

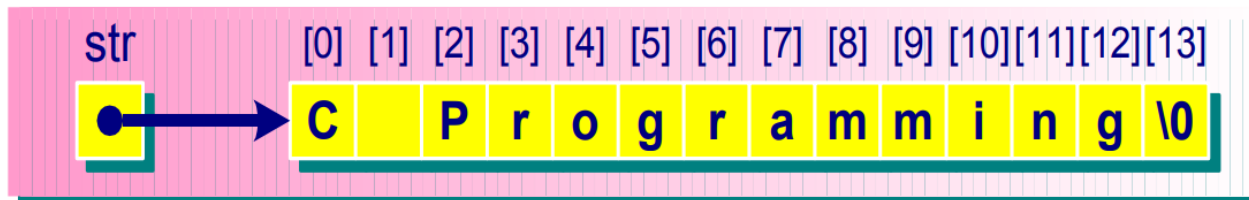
CS100

Introduction to Programming

Lecture 8. Character Strings

String Constants

- A **string** is an array of characters terminated by a NULL character `\0`, e.g. “Hello World”;
- **String constant**: a set of characters in double quotes
 - data type: array of char
 - Automatically terminated with the **NULL** character `\0`
 - Define a string constant
e.g. `#define SHTU “ShanghaiTech University”`
 - As arguments of functions like `printf()`:
e.g. `printf(“Hello, how are you?”);`
 - `char *str = “C Programming”;`
`/* str is a pointer variable */`



String Variables

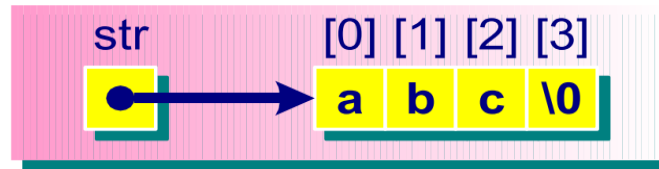
- **String and Array**

```
char str[] = "some text";    // string
```

```
char str[10] = "yes";        // legal
```

```
char str[4] = "four";        // incorrect: space for null character?
```

```
char str[] = {'a', 'b', 'c', '\0'}; equivalent to: char str[] = "abc";
```



- **Note: '\0' differentiates a string from an array of characters**

- Just like other kinds of arrays, the array name **str** gives the **address** of the 1st element of the array:

```
str == &str[0]                // true
```

```
*str == 'a'                    // true
```

```
*(str+1) == str[1] == 'b'      // true
```

String and Pointer

- Can you tell the difference between

```
char str1[] = "\n how are you?"; //using array
```

and

```
char *str2 = "\n how are you?"; //using pointer
```

- **str1**: address constant; **str2**: pointer variable.

Therefore,

<code>++str1;</code>	<code>// not allowed</code>
<code>++str2;</code>	<code>// allowed</code>
<code>str1 = str2;</code>	<code>// not allowed</code>
<code>str2 = str1;</code>	<code>// allowed</code>

String and Pointer

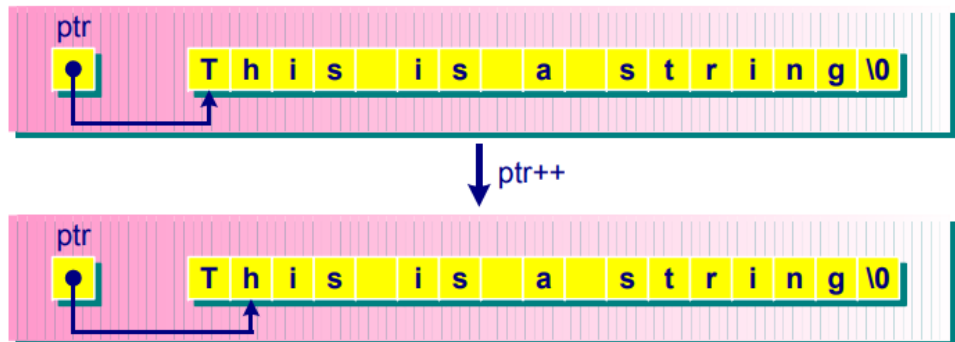
- Assign a string constant to a pointer that points to a char:

```
char *ptr;  
ptr = "This is a string";
```

- When a string constant is assigned to a pointer variable, the C compiler will:
 - allocate memory space to hold the string constant,
 - store the starting address of the string in the pointer variable, and
 - terminate the string with a '\0' character.
- Hence, the **expression value of a string**, e.g. "This is a string", is an **address**.

