Module G-Strings

Slots 22 & 23: Theory and Demo.
Slot 24: Exercise

Objectives

- String is a common-used data type → The way is used to store a string of characters in C.
- How to declare/initialize a string in C?
- How to access a character in a string?
- What are operations on strings
  - Input/output (stdio.h)
  - Some common used functions in the library **string.h**
- **How to manage an array of strings?**

Content

- Null-String/C-String
- To Declare/Initialize a string
- Gap: A safe method for string content.
- Data stored in a string
- Output a String
- Input a string
- May Operators Applied to String?
- Other String Functions
- Array of strings

1- Null-String/ C-String

- A string is a group of characters → It is similar to an array of characters.
- A NULL byte (value of 0 – escape sequence ‘\0’) is inserted to the end of a string. → It is called NULL-string or C-string.
- A string is similar to an array of characters. The difference between them is at the end of a string, a NULL byte is inserted to locate the last meaningful element in a string.

➔ If a string with the length **n** is needed, declare it with the length **n+1**.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
M	y		n	a	m	e		i	s		A	r	n	o	l	d	\0

2- Declare/ Initialize a String

- **Static strings**: stored in data segment or stack segment → Compiler can determine the location for storing strings.

```
char  s1[21]; /* for a string of 20 characters*/
```

  Initialize a string: NULL byte is automatically inserted.

```
char name[31] = “I am a student”;
char name2[31] = {‘H’, ‘e ‘, ‘l’, ‘l’, ‘o’, ‘\0’ };
```

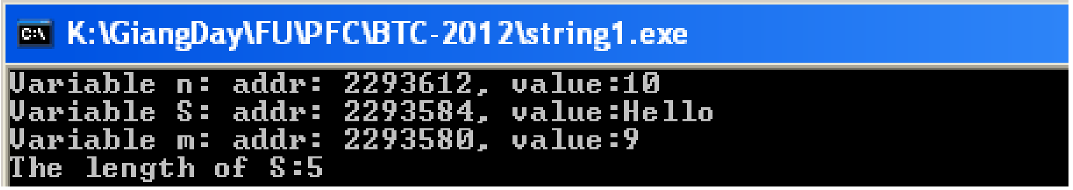
- Dynamic strings: Stored in the heap

```
char* S;
S = (char*) malloc( lengthOfString+1);
S = (char*) calloc( lengthOfString+1, sizeof(char));
```

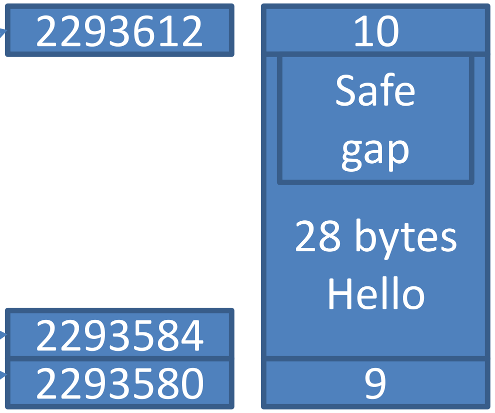
3- Gap:
A safe method for string content.

- Some compilers use a gap between variables to make a safety for strings.

```c
/* thu nghiem chuoi */
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{    int n=10;
     char S[11]="Hello";
     int m=9;
     printf("Variable n: addr: %u, value:%d\n", &n, n);
     printf("Variable S: addr: %u\, value:%s\n", S, S);
     printf("Variable m: addr: %u\, value:%d\n", &m, m);
     printf("The length of S:%d\n", strlen(S));
     getch();
     return 0;
}
```


K:\GiangDay\FU\PFC\BTC-2012\string1.exe

```
Variable n: addr: 2293612, value:10
Variable S: addr: 2293584, value:Hello
Variable m: addr: 2293580, value:9
The length of S:5
```



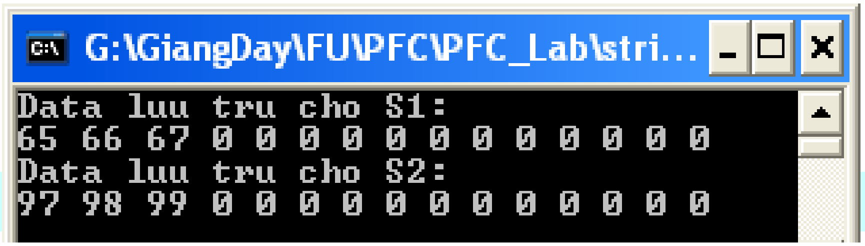
If a so-long string is accepted, this string can overflow into the memory of the variable n

4- Data Stored in a strings

- Each character in a string is stored as it’s ASCII code.

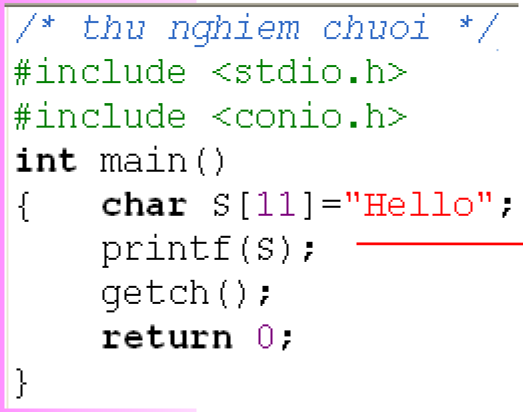
```c
/* string01.c-xem noi dung luu tru 1 chuoi */
#include <stdio.h>
#include <conio.h>
int main(){
   char S1[15]="ABC";
   char S2[15]= {'a','b','c','\0'};
   int i ;
   printf("Data luu tru cho S1:\n");
   for (i=0;i<15;i++) printf("%d ", S1[i]);
   printf("\n");
   printf("Data luu tru cho S2:\n");
   for (i=0;i<15;i++) printf("%d ", S2[i]);
   getch();
   return 0;
}
```

S1[i]: The character at the position i in the string S1

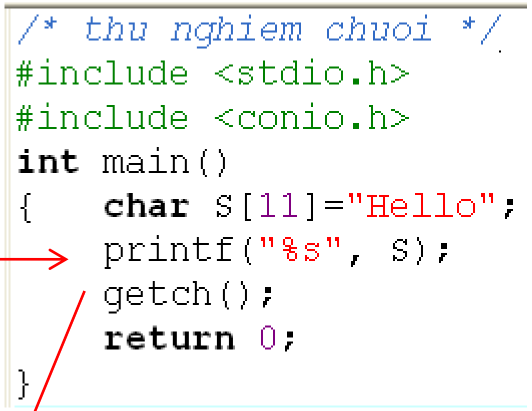

G:\GiangDay\FU\PFC\PFC_Lab\stri...

```
Data luu tru cho S1:
65 66 67 0 0 0 0 0 0 0 0 0 0 0 0
Data luu tru cho S2:
97 98 99 0 0 0 0 0 0 0 0 0 0 0 0
```

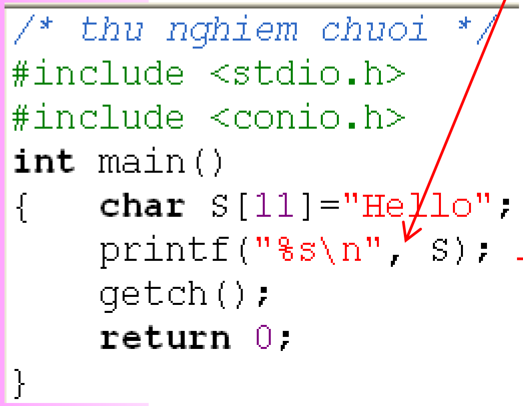
5- Output Strings – Test yourself



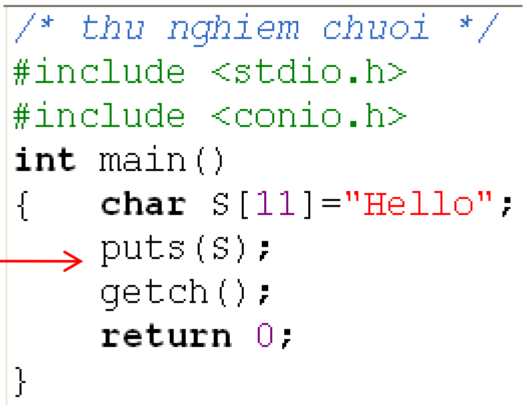
```
/* thu nghiem chuoi */
#include <stdio.h>
#include <conio.h>
int main()
{    char S[11]="Hello";
     printf(S);
     getch();
     return 0;
}
```



```
/* thu nghiem chuoi */
#include <stdio.h>
#include <conio.h>
int main()
{    char S[11]="Hello";
     printf("%s", S);
     getch();
     return 0;
}
```



```
/* thu nghiem chuoi */
#include <stdio.h>
#include <conio.h>
int main()
{    char S[11]="Hello";
     printf("%s\n", S);
     getch();
     return 0;
}
```



```
/* thu nghiem chuoi */
#include <stdio.h>
#include <conio.h>
int main()
{    char S[11]="Hello";
     puts(S);
     getch();
     return 0;
}
```

Observe the prompt symbol on the result screen .

