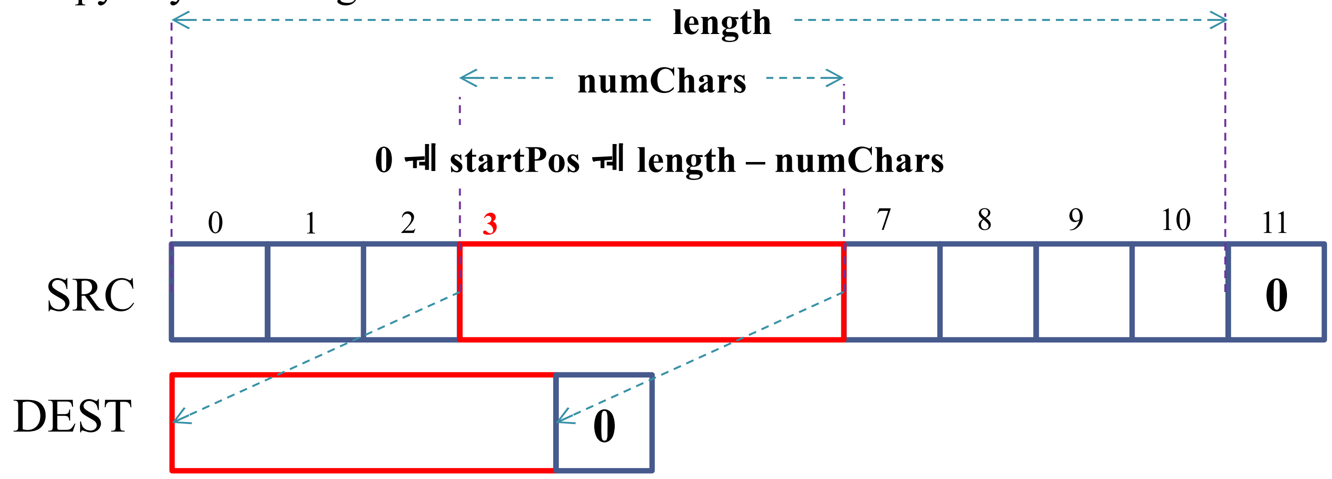
- String copy
  - Copy substring with startPos = length - numChars



  - Example:
    - void GetRightSubStr(char* d, char* s, int numChars){
      - int len = strlen(s);
      - if(numChars > len) numChars = len;
      - **CopySubStr**(d, s, len – numChars, numChars);
  - }

SIMPLE OPERATION

- String copy
  - Copy any substring

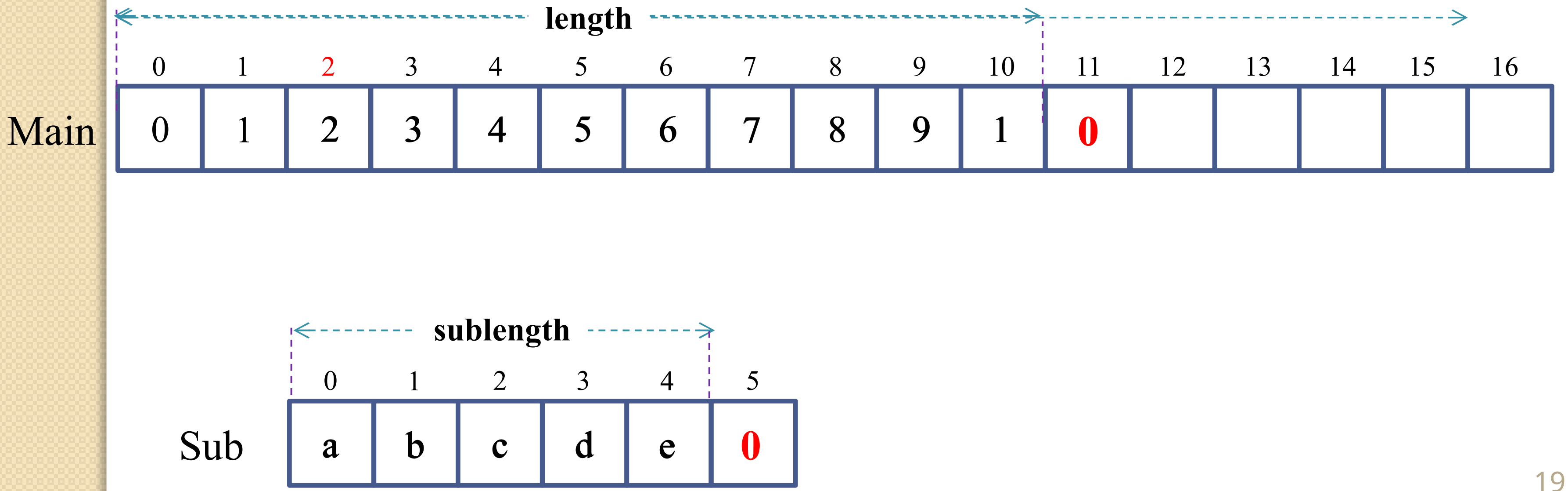


  - Example:

```
void GetSubStr(char* d, char* s, int startPos, int numChars){
    int len = strlen(s);
    if(startPos < len){
        if(startPos + numChars > len) numChars = len – startPos;
        CopySubStr(d, s, startPos, numChars);
    }
    else strcpy(d, “”);
}
```