- For the statement:

```
ptr++;
```



String and Pointer: Example 1

```
int main(void)
{
    char array[10];
    char *ptr1 = "10 spaces", *ptr2;
    ptr1[5] = 'B';           // not OK, ptr1 should point to const
    ptr1 = "OK";             // OK
    ptr1[5] = 'D';           /* not OK, ptr1 points to a 3-char-
                             long string constant */

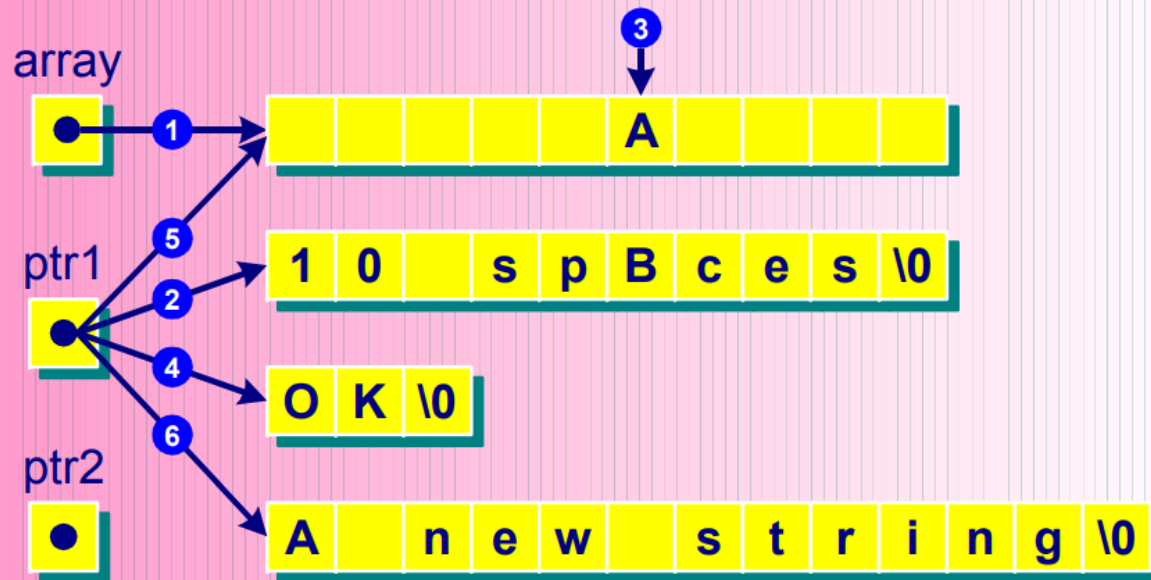
    ptr2[5] = 'C';           // not OK, ptr2 points to nowhere
    *ptr2 = "not OK";        // not OK, type mismatch
    array = "not OK";        /* not OK. C does not support this.
                             Also, right hand side (RHS) is an
                             address while left hand side (LHS)
                             is a constant pointer */

    array[5] = 'A';          // OK
    ptr1 = array;             // OK
    ptr1 = "A new string";    // OK, the array is unchanged
    return 0;
}
```

String and Pointer: Example 1

Memory

- 1 char array[10];
- 2 char *ptr1 = "10 spaces"; ptr1[5] = 'B';
- 3 array[5] = 'A';
- 4 ptr1 = "OK";
- 5 ptr1 = array;
- 6 ptr1 = "A new string";



String and Pointer: Example 2

```
#include <stdio.h>
int main(void)
{
    char *mesg = "Don't be a fool!";    // should be const
    char *copy;                        // should be const

    copy = mesg;
    printf("%s\n", copy);
    printf("mesg=%s; &mesg=%p; value=%p\n",
           mesg, &mesg, mesg);
    printf("copy=%s; &copy=%p; value=%p\n",
           copy, &copy, copy);
    return 0;
}
```

Output:

Don't be a fool!

mesg=Don't be a fool!; &mesg=00AA; value=00AC

copy=Don't be a fool!; ©=038E; value=00AC

String Input: gets()

- **gets()** returns NULL if it fails; otherwise a pointer to the string is returned.
- Make sure enough memory space is allocated to hold the input string, e.g. **name**.

```
#include <stdio.h>
int main(void)
{
    char name[20];    /* string */
    /* read name */
    printf("Hi, what is your name?\n");
    gets(name);
    /* display name */
    printf("Hello, %s.\n", name);
    return 0;
}
```

Output:

Hi, what is your name?