6- Input Strings

- Library: stdio.h
- Function *scanf()* with type conversion %s
- Function *gets(string)*
- Each function has it’s own advantages and weaknesses.

Input Strings: scanf(…)

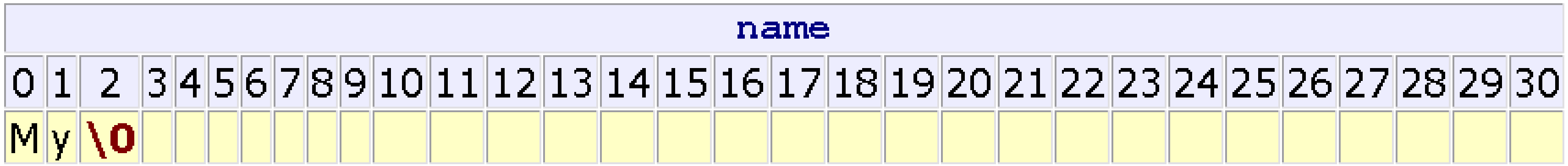
The **%s** conversion specifier

- reads all characters until the first whitespace character,
- stores the characters read in memory locations starting with the address passed to **scanf**,
- Automatically stores the null byte in the memory byte following the last character accepted and
- leaves the delimiting **whitespace** plus any subsequent characters in the input buffer → ignores any leading whitespace characters (default).
- Option specifiers are used to change default characteristics of the function **scanf** on strings.

Input Strings: scanf(…)

char name[31];
scanf("%s", name);
Enter: **My name is Arnold**

Default

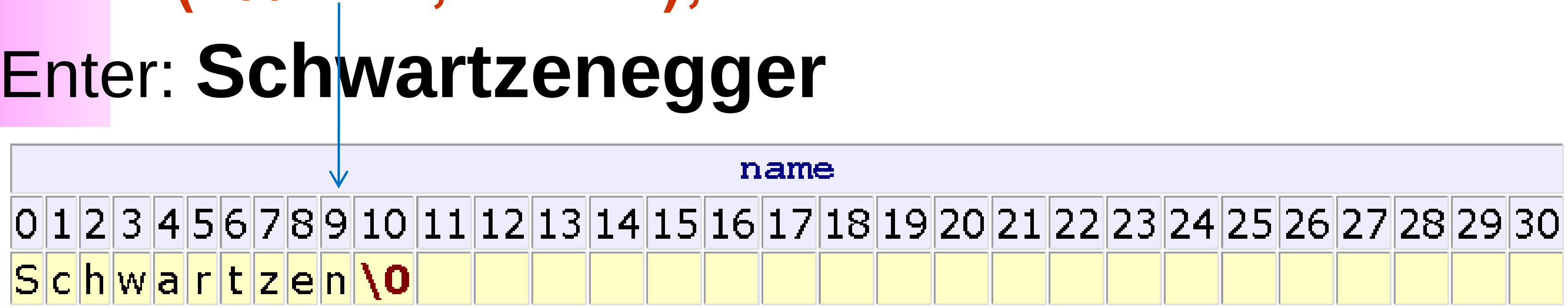


name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y	\0																												

char name[31];

scanf(“%10s", name);

Enter: **Schwartzenegger**



name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
S	c	h	w	a	r	t	z	e	n	\0																				

Input Strings: scanf(…)

How to accept blanks in a input string?

➔**%[^\n]** conversion specifier

- reads all characters <u>until the newline</u> (**'\n'**),
- stores the characters read in memory locations starting with the address passed to **scanf**,
- stores the null byte in the byte following that where **scanf** stored the last character and
- leaves the delimiting character (here, **'\n'**) in the input buffer.

Input Strings: scanf(…)

How to accept blanks in a input string?

➔**%[^\n]** conversion specifier.

```
scanf("%[^\n]", name );
```

```
My name is Arnold
```

stores

name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s		A	r	n	o	l	d	\0													

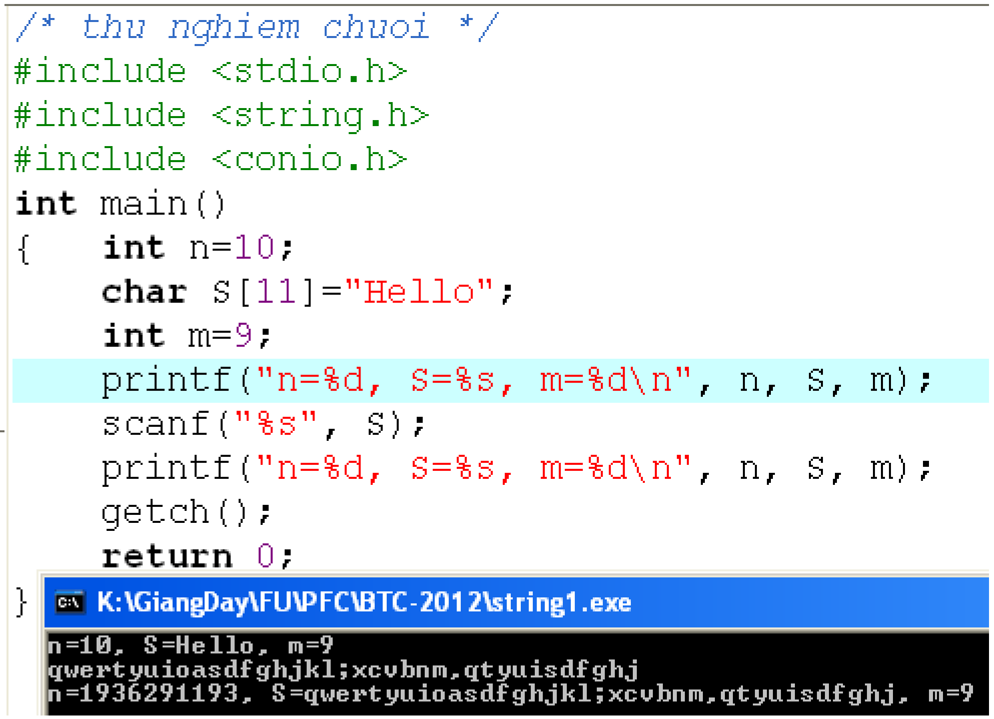
```
scanf("%10[^\n]", name );
```

```
My name is Arnold
```

stores

name																														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
M	y		n	a	m	e		i	s	\0																				