SIMPLE OPERATION

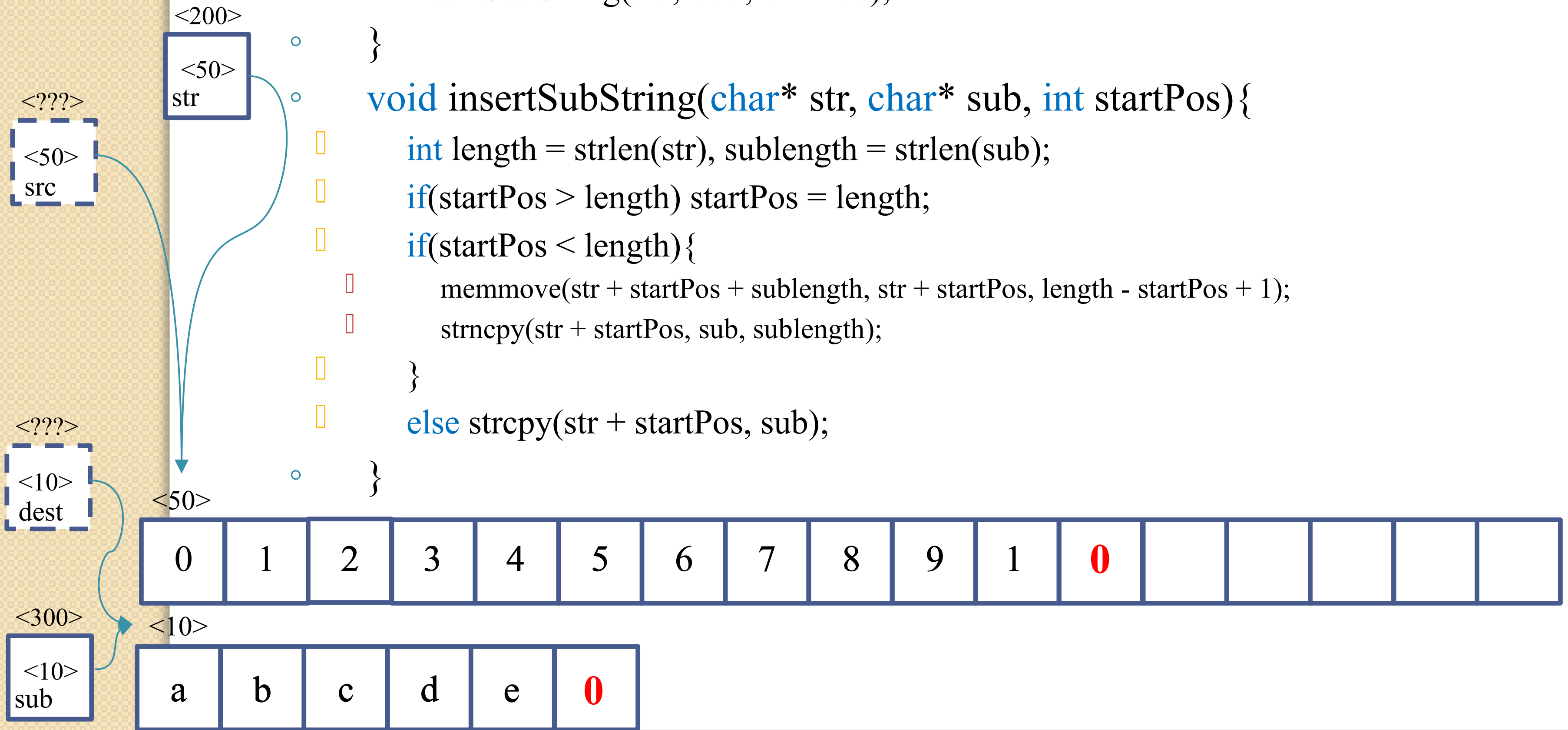
- Insert external string: insert a substring into main-string at another position
  - Example: insert **"abcde"** into **"01234567891"** at the position of character '2'. So, the result is "01abcde234567891"



SIMPLE OPERATION

- Insert external string :

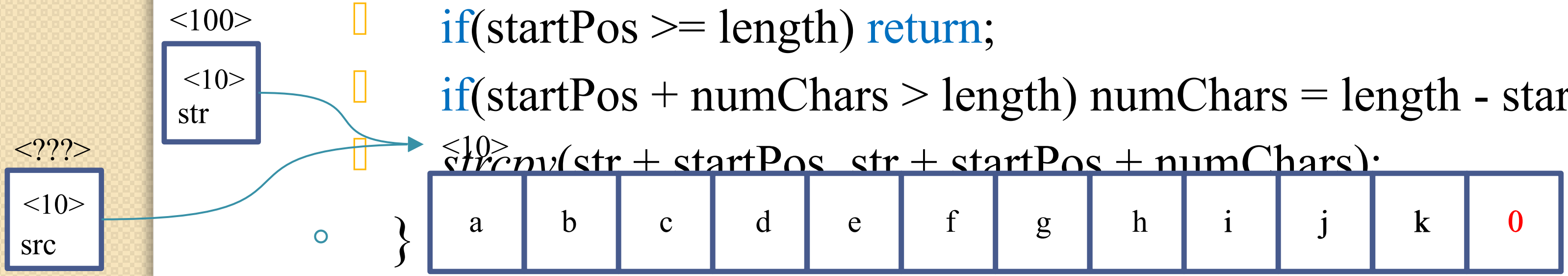
```
void main(){
    char src[] = "01234567891", dest[] = "abcde";
    int startPos = 2;
    insertSubString(src, dest, startPos);
}
void insertSubString(char* str, char* sub, int startPos){
    int length = strlen(str), sublength = strlen(sub);
    if(startPos > length) startPos = length;
    if(startPos < length){
        memmove(str + startPos + sublength, str + startPos, length - startPos + 1);
        strncpy(str + startPos, sub, sublength);
    }
    else strcpy(str + startPos, sub);
}
```