Xiaoming

Hello, Xiaoming.

String Output: puts()

```
#include <stdio.h>
int main(void)
{
    char str[80];    // string with allocated memory
    printf("Enter a line of string: ");
    if (gets(str) == NULL) {
        printf("Error\n");
    }
    puts(str);
    return 0;
}
```

Input: **0123456789 OK**

0	1	2	3	4	5	6	7	8	9			O	K	'\0'	
---	---	---	---	---	---	---	---	---	---	--	--	---	---	------	--

Output: **0123456789 OK**

String Input/Output: scanf() and printf()

- scanf()
 - The scanf() function reads the string up to the next white-space character.
 - scanf() returns **the number of items read** by scanf(), otherwise **EOF** is returned if it fails.
 - Make sure that enough memory space is allocated for input string.
- printf()
 - The printf() function returns **the number of characters transmitted**, otherwise **a negative value** will be returned if it fails.

String Input/Output: scanf() and printf()

```
#include <stdio.h>
int main(void)
{
    char name1[20], name2[20];
    int count;
    printf("Please enter your strings.\n");
    count = scanf("%s %s", name1, name2);
    printf("I read the %d strings %s %s.\n",
           count, name1, name2);
    return 0;
}
```

Output:

Please enter your strings.

[C programming](#)

I read the 2 strings C programming.

Length of a String

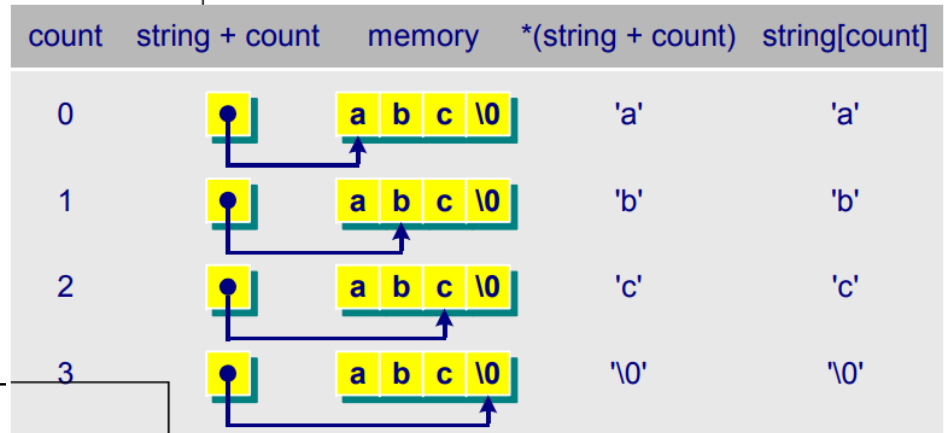
using array notation

```
#include <stdio.h>
int length1(char []);
int length2(char *);
int main(void)
{
    char word[] = "abc";
    char *greeting = "hello";
    printf("The lengths are %d, %d\n",
        length1(word), length2(greeting));
    return 0;
}

int length1(char string[])
{
    int count = 0;
    while (string[count] != '\0')
        count++;
    return count;
}
```

using pointer notation

```
int length2(char *string)
{
    int count = 0;
    while (*(string+count))
        count++;
    return count;
}
```



Output:

The lengths are 3, 5

Lengths of a String v.s an Array

- **Length of an array**

- Usually the number of elements in the whole array allocated in the memory

```
float array[10];  
int len=10;  
int size=len*sizeof(float);
```

- **Length of a string**

- Can be smaller than the array containing it

```
char str[64];  
scanf("%s", str);  
int len = 0;  
while(str[len] != '\0')  
    len++;  
printf("string length = %d", len);
```