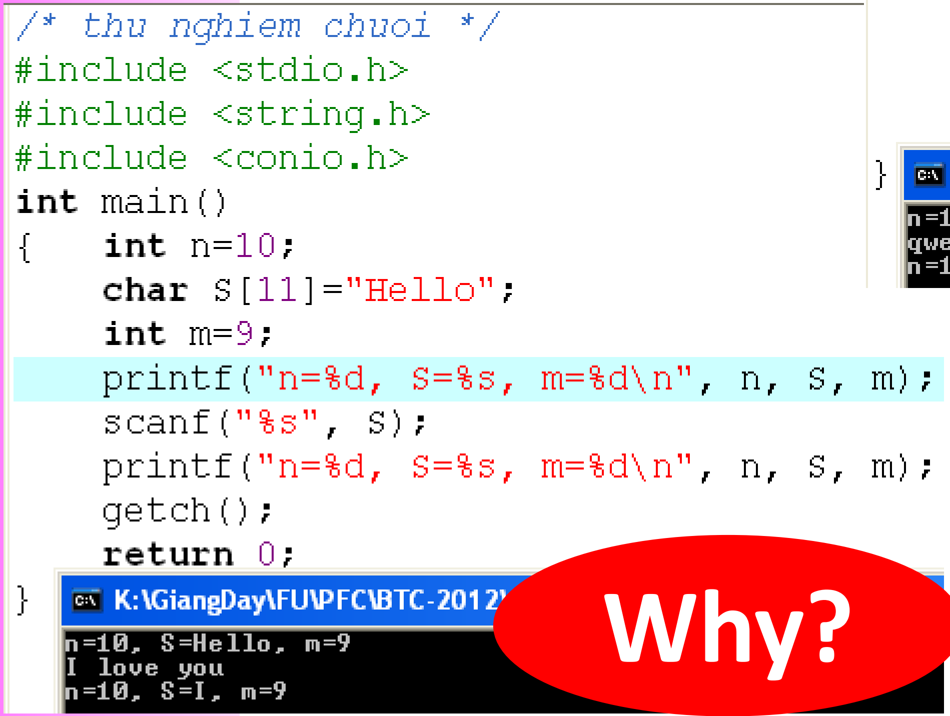
Input Strings: scanf(…) - Test



```
/* thu nghiem chuoi */
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{    int n=10;
     char S[11]="Hello";
     int m=9;
     printf("n=%d, S=%s, m=%d\n", n, S, m);
     scanf("%s", S);
     printf("n=%d, S=%s, m=%d\n", n, S, m);
     getch();
     return 0;
}
```

K:\GiangDay\FU\PFC\BTC-2012\string1.exe

```
n=10, S=Hello, m=9
qwertyuioasdfghjkl;xcvbnm,qtyuisdfghj
n=1936291193, S=qwertyuioasdfghjkl;xcvbnm,qtyuisdfghj, m=9
```



```
/* thu nghiem chuoi */
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{    int n=10;
     char S[11]="Hello";
     int m=9;
     printf("n=%d, S=%s, m=%d\n", n, S, m);
     scanf("%s", S);
     printf("n=%d, S=%s, m=%d\n", n, S, m);
     getch();
     return 0;
}
```

K:\GiangDay\FU\PFC\BTC-2012\

```
n=10, S=Hello, m=9
I love you
n=10, S=I, m=9
```

Why?

Why?

Replace:
scanf(“%s”, S) → scanf(“%10[^\n]”, S)

Input Strings: scanf(…)

Some character specifiers used in the function scanf(): Set of character are or not accepted.

Specifier	Description
%[abcd]	Searches the input field for any of the characters a, b, c, and d
%[^abcd]	Searches the input field for any characters ***except*** a, b, c, and d
%[0-9]	To catch all decimal digits
%[A-Z]	Catches all uppercase letters
%[0-9A-Za-z]	Catches all decimal digits and all letters
%[A-FT-Z]	Catches all uppercase letters from A to F and from T to Z

Input Strings: gets(…)

gets is a standard library function (stdio.h) that

– accepts an empty string

– uses the **'\n'** as the delimiter

– throws away the delimiter after accepting the string

– Automatically appends the null byte to the end of the set stored

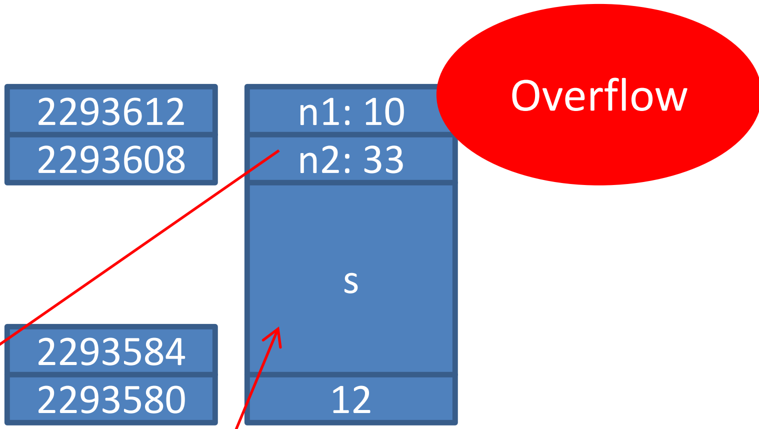
The prototype for **gets** is

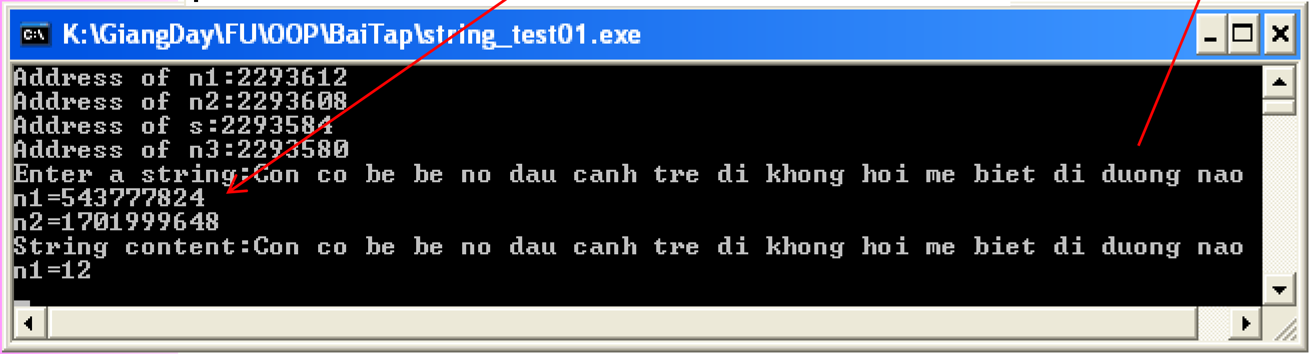
char* gets(char [ ]);

*(**gets** is dangerous. It can fill beyond the memory that allocated for the string*)

Input Strings: gets(…)

```c
#include <stdio.h>
int main()
{   int n1=10;
    int n2= 33;
    char s[11];
    int n3=12;
    printf("Address of n1:%u\n", &n1);
    printf("Address of n2:%u\n", &n2);
    printf("Address of s:%u\n", s);
    printf("Address of n3:%u\n", &n3);
    printf("Enter a string:");
    gets(s);
    printf("n1=%d\n", n1);
    printf("n2=%d\n", n2);
    printf("String content:%s\n", s);
    printf("n1=%d\n", n3);
    getchar();
    return 0;
}
```




K:\GiangDay\FU\OOP\BaiTap\string_test01.exe

```
Address of n1:2293612
Address of n2:2293608
Address of s:2293584
Address of n3:2293580
Enter a string:Con co be be no dau canh tre di khong hoi me biet di duong nao
n1=543777824
n2=1701999648
String content:Con co be be no dau canh tre di khong hoi me biet di duong nao
n1=12
```