SIMPLE OPERATION

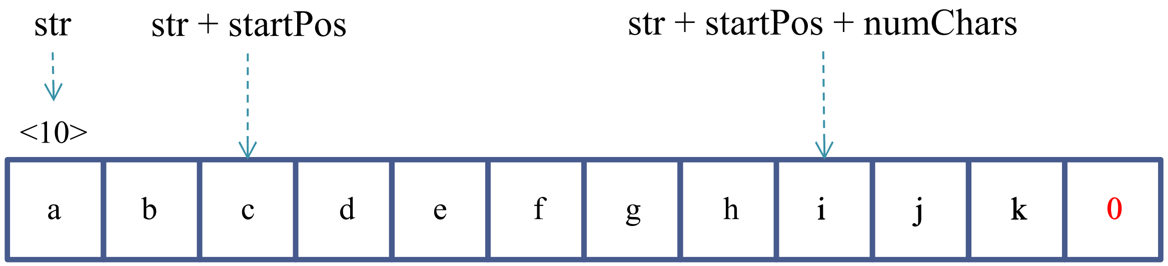
- Delete a substring in a main-string: delete a substring at another position in a main-string
  - Example: main-string "abcdefghijk" is deleted at index = 2 and the amount of character deleted is 6. So, the result is "abijk".

```
void main(){
    char src[] = "abcdefghijk";
    deleteSubString(src, 2, 6);
}
void deleteSubString(char* str, int startPos, int numChars){
    int length = strlen(str);
    if(startPos >= length) return;
    if(startPos + numChars > length) numChars = length - startPos;
    strcpy(str + startPos, str + startPos + numChars);
}
```

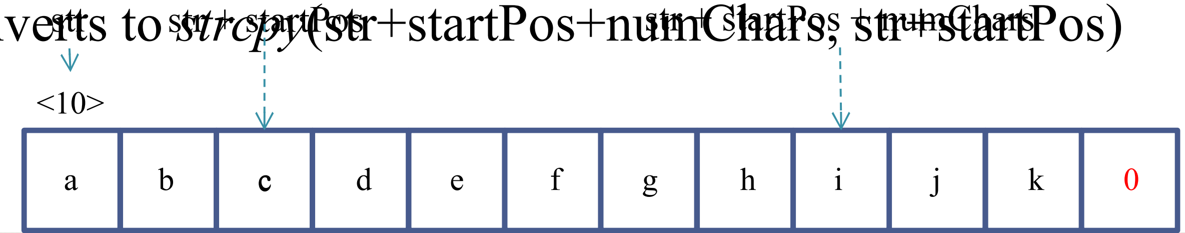


SIMPLE OPERATION

- Delete a substring in a main-string
  - Note with *strcpy*(char* dest, char* src)
    - This function is valid with a back-off operation (similar to demonstration of deleteSubString)



    - This function isn' valid with a forward operation
      - Example: *strcpy*(str+startPos, str+startPos+numChars) converts to *strcpy*(str+startPos+numChars, str+startPos)



TOKEN PROCESSING

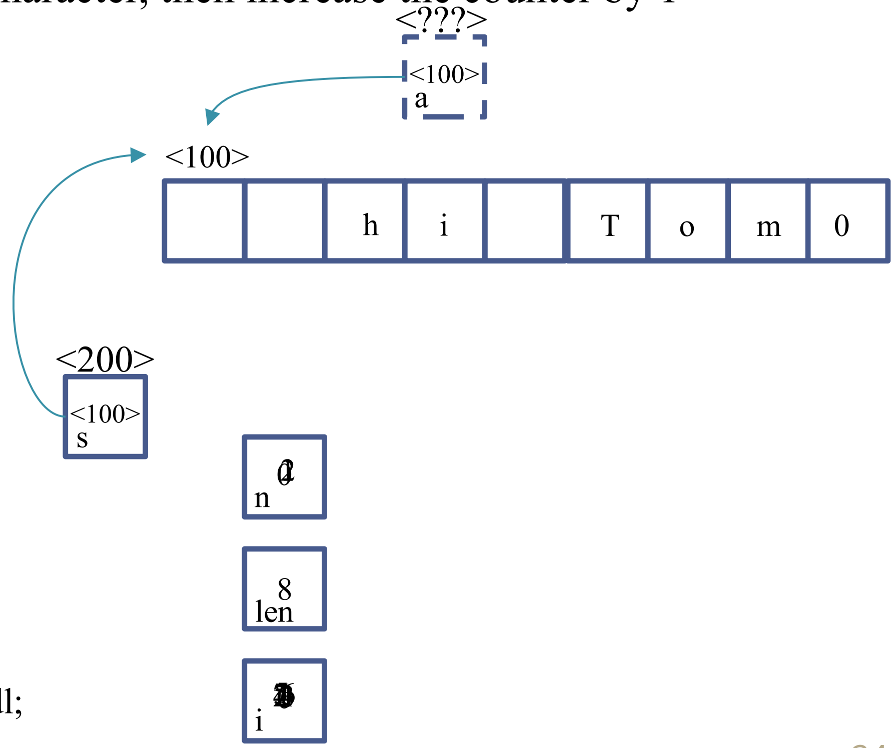
- What token is depends on separation-character.
- Example: “Ky thuat lap trinh, nhap mon lap trinh.”