String Functions

Must add `#include <string.h>`

Some standard string functions are:

strcat()	appends one string to another
strncat()	appends a portion of a string to another string
strchr()	finds the first occurrence of a specified character in a string
strrchr()	finds the last occurrence of a specified characters in a string
strcmp()	compares two strings
strncmp()	compares two strings up to a specified number of characters
strcpy()	copies a string to an array
strncpy()	copies a portion of a string to an array

String Functions

strcspn()	computes the length of a string that does not contain specified characters
strerror()	maps an error number with a textual error message
strlen()	computes the length of a string
strpbrk()	finds the first occurrence of any specified characters in a string
strtok()	breaks a string into a sequence of tokens

String Functions

- The function prototype of **strlen** is
`unsigned strlen(const char *str);`
- `strlen()` computes and returns the length of the string pointed to by **str**, i.e. the number of characters that precede the terminating **NULL character** `'\0'`.
- The function prototype of **strcat** is
`char *strcat(char *s1, const char *s2);`
- `strcat()` appends a copy of the string pointed to by **s2** to the end of the string pointed to by **s1**. The initial character of **s2** overwrites the NULL character at the end of **s1**.
- `strcat()` returns the value of `s1`.

- The function prototype of **strcpy** is

char *strcpy(char *s1, const char *s2);

- strcpy() copies the string pointed to by s2 into the array pointed to by s1. It returns the value of s1.

- The function prototype of **strcmp** is

int strcmp(const char *s1, const char *s2);

- strcmp() compares the string pointed to by s1 to the string pointed to by s2. It returns an integer >, =, or < zero, according to if the string pointed to by s1 is >, =, or < the string pointed to by s2 alphabetically:

0 if the two strings are equal

> 0 if the first string follows the second string alphabetically,
i.e. the first string is larger

< 0 if the first string comes first alphabetically, i.e. the first
string is smaller

strlen(): Example

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char line[81] = "This is a string";
    printf("The length of the string is %d.\n",
           strlen(line));
    return 0;
}
```

Output:

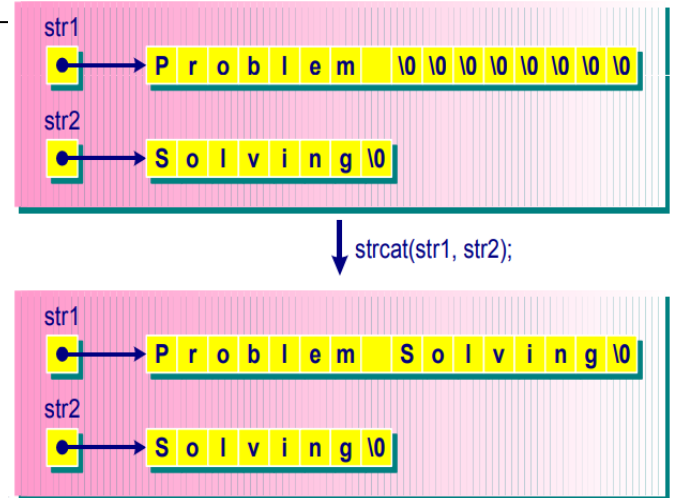
The length of the string is 16.

strcat(): Example

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char str1[40] = "Problem ";
    const char *str2 = "Solving";

    printf("The first string: %s\n", str1);
    printf("The second string: %s\n", str2);
    strcat(str1, str2);
    printf("The combined string: %s\n", str1);
    return 0;
}
```



Output:

The first string: Problem

The second string: Solving

The combined string: Problem Solving

```

#include <stdio.h>
#include <string.h>

int main(void)
{
    char str[81], str2[81];
    int result;

    printf("String Comparison:\n");
    str1[0] = 'A';
    while (str1[0]) {
        printf("Enter the first string: ");
        gets(str1);
        printf("Enter the second string: ");
        gets(str2);
        result = strcmp(str1, str2);
        printf("The comparison result:
                %d\n\n", result);
    }
    return 0;
}

```

strcmp(): Example 1

Output:

String Comparison:

Enter the first string: A

Enter the second string: B

The comparison result: -1

Enter the first string: ABa

Enter the second string: ABA

The comparison result: 1

Enter the first string: A0

Enter the second string: A1

The comparison result: -1

strcmp(): Example 2

```
/* Read a few lines from standard input
and write each line to standard output
with the characters reversed. The input
terminates with the line "ZZZ" */
#include <stdio.h>
#include <string.h>
Void reverse(char *);
int main(void)
{
    char line[132];
    gets(line);
    while (strcmp(line, "ZZZ") != 0) {
        reverse(line);
        printf("%s\n", line);
        gets(line);
    }
    return 0;
}
```

```
void reverse(char *s)
{
    char c, *end;
    end = s + strlen(s) - 1;
    while (s < end) {
        // 2 ends approach center
        c = *s;

        *s++ = *end;
        // or *s = *end; s++;

        *end-- = c;
        // or *end = c; end--;
    }
}
```

Output:

How are you

uoy era woH

ZZZ

strcpy(): Example

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char target[40] = "This is the target string.";
    char source[40] = "This is the source string.";

    puts(target);
    puts(source);
    strcpy(target, source);
    puts(target);
    puts(source);

    return 0;
}
```

Output:

This is the target string.
This is the source string.
This is the source string.
This is the source string.

String Copy

- Implementing strcpy(...)

```
char * strcpy(char *str_dest, const char *str_src)
{
    char *str_ret=str_dest;
    if((str_dest!=NULL) && (str_src!=NULL))
    {
        while(*str_src!='\0') // *str_dest++ = *str_src++
            *str_dest++=*str_src++;

        *str_dest='\0';
    }

    return str_ret;
}
```


String Concatenation

- Implementing strcat(...)

```
char * strcat(char *str_dest, const char *str_src)
{
    char *str_ret=str_dest;
    if((str_dest!=NULL) && (str_src!=NULL))
    {
        while(*str_dest!='\0')
            str_dest++;

        while(*str_src!='\0')
            *str_dest++=*str_src++;

        *str_dest='\0';
    }

    return str_ret;
}
```

ctype.h

Functions

Name	True If Argument is
isalnum	Alphanumeric (alphabetic or numeric)
isalpha	Alphabetic
isctrl	A control character, e.g. Control-B
isdigit	A digit
isgraph	Any printing character other than a space
islower	A lowercase character
isprint	A printing character
ispunct	A punctuation character (any printing character other than a space or an alphanumeric character)
isspace	A whitespace character: space, newline, formfeed, carriage return, etc.
isupper	An uppercase character
isxdigit	A hexadecimal-digit character

- Return **true** (non-zero) if the character belongs to a particular class;
- Return **false** (zero) otherwise.
- Must add

#include <ctype.h>

ctype: Character Conversion Functions

- **toupper()** – maps lowercase character to uppercase;
- **tolower()** – maps uppercase character to lowercase.

```
void modify(char* str)
{
    while (*str != NULL)
    {
        if (isupper(*str))
            *str = tolower(*str);
        else if (islower(*str))
            *str = toupper(*str);
        str++;
    }
}
```

Output:

This is a test.
tHIS IS A TEST.

String to Number Conversions

atof()

- **Prototype:** `double atof(const char *ptr);`
- **Functionality:** converts the string pointed to by the pointer `ptr` into a **double precision floating number**.
- **Return value:** converted value.

atoi()

- **Prototype:** `int atoi(const char *ptr);`
- **Functionality:** converts the string pointed to by the pointer `ptr` into an **integer**.
- **Return value:** converted value.

String to Number Conversions: Example

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main(void)
{
    char ar[80];
    int i;
    double f;

    scanf("%s", ar);
    for (i=0; isdigit(ar[i]); i++);
    if (ar[i] != NULL)
        printf("The input is not a number\n");
    else {
        f = atof(ar);
        printf("Input is %f\n", f);
    }
    return 0;
}
```

Output:

123

Input is 123.000000

Formatted String I/O

sscanf()

- The function *sscanf()* is similar to *scanf()*. The only difference is that *sscanf()* takes input characters from a **string** instead of from the keyboard.
- *sscanf()* can be used **to transform numbers represented in characters/strings**, i.e. “123”, into numbers, e.g. 123, 123.0, of data types int, float, double, etc.

sprintf()

- The function *sprintf()* is similar to *printf()*. The only difference is that *sprintf()* prints output to a string.
- *sprintf()* can be used **to transform numbers into strings**.