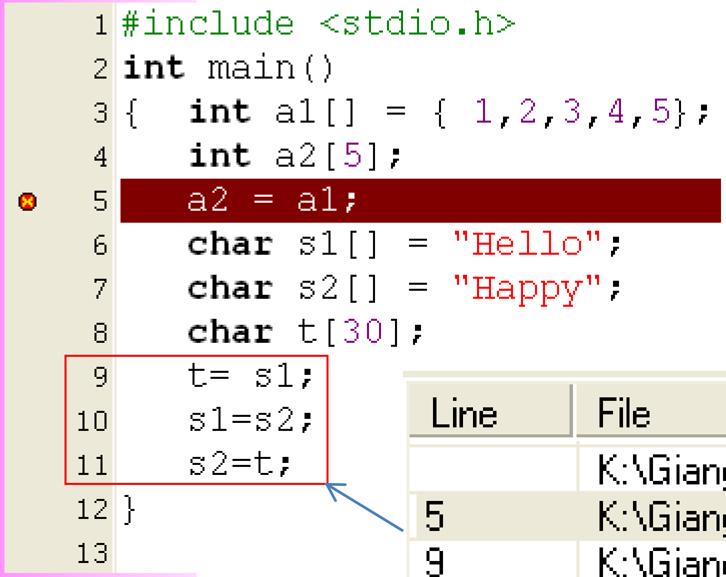
Input Strings:
Do yourself a function for input s string

```
/* getstr accepts a newline terminated string s of up
*   to max characters, appends a null byte and throws
*   away the terminating character
*/
void getstr(char s[], int max) {
    int i, c;

    i = 0;
    while((c = getchar()) != '\n' && c != EOF)
        if (i < max)
            s[i++] = (char) c;
    s[i] = '\0';
}
```

7- May Operators Applied to String?

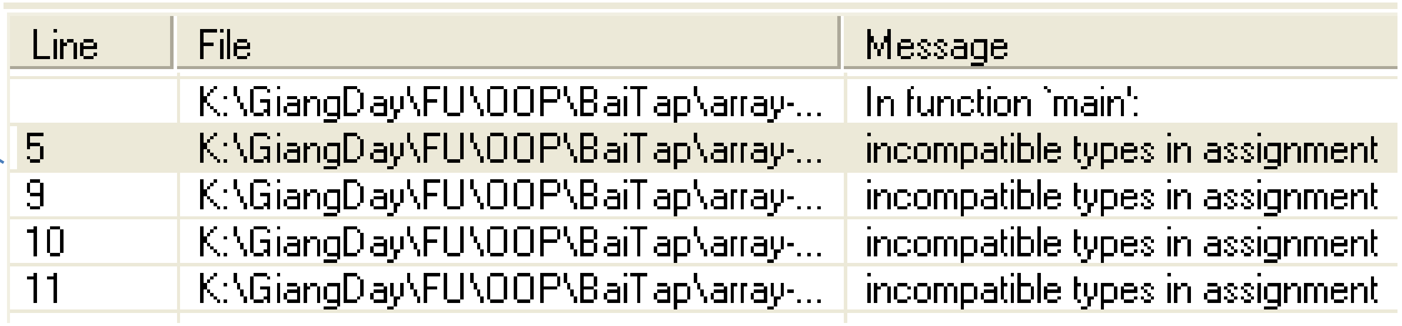
- C operators act on basic data type only.

➔ They can not be applied to static arrays and static strings. $\cancel{=}$ $\cancel{+}$ $\cancel{>}$ $\cancel{<}$



```
1  #include <stdio.h>
2  int main()
3  {   int a1[] = { 1,2,3,4,5};
4      int a2[5];
5      a2 = a1;
6      char s1[] = "Hello";
7      char s2[] = "Happy";
8      char t[30];
9      t= s1;
10     s1=s2;
11     s2=t;
12 }
13
```

We need functions for processing arrays and string

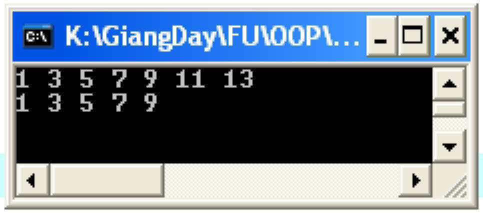


Line	File	Message
	K:\GiangDay\FU\OOP\BaiTap\array-...	In function `main':
5	K:\GiangDay\FU\OOP\BaiTap\array-...	incompatible types in assignment
9	K:\GiangDay\FU\OOP\BaiTap\array-...	incompatible types in assignment
10	K:\GiangDay\FU\OOP\BaiTap\array-...	incompatible types in assignment
11	K:\GiangDay\FU\OOP\BaiTap\array-...	incompatible types in assignment

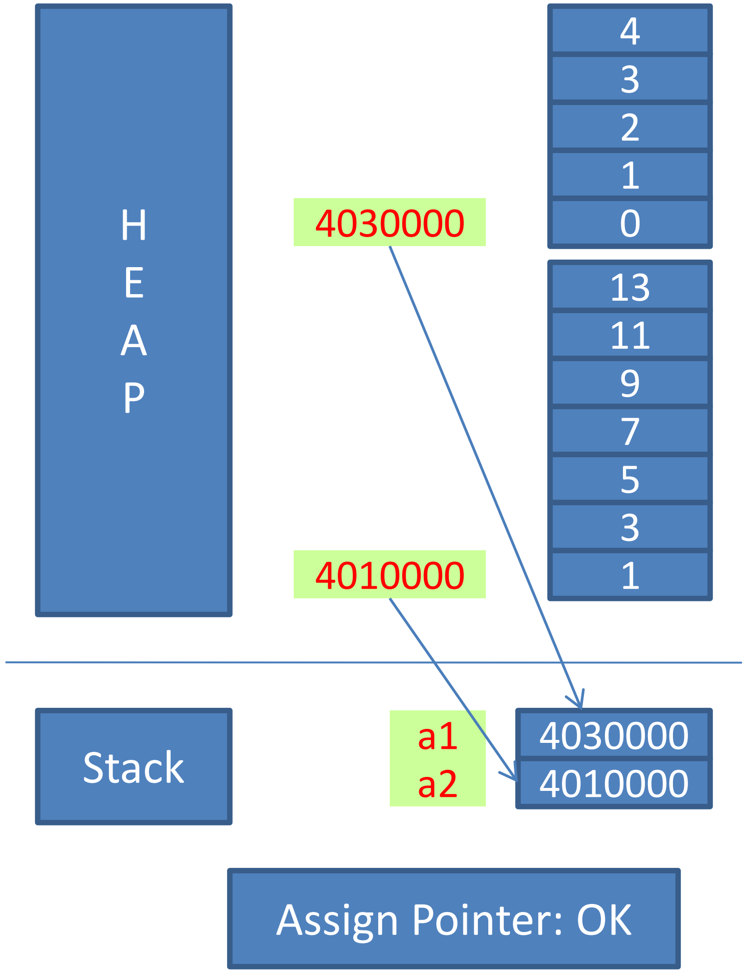
7- May Operators Applied to String?

- The assign operator can act on pointers to dynamic array.

```
#include <stdio.h>
void print (int*a, int n)
{ int i;
  for (i=0;i<n;i++) printf("%d ", a[i]);
}
int main()
{   int *a1 = (int*)calloc(5,sizeof(int));
    int *a2 = (int*)calloc(7,sizeof(int));
    int i;
    for (i=0; i<5; i++) a1[i]=i;
    for (i=0; i<7; i++) a2[i]=2*i+1;
    a1= a2;
    print(a1,7);
    puts("");
    print(a2,5);
    getchar();
    return 0;
}
```



```
1 3 5 7 9 11 13
1 3 5 7 9
```