Separation characters	**Token**
‘ ’ (space), ‘,’ (comma), ‘.’ (period)	There are **8 token**: “Ky”, “thuat”, “lap”, “trinh”, “nhap”, “mon”, “lap”, “trinh”
‘,’ (comma), ‘.’ (period)	There are **2 token**: “Ky thuat lap trinh” and “nhap mon lap trinh”
‘.’ (period)	There is **1 token**: “Ky thuat lap trinh, nhap mon lap trinh”

TOKEN PROCESSING

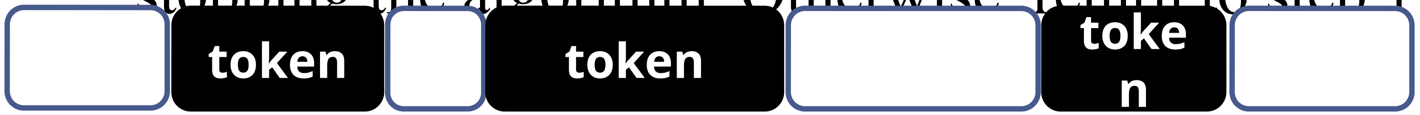
- Count a number of words in text file
  - 1st case: the first character is normal one
    - Increase counter var by 1, then finding other words
  - 2nd case: the first character is separation character
    - Scan until finding the first character, then increase the counter by 1
  - Algorithm:

```
int countWords(char* s){
    int n = 0, len = strlen(s), i = 0;
    if(s[0] != ‘ ’) { n++; i++; }
    for(; i < len - 1; i++)
        if(s[i] == ‘ ’)
            if(s[i + 1] != ‘ ’)
                n++;
    return n;
}
void main(){
    char a[] = “  hi Tom”;
    cout << countWords(a) << endl;
}
```



TOKEN PROCESSING

- Count the words in text file
  - Use some convenient function of C++ to implement this counting function
  - Idea:
    - Step 1: Ignore all the separation-characters at the start of a string to come the position of the first word. If it cannot find this position, stopping the algorithm. Otherwise go to step 2
    - Step 2: Ignore all the characters of the word just found at step 1 to come the position of the next separation-character. If it cannot find this position, stopping the algorithm. Otherwise, return to step 1



TOKEN PROCESSING

- Count the words in text file
  - Use some convenient function of C++ to implement this counting function
  - string.**find_first_not_of**(*sepString*, *startPos*): return the position of the first character ㅎ *sepString* from *startPos*
    - Example: “12345”.find_first_not_of(“345”, 0) ㅇ 0 because ‘1’ ㅎ “345”
  - string.**find_first_of**(*sepString*, *startPos*): return to the position of the first character ㅍ *sepString* from *startPos*
    - Example: “12345”.find_first_of(“345”, 0) -> 2 because ‘2’ ㅍ “345”

TOKEN PROCESSING

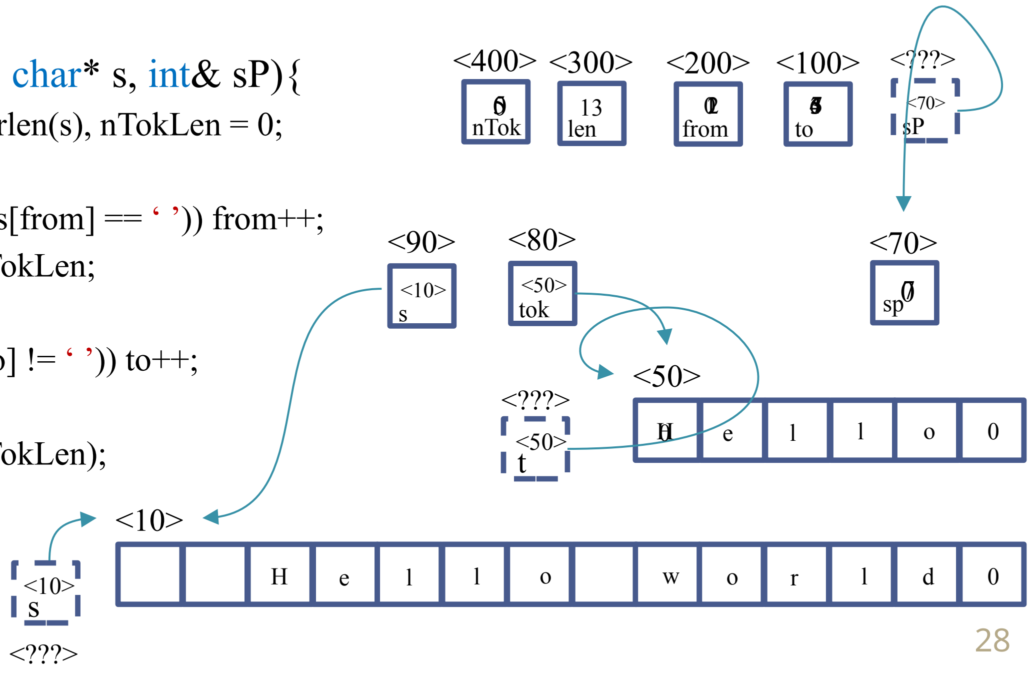
- Count the words in text file

```
void main(){
    string s = " hi Tom ";
    cout << countWords(s) << endl;
}
int countWords(string s){
    string sep = " ;:,.\n\t";
    int nWords = 0;
    string::size_type lastPos = s.find_first_not_of(sep, 0);
    string::size_type pos = s.find_first_of(sep, lastPos);
    while(string::npos != pos || string::npos != lastPos){
        nWords++;
        lastPos = s.find_first_not_of(sep, pos);
        pos = s.find_first_of(sep, lastPos);
    }
    return nWords;
}
```