Formatted String I/O – Example

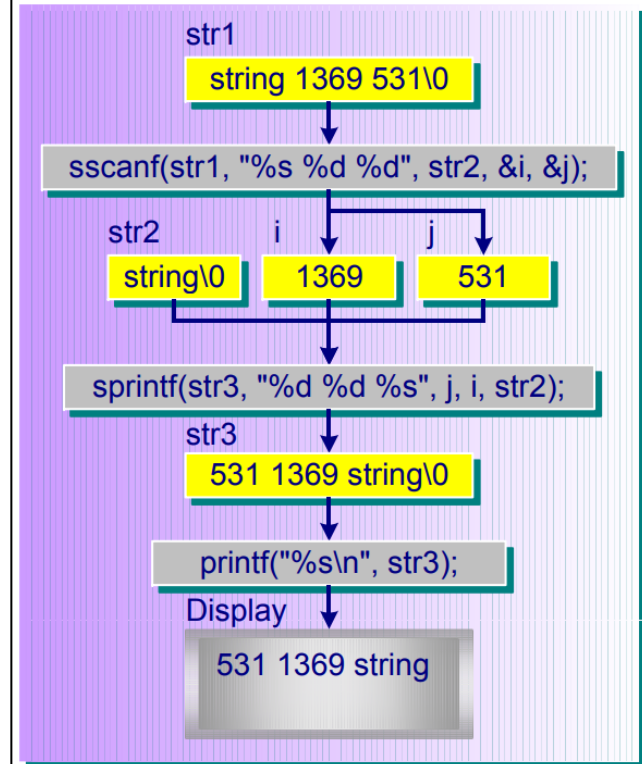
```
#include <stdio.h>
#define MAX_CHAR 80

int main(void)
{
    char str1[MAX_CHAR] = "string 1369 531";
    char str2[MAX_CHAR], str3[MAX_CHAR];
    int i, j;

    sscanf(str1, "%s %d %d", str2, &i, &j);

    sprintf(str3, "%d %d %s", j, i, str2);

    printf("%s\n", str3);
    return 0;
}
```



Output:

531 1369 string

Array of Character Strings

- Arrays of Character Strings

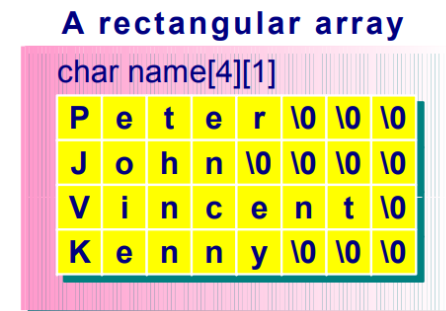
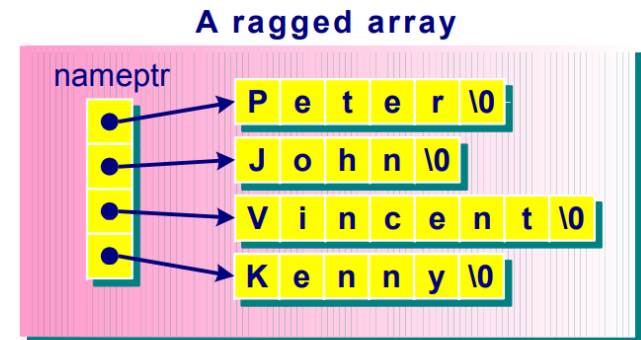
```
char *nameptr[4] = {"Peter",  
    "John", "Vincent", "Kenny"};  
for (i=0; i < 4; i++)  
    printf("nameptr[%d]=%s\n", i,  
        nameptr[i]);
```

=> **nameptr** is a ragged array, i.e. an array of pointers (to save storage)

- Compare **nameptr** with

```
char name[4][8] = {"Peter",  
    "John", "Vincent", "Kenny"};
```

=> **name** is a rectangular array



Array of Strings: Example

```
#include <stdio.h>
int main(void)
{
    const char *nameptr[4] = {"Peter", "John", "Vincent", "Kenny"};
    char name[4][10] = {"Mary", "Victoria", "Susan", "May"};
    int i, j;

    printf("Ragged Array:\n");
    for (i=0; i < 4; i++) {
        printf("nameptr[%d] = %s\n", i,
            nameptr[i]);
    }
    printf("Rectangular Array: \n");
    for (j=0; j < 4; j++) {
        printf("name[%d] = %s\n", j, name[j]);
    }
    return 0;
}
```

Output:

Ragged Array:

nameptr[0] = Peter

nameptr[1] = John

nameptr[2] = Vincent

nameptr[3] = Kenny

Rectangular Array:

name[0] = Mary

name[1] = Victoria

name[2] = Susan

name[3] = May

Command Line Arguments

- The **command line** is the line you type to run your program. Arguments can be given to commands as options. For example, `$cat file1 file2 file3 ...`, where file1, file2, file3, ..., are the arguments for cat.
- User can also supply arguments to his program, i.e.

a.out argument1 argument2 ...