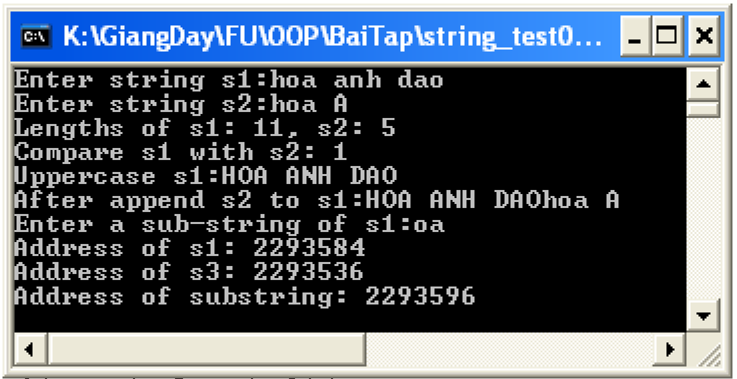


7- Others String Functions: string.h

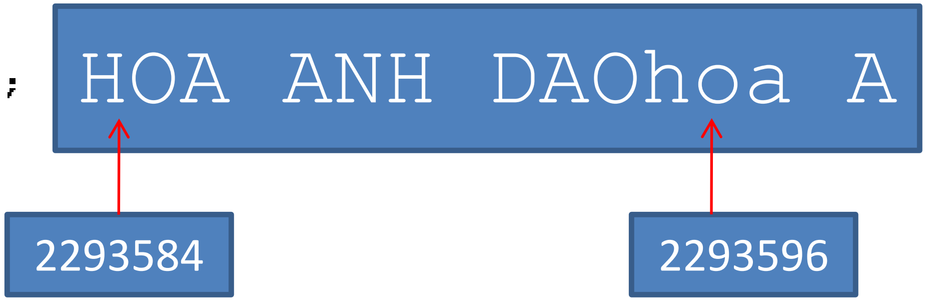
Purpose	Function
Get the length of a string	int ***strlen*** (char s[])
Copy **s**ou**rc**e string to **dest**ination string	char* ***strcpy***(char dest[], char src[])
Compare two strings	int ***strcmp***(char s1[], char s2[]) → -1, 0, 1
Concatenate string src to the end of dest	char* ***strcat***(char dest[], char src[])
Convert a string to uppercase	char* ***strupr***(char s[])
Convert a string to lowercase	char* ***strlwr***(char s[])
Find the address of a substring	char* ***strstr*** (char src[], char subStr[]) → **NULL** if **subStr** does not exist in the **src**.

Others String Functions: string.h

```
#include <stdio.h>
#include <string.h>
int main()
{   char s1[21];
    char s2[21];
    printf("Enter string s1:");
    gets(s1);
    printf("Enter string s2:");
    gets(s2);
    printf("Lengths of s1: %d, s2: %d\n", strlen(s1), strlen(s2));
    printf("Compare s1 with s2: %d\n", strcmp(s1,s2));
    strupr(s1);
    printf("Uppercase s1:%s\n", s1);
    strcat(s1, s2);
    printf("After append s2 to s1:%s\n", s1);
    char s3[10];
    printf("Enter a sub-string of s1:");
    gets(s3);
    char* ptr = strstr(s1, s3);
    printf("Address of s1: %u\n", s1);
    printf("Address of s3: %u\n", s3);
    printf("Address of substring: %u\n", ptr);
    getchar();
    return 0;
}
```



```
Enter string s1:hoa anh dao
Enter string s2:hoa A
Lengths of s1: 11, s2: 5
Compare s1 with s2: 1
Uppercase s1:HOA ANH DAO
After append s2 to s1:HOA ANH DAOhoa A
Enter a sub-string of s1:oa
Address of s1: 2293584
Address of s3: 2293536
Address of substring: 2293596
```



strstr() → NULL if the substring doesn't exist.

8- Some User-Defined String Functions

Purpose	Prototype
Trim blanks at the beginning of a string: " Hello" → "Hello"	char* **lTrim**(char s[])
Trim blanks at the end of a string: "Hello " → "Hello"	char* **rTrim**(char s[])
Trim extra blanks in a string: " I am a student " → "I am a student"	char* **trim** (char s[])
Convert a string to a name: " hoang thi hoa " → "Hoang Thi Hoa"	char* **nameStr**(char s[])

Some user-defined String Functions

0	1	2	3	4	5	6
			H	o	a	NULL
i=0	1	2	3			

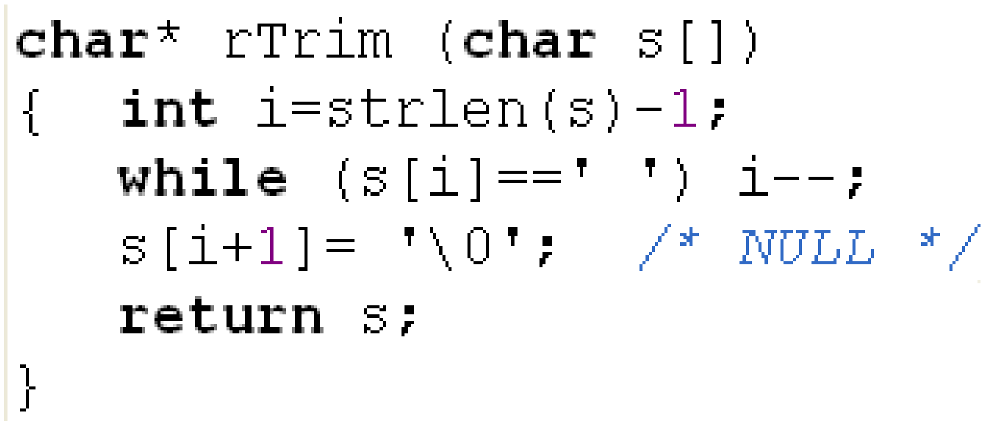
0	1	2	3	4	5	6
H	o	a	NULL	o	a	NULL

```
char* lTrim (char s[])
{   int i=0;
    while (s[i]==' ') i++;
    if (i>0) strcpy(&s[0], &s[i]);
    return s;
}
```