TOKEN PROCESSING

- Take a token from a string
  - Idea: reuse the idea of countWords function
  - Return the length just extracted from a main-string, and record the position of newest separation-character for the next extraction
  - Example:

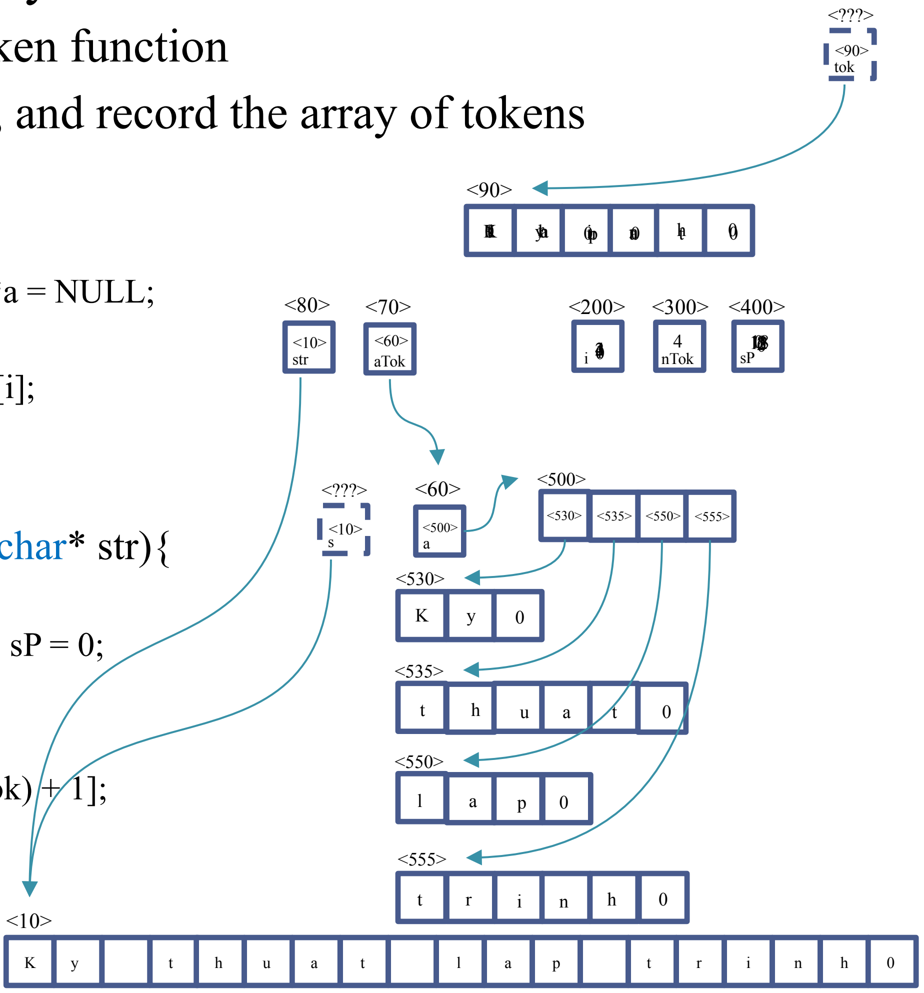
```
void main(){
    char s[] = "  Hello world", t[6]; int sp = 0; getToken(t, s, sp);
}
int getToken(char* tok, char* s, int& sP){
    int from = sP, to, len = strlen(s), nTokLen = 0;
    strcpy(tok, "");
    while((from < len) && (s[from] == ' ')) from++;
    if(from == len) return nTokLen;
    to = from + 1;
    while((to < len) && (s[to] != ' ')) to++;
    nTokLen = to - from;
    strncpy(tok, s + from, nTokLen);
    tok[nTokLen] = '\0';
    sP = to;
    return nTokLen;
}
```



TOKEN PROCESSING

- Separate a string into an array of tokens
  - Idea: reuse the idea of getToken function
  - Return the amount of tokens, and record the array of tokens
  - Example:

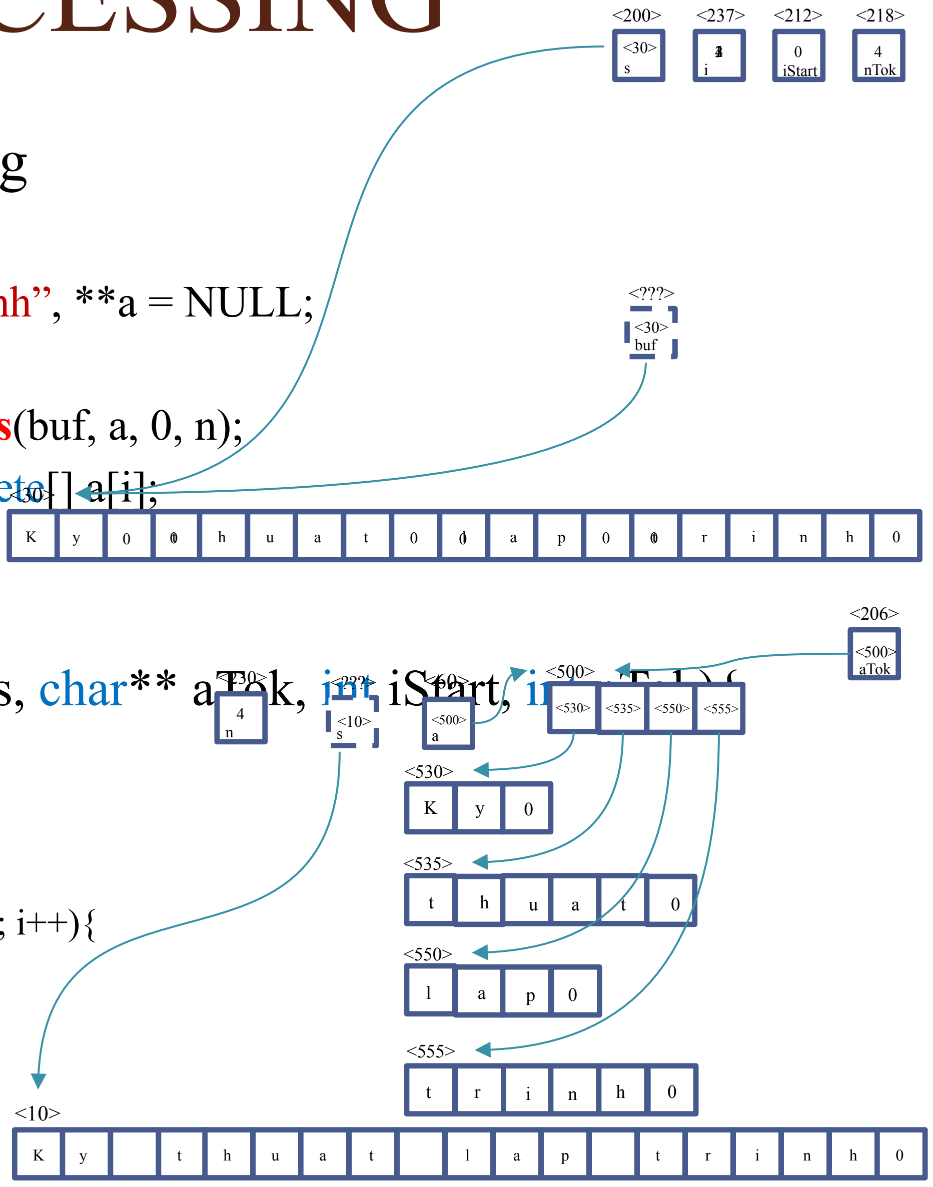
```
void main(){
    char s[] = "Ky thuat lap trinh", **a = NULL;
    cout << parseString(&a, s);
    for(int i = 0; i < 4; i++) delete[] a[i];
    delete[] a;
}
int parseString(char*** aTok, char* str){
    char tok[6];
    int i = 0, nTok = countWords(str), sP = 0;
    *aTok = new char*[nTok];
    while(getToken(tok, str, sP) > 0){
        (*aTok)[i] = new char[strlen(tok) + 1];
        strcpy((*aTok)[i], tok);
        i++;
    }
    return nTok;
}
```



TOKEN PROCESSING

- Merge tokens into a string

```
void main(){
    char s[] = “Ky thuat lap trinh”, **a = NULL;
    int n = parseString(&a, s);
    char buf[19]; mergeTokens(buf, a, 0, n);
    for(int i = 0; i < 4; i++) delete[] a[i];
    delete[] a;
}
void mergeTokens(char* s, char** aTok, int iStart, int nTok){
    if(nTok == 0) strcpy(s, “”);
    else{
        strcpy(s, aTok[iStart]);
        for(int i = iStart + 1; i < nTok; i++){
            strcat(s, “ ”);
            strcat(s, aTok[i]);
        }
    }
}
```



TOKEN PROCESSING

- Different applications
  - Normalize separations: " hello world " → "hello world"

```
void normalizeString(char* dest, char* src){
    char** aTok = NULL;
    int nTok = parseString(aTok, src);
    mergeToken(aTok, 0, nTok,dest);
}
```

  - Separate surname, name and middle-name: "Nguyen Thi Be Ba" → "Nguyen", "Thi Be", "Ba".

```
void parseName(string sHoTen, string& h, string& cl, string& t){
    vector<string> aTok;
    int n = parseString(aTok, sHoTen);
    h = aTok[0]; t = aTok[n - 1];
    mergeToken(cl, aTok, 1, n – 2);
}
```

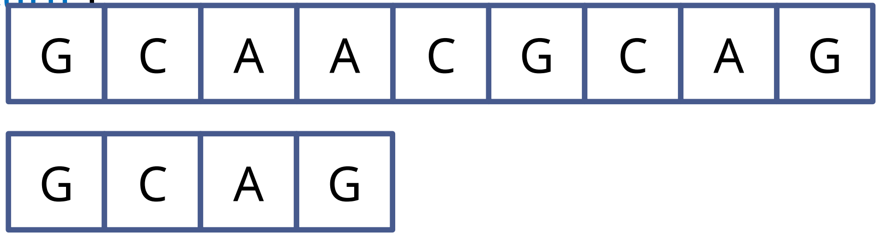
  - Separate day, month, year: "20/10/2100" → 20, 10, 2100

```
void parseDate(int& dd, int& mm, int& yyyy, char* strNgay){
    char** aTok = NULL;
    int n = parseString(aTok, strNgay);
    dd = atoi(aTok[0]); mm = atoi(aTok[1]); yyyy = atoi(aTok[2])
}
```

SEARCH IN STRING

- String matching algorithm (Brut-force)
  - Input: string needed to check (pat), main-string (s) and the position where starting to match (starPos)
  - Output: index if found and -1 if not

```
int isMatch(char* pat, char* s, int startPos){
    int pLen = strlen(pat), sLen = strlen(s), i, j;
    for(i = startPos; i <= (sLen - pLen); i++){
        for(j = 0; j < pLen && s[i + j] == pat[j]; j++);
        if(j == pLen) return i;
    }
    return -1;
}
```



SEARCH IN STRING

- String matching algorithm (Brut-force)
  - Can ‘break’ previous function into two sub simpler function
    - bool isMatch(char* pat, char* s, int startPos): check if **pat** is in **s** from **startPos** or not

```
bool isMatch(char* pat, char* s, int startPos){
    int pLen = strlen(pat), sLen = strlen(s), i;
    if(startPos + pLen > sLen) return false;
    for(i = 0; i < pLen; i++)
        if(pat[i] != s[startPos + i])
            return false;
    return true;
}
```

- int findSubString(char* pat, char* s, int startPos): find the index where **pat** appears

```
int FindSubString(char* pat, char* s, int startPos = 0){
    int pLen = strlen(pat), sLen = strlen(s), i, maxStartPos = sLen - pLen;
    if(startPos > maxStartPos) return -1;
    for(i = startPos; i <= maxStartPos; i++)
        if(isMatch(pat, s, i) == true)
            return i;
    return -1;
}
```

SEARCH IN STRING

- Substring checking algorithm
  - Reuse “isMatch” and “findSubString”

```
bool isSubString(char* pat, char* s){
    if(findSubString(pat, s, 0) >= 0) return true;
    return false;
}
```

- Counting a number of appearance of substring
  - Reuse the ideas of “findSubString” and “isMatch”

```
int CountMatches(char* pat, char* s){
    int pLen = strlen(pat), sLen = strlen(s);
    int maxStartPos = sLen – pLen, count = 0;
    for(i = 0; i <= maxStartPos; i++)
        if(isMatch(pat, s, i) == true) count++;
    return count;
}
```

  - Ex 1: pat = “abc” and s = “**abcdabce**” => count = 2
  - Ex 2: pat = “aa” and s = “**aaaa**” => count = 3

SEARCH IN STRING

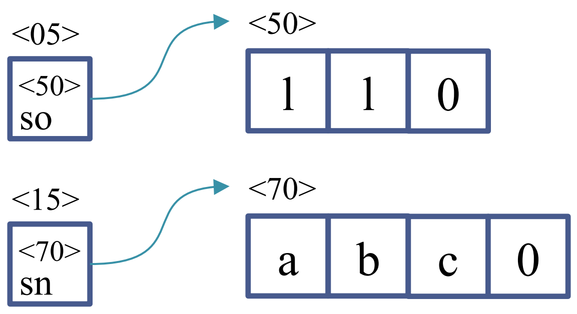
- Counting a number of appearance of disjoint substring
  - Ex 1: pat = “abc”, s = “abcdabce” → count = 2
  - Ex 2: pat = “aa”, s = “aaaa” → count = 2

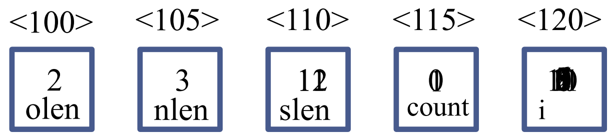
```
int CountDisjointMatches(char* pat, char* s){
    int pLen = strlen(pat), sLen = strlen(s);
    int maxStartPos = sLen – pLen, count = 0;
    for(i = 0; i <= maxStartPos; i++)
        if(isMatch(pat, s, i) == true)
            count++;
            i += (pLen - 1);
    return count;
}
```