Some user-defined String Functions

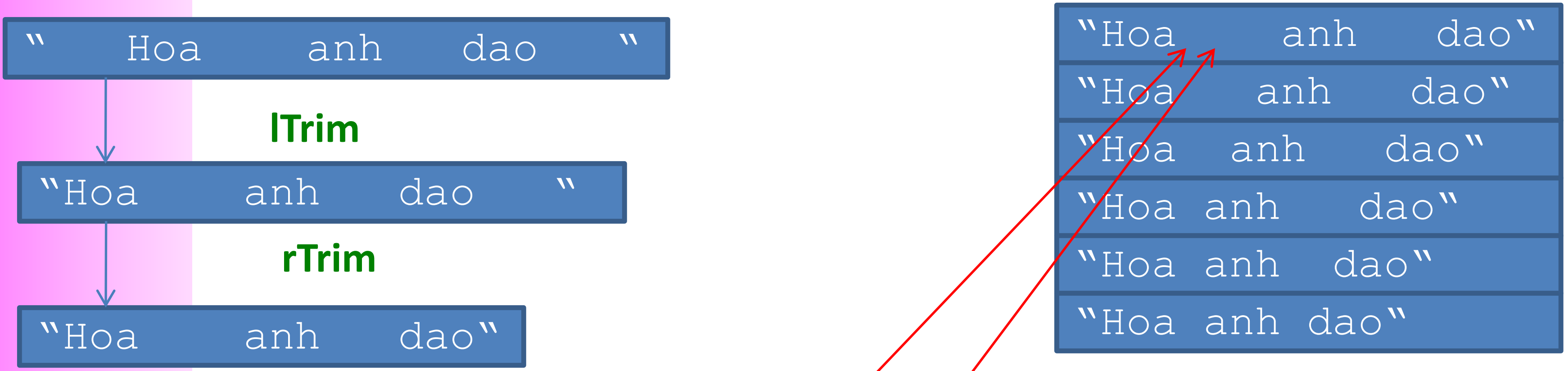
0	1	2	3	4	5	6
H	o	a				NULL
		2	3	4	i=5	

0	1	2	3	4	5	6
H	o	a	NULL			NULL



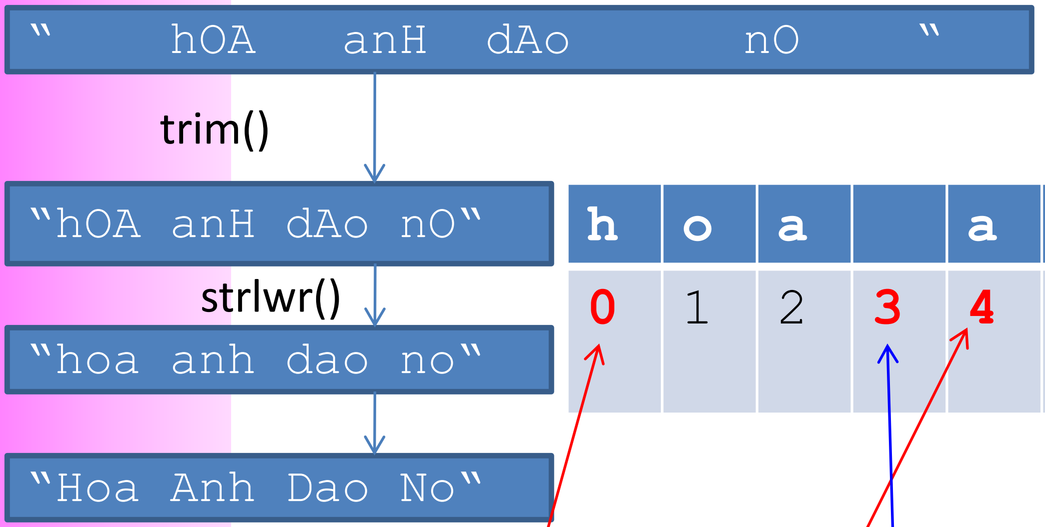
```
char* rTrim (char s[])
{   int i=strlen(s)-1;
    while (s[i]==' ') i--;
    s[i+1]= '\0';   /* NULL */
    return s;
}
```

Some user-defined String Functions


"   Hoa   anh   dao   "
lTrim
"Hoa   anh   dao   "
rTrim
"Hoa   anh   dao"
"Hoa   anh   dao"
"Hoa   anh   dao"
"Hoa   anh   dao"
"Hoa anh   dao"
"Hoa anh   dao"
"Hoa anh dao"

```
char* trim (char s[])
{   rTrim(lTrim(s));
    char *ptr = strstr(s, "  ");
    while (ptr!=NULL)   /* While two blanks exist */
     {   strcpy( ptr, ptr+1);      /* remove one blank */
         ptr = strstr(s, "  ");
     }
     return s;
}
```

Some user-defined String Functions



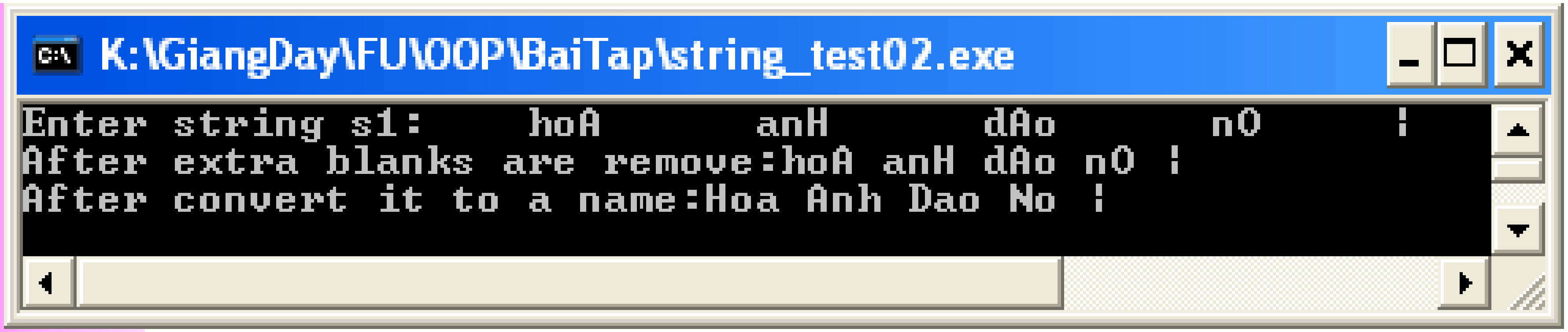
h	o	a		a	n	h		d	a	o		n	o
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```
char* nameStr(char s[])
{  trim(s); /* trim all extra blanks */
   strlwr(s);  /* convert it to lowercase */
   int L = strlen(s);
   int i;
   for (i=0;i<L; i++)
     if (i==0 || (i>0 && s[i-1]==' ')) s[i] = toupper (s[i]);
   return s;
}
```

Some user-defined String Functions

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
char* lTrim (char s[])
{   < your code >
}
char* rTrim (char s[])
{  < your code >
}
char* trim (char s[])
{  < your code >
}
char* nameStr(char s[])
{  < your code >
}
```

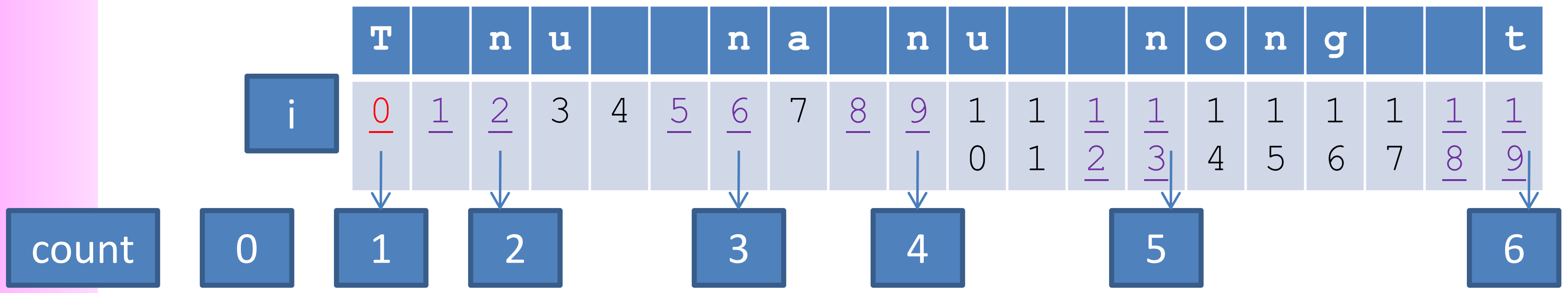
```

int main()
{   char s[21];
    printf("Enter string s1:");
    gets(s);
    trim(s);
    printf("After extra blanks are remove:");
    puts(s);
    nameStr(s);
    printf("After convert it to a name:");
    puts(s);
    getchar();
    return 0;
}
```



```
Enter string s1:     hoA        anH        dAo        nO        !
After extra blanks are remove:hoA anH dAo nO !
After convert it to a name:Hoa Anh Dao No !
```

Some user-defined String Functions

Suppose that only the blank character is used to separate words in a sentence. Implement a function for counting number of words in a sentence.

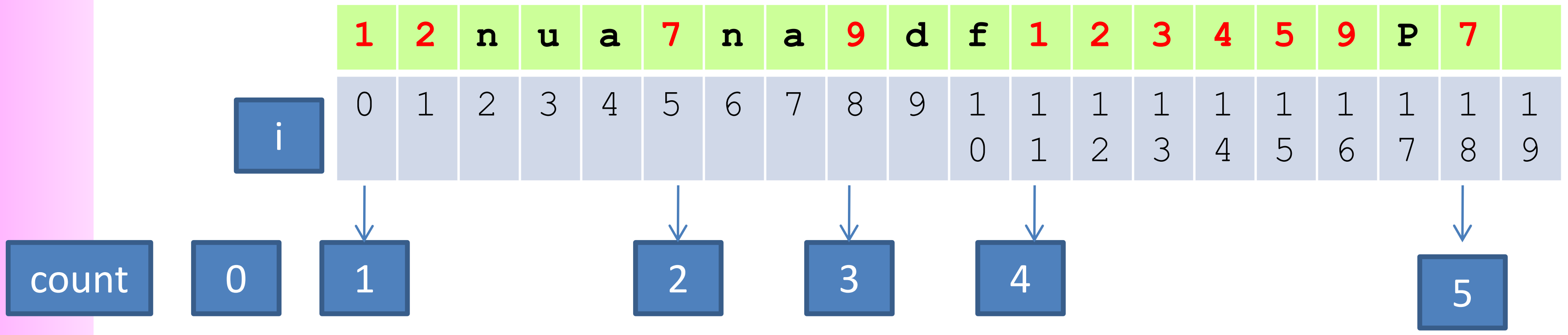


Counting words in a string
Do Yourself

Criteria for increasing count:
- s[i] is not a blank and (i==0 or s[i-1] is a blank)

Some user-defined String Functions

Counting integers in a string



Do Yourself

Criteria for increasing count:
- s[i] is a digit and (i==0 or s[i-1] is not a digit)

Some user-defined String Functions

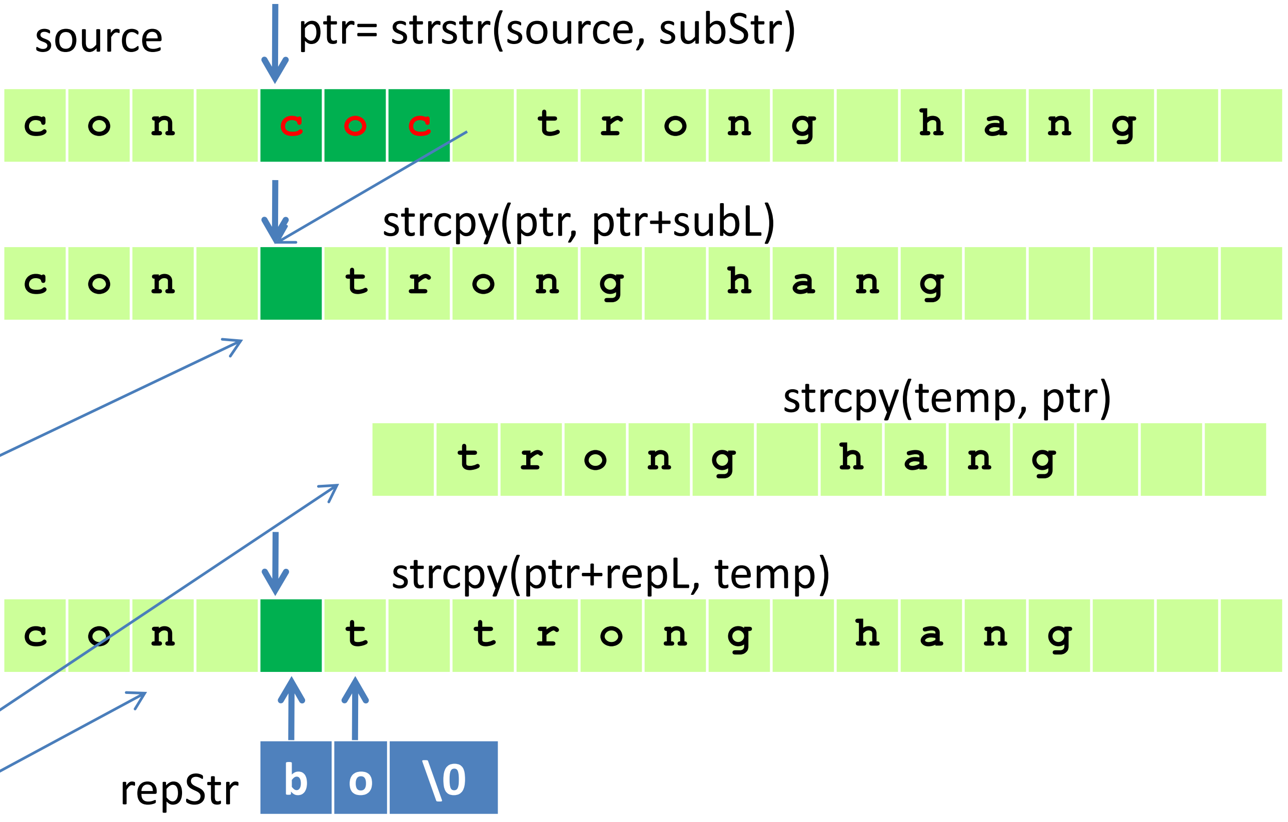
Replace all existences of a sub-string (subStr) in a string (source) by another (repStr)

subStr: "**coc**", subL=3

repStr: "**bo**", repL=2

The function **strcpy** will copy char-by-char from the left to the right of the source to the destination. So, it will work properly when a sub-string is shifted up only.

A temporary string is used when a sub-string is shifted down.


source
ptr= strstr(source, subStr)
c o n   c o c   t r o n g   h a n g
strcpy(ptr, ptr+subL)
c o n   t r o n g   h a n g
strcpy(temp, ptr)
t r o n g   h a n g
strcpy(ptr+repL, temp)
c o n   t   t r o n g   h a n g
repStr
b o \0

for (i=0; i<repL, i++) *(ptr+i) = repStr[i];

Some user-defined String Functions

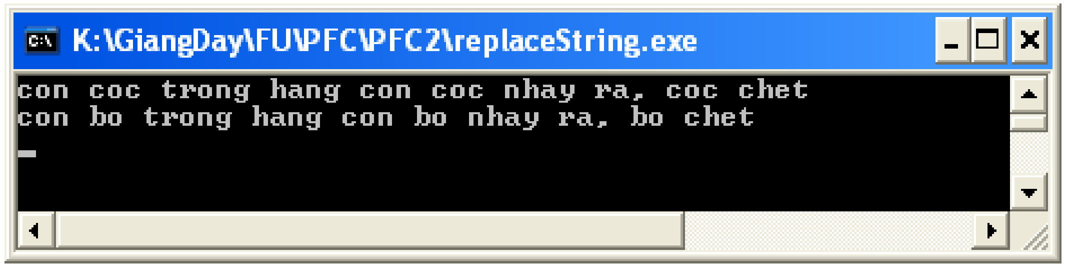
Replace all existences of a sub-string (subStr) in a string (source) by another (repStr)

```
char* replaceAll (char* source, char* subStr, char* repStr)
{   int subL = strlen (subStr);
    int repL = strlen(repStr);
    char temp[100];
    char* ptr = strstr(source, subStr);
    int i;
    while (ptr!=NULL)/* while sudStr exists */
    { strcpy(ptr, ptr+subL);/* Shift subStr up */
      if (repL>0)
       { strcpy(temp, ptr);/* prepare space for repStr*/
         strcpy(ptr+repL, temp);
         /* copy characters in repStr to source */
         for (i=0; i<repL; i++) *(ptr+i)= repStr[i];
       }
      ptr=strstr(source, subStr);
    }
    return source;
}
```

Some user-defined String Functions

Replace all existences of a sub-string (subStr) in a string (source) by another (repStr)

```
int main()
{   char S[80]= "con coc trong hang con coc nhay ra, coc chet";
    char subStr[21]="coc";
    char repStr[21]="bo";
    puts(S);
    replaceAll(S, subStr, repStr);
    puts(S);
    getchar();
    getchar();
    return 0;
}
```



K:\GiangDay\FU\PFC\PFC2\replaceString.exe

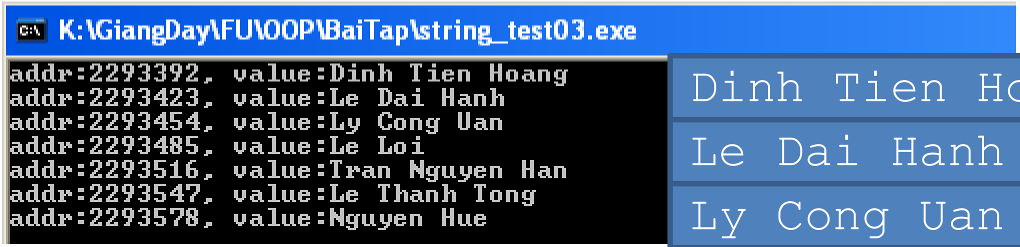
```
con coc trong hang con coc nhay ra, coc chet
con bo trong hang con bo nhay ra, bo chet
_
```