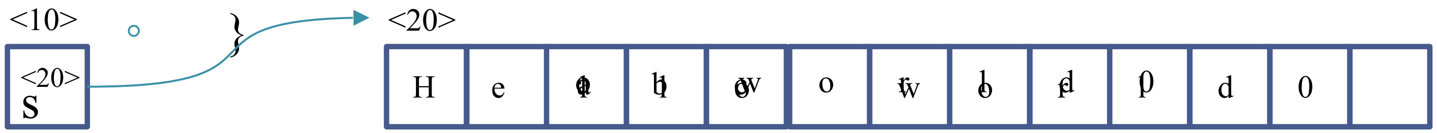
SEARCH IN STRING

- Replace a substring in a main-string
  - Ex: s = "Hello world", so = "ll", sn = "abc" → s = "Heabco world"
  - Input: original string *s*, string to be replaced *so* and string to replace *sn*
  - Output: a number of replacement, and original string s will be changed
  - int **replaceSubString**(char* so, char* sn, char* s){

```
    int olen = strlen(so), nlen = strlen(sn), slen = strlen(s), count = 0, i = 0;
    while(i <= (slen - olen)){
        if(isMatch(so, s, i)){
            deleteSubString(s, i, olen);
            insertSubString(s, sn, i);
            slen = slen + (nlen - olen);
            i += nlen;
            count++;
        }
        else i++;
    }
    return count;
}
```







STRING MANIPULATION

- String normalization
  - Need to normalize each token in string
    - Capitalize the first character of the token
    - Uncapitalize the remaining characters of the token

1	int isCapitalLet(char c){	int isLowercaseLet(char c){
2	if(c >= 'A' && c <= 'Z') return 1;	if(c >= 'a' && c <= 'z') return 1;
3	return 0;	return 0;
4	}	}
5	void normalizeWord(char* w){	
6	if(**isLowercaseLet**(w[0])) w[0]-=32;	
7	for(int i = 1; i < strlen(w); i++)	
8	if(**isCapitalLet**(w[i])) w[i]+=32;	
9	}	

STRING MANIPULATION

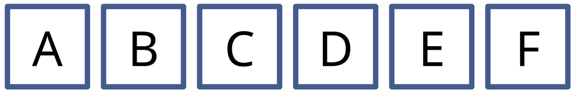
- String normalization
  - Some steps to normalize
    - Parse a string into a list of tokens
    - Normalize each token in the list
    - Merge all tokens into a string

```
void normalizeString(char* des, char* src){
  char** aTok = NULL;
  int nTok = parseString(aTok, src);
  for(int i = 0; i < nTok; i++)
    normalizeWord(aTok[i]);
  mergeTokens(des, aTok, 0, nTok);
}
```

STRING MANIPULATION

- Reverse string
  - Reverse the order of the characters of a string
  - Ex: “Hello world” → “dlrow olleH”

```
void reverseString(char* s){
  for(int i = 0; i < strlen(s)/2; i++){
    char t = s[i];
    s[i] = s[strlen(s) – 1 – i];
    s[strlen(s) – 1 – i] = t;
  }
}
```



CHARACTER/EXTENDED STRING

- One-byte character: 1 byte → 1 character
  - Example: 97 → ‘a’ ($97_{10} = 01100001_2$)
- Multi-byte: 1 character → multi bytes
  - Example: codepage VNI

Characters use 1 byte			Characters use 2 byte		
Character	**Dec value**	**Hex value**	**Character**	**Dec value**	**Hex value**
‘a’	94	0x61	‘á’	63841	0xF961
‘B’	66	0x42	‘ậ’	58465	0xE461
‘0’	48	0x30	‘ỹ’	62841	0xF579
‘ì’	236	0xEC	‘ỏ’	64367	0xFB6F
‘@’	64	0x40	‘ê’	57957	0xE265

  - Ex: a string has characters with different bytes

K	ỹ		t	h	u	ậ	t		l	ậ	p		t	r	ì	n	h	0

CHARACTER/EXTENDED STRING

- Extended character: all characters of a string must be the same bytes
  - Example: codepage built-in Unicode (2-byte characters)

Character	Dec value	Hex value	Character	Dec value	Hex value
‘a’	94	0x61	‘á’	225	0x00E1
‘B’	66	0x42	‘ậ’	7853	0x1EAD
‘0’	48	0x30	‘ỹ’	7929	0x1EF9
‘9’	57	0x39	‘ì’	236	0x00EC
‘@’	64	0x40	‘ê’	7887	0x1ECF

  - Ex: string with 2-byte characters (use wchar_t)
    - wc

H	e	l	l	o	0

CHARACTER/EXTENDED STRING

- Codepage Unicode
  - A numbering system of all characters of all nations
  - Contain 1114112 different characters
  - 96000 characters are used
  - There are many methods of presenting a character with Unicode
    - Use UTF-32: one character with 4 bytes
    - Use UTF-16: one character with 2 or 4 bytes
    - Use UTF-8: one character with 1 ○ 4 bytes
  - Some text files with strings of UTF-8 characters need special processing functions

CHARACTER/EXTENDED STRING

- Process a string with extended characters
  - A string of multi-byte characters: build functions to recognize the boundary of characters of string

K	ỹ		t	h	u	ậ	t		l	ậ	p		t	r	ì	n	h	0

  - Extended string: characters with the same bytes

H	e	l	l	o	0

    - C language supports 16-bit string in <string.h>
      - Replace char with wchar_t
      - Replace **str**len(8-bit string) with **wcs**len(16-bit string)
      - Replace printf with **w**printf
      - …
    - C++ language supports 16-bit string in <string>
      - Replace string with **w**string
      - Replace cout with **w**cout
      - …