5- Array of Strings

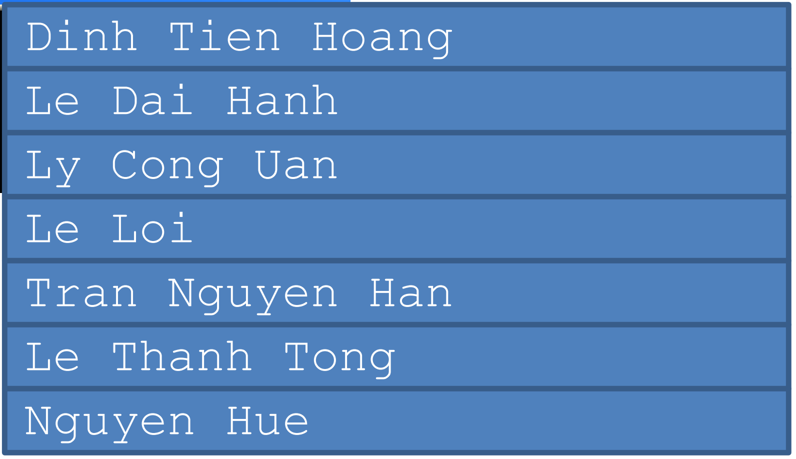
Declaration: char identifier [numberOfString][number_byte_per_string];

Initialization:

```c
#include <stdio.h>
int main()
{ char names[7][31] = { "Dinh Tien Hoang", "Le Dai Hanh",
                         "Ly Cong Uan", "Le Loi",
                        "Tran Nguyen Han", "Le Thanh Tong",
                        "Nguyen Hue" };
  int i;
  for (i=0; i<7; i++)
     printf ("addr:%u, value:%s\n", &names[i], names[i]);
  getchar();
  return 0;
}
```


K:\GiangDay\FU\OOP\BaiTap\string_test03.exe

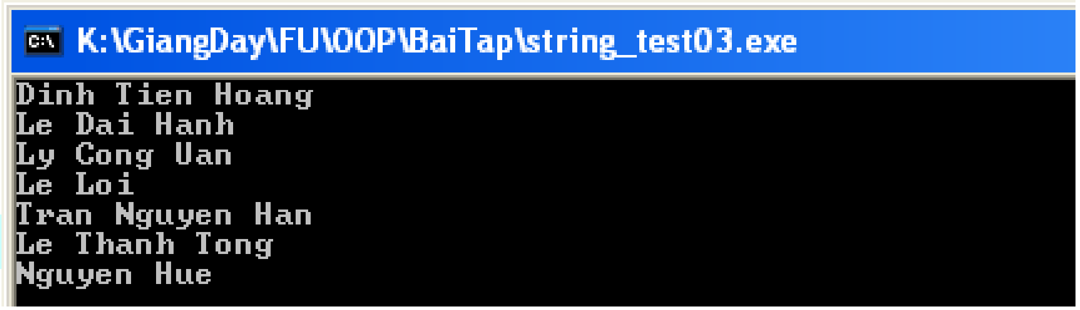
```
addr:2293392, value:Dinh Tien Hoang
addr:2293423, value:Le Dai Hanh
addr:2293454, value:Ly Cong Uan
addr:2293485, value:Le Loi
addr:2293516, value:Tran Nguyen Han
addr:2293547, value:Le Thanh Tong
addr:2293578, value:Nguyen Hue
```



Array of Strings…

Parameter in a function

```c
#include <stdio.h>
void print (char list[][31], int n)
{ int i;
  for (i=0; i<n; i++) puts(list[i]);
}
int main()
{ char names[7][31] = { "Dinh Tien Hoang", "Le Dai Hanh",
                         "Ly Cong Uan", "Le Loi",
                        "Tran Nguyen Han", "Le Thanh Tong",
                        "Nguyen Hue" };
  print(names, 7);
  getchar();
  return 0;
}
```

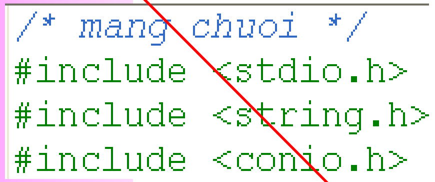


K:\GiangDay\FU\OOP\BaiTap\string_test03.exe

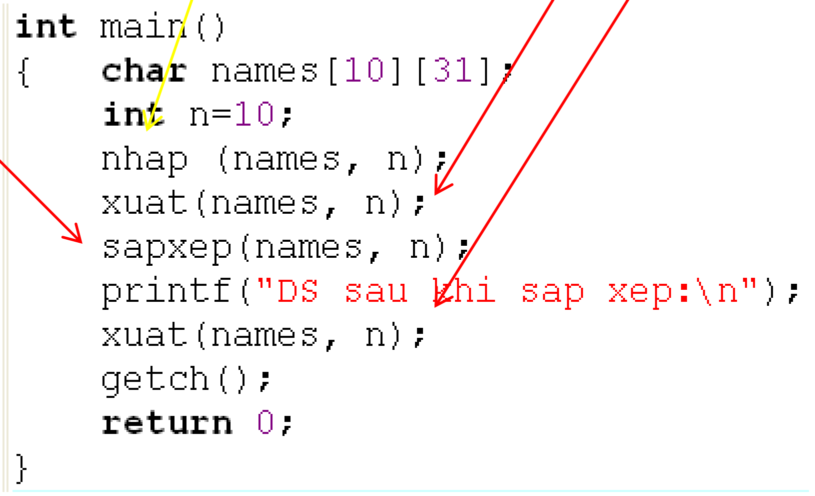
```
Dinh Tien Hoang
Le Dai Hanh
Ly Cong Uan
Le Loi
Tran Nguyen Han
Le Thanh Tong
Nguyen Hue
```

Demo: Array of Names

Write a C program that will accept 10 names, print out the list, sort the list using ascending order, print out the result.



```
/* mang chuoi */
#include <stdio.h>
#include <string.h>
#include <conio.h>
```



```
int main()
{    char names[10][31];
     int n=10;
     nhap (names, n);
     xuat(names, n);
     sapxep(names, n);
     printf("DS sau khi sap xep:\n");
     xuat(names, n);
     getch();
     return 0;
}
```

Demo: Array of Names

```
void nhap(char names[][31], int n)
{ int i;
  for (i=0;i<n;i++)
  {  printf("Nhap ten thu %d/%d:",i+1, n);
     fflush(stdin);
     scanf("%30[^\n]", names[i]);
     strupr(names[i]);
  }
}
void xuat (char names[][31], int n)
{ int i;
  for (i=0;i<n;i++) puts(names[i]);

}
```

```
/* bubble sort- sap xep ten tang dan */
void sapxep(char names[][31], int n)
{ int i,j;
  char t[31]; /* bien hoan vi */
  for (i=0;i<n-1;i++)
     for (j=n-1; j>i; j--)
       /* ten sau < ten truoc */
       if (strcmp(names[j], names[j-1])<0)
       {  strcpy(t, names[j]); /* t= names[j] */
          strcpy(names[j], names[j-1]);
          strpy(names[j-1], t);
       }
}
```

Summary

- String in C is terminated by the NULL character (‘\0’)
- A string is similar to an array of characters.
- All input functions for string will automatically add the NULL character after the content of the string.
- C-operators will operate on simple data types ➔ Function on arrays, strings are implemented to operate on arrays and strings
- If dynamic arrays or strings (using pointers), the assignment can be used on these pointers.

Summary

String Input

- scanf
- gets
- Do yourself using getchar()

String Functions and Arrays of Strings

- Functions
  - strlen
  - strcpy
  - strcmp
  - strcat
  - strstr
- Arrays of Strings
  - Input and Output
  - Passing to Functions
  - Sorting an Array of Names

Q&A

Slot 24- Exercise

Write a C-program that helps user managing a list of 100 student names using the following menu:

1- Add a student

2- Remove a student

3- Search a student

4- Print the list in ascending order

5- Quit

Thank You