- Arguments to main() function. The syntax to receive these arguments is:

```
main(int argc, char *argv[]) { ... }
```

where **argc** is the argument counter which reports how many words are there in the command line. The command itself, e.g. a.out, ls, cat, ..., is also counted.

- **argv** is the argument value represented by an array of pointers pointing to the input strings.

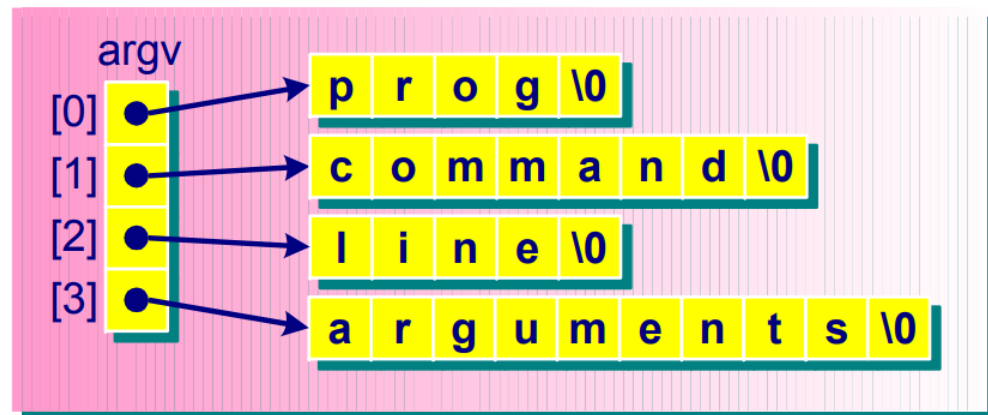
Command Line Arguments: Example

- For example:
\$prog command line arguments

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int count;
    printf("Command Line Arguments:\n");
    printf("argc = %d\n", argc);
    for (count=0; count < argc; count++) {
        printf("argv[%d] = %s\n", count,
               argv[count]);
    }
    return 0;
}
```

Output:

```
argc = 4
argv[0] = prog
argv[1] = command
argv[2] = line
argv[3] = arguments
```



Dynamic Strings

- **Problem with previous string construction**
 - They are all static: constructed in compilation
 - Hard to adapt to run-time string construction
- **Solution**
 - String with dynamic memory (array) allocation
 - Make sure the array end with '\0'

Dynamic Strings

- **How to construct dynamic strings?**

```
#include <stdio.h>
#include <cstring>
```

```
int main(void)
{
    int len=64;
    char* name_str=(char*)malloc(sizeof(char)*len);
    memset(name_str,0,sizeof(char)*len);

    scanf("Input your name: %s", name_str);
    printf("Your name is: %s", name_str);

    free(name_str);

    return 0;
}
```

Dynamic String Concatenation

- How to concatenate two strings?

```
char * strcat(const char *str1, const char *str2)
{
    char *str_ret=NULL;
    if((str_dest!=NULL) && (str_src!=NULL))
    {
        int cat_len=strlen(str1)+strlen(str2);
        str_ret=(char *)malloc(sizeof(char)*(cat_len+1));

        while(*str1!='\0')
            *str_ret++=*str1++;
        while(*str2!='\0')
            *str_ret++=*str2++;

        *str_ret='\0';
    }

    return str_ret;
}
```

Array of Strings

- **Array of statically allocated strings**
 - Basically a 2D array

```
char str_array[10][64]; //an array of 10 strings

for(int i=0;i<10;i++)
    memset(str_array[i],0,sizeof(char)*64);

for(int i=0;i<10;i++)
    scanf("%s", str_array[i]);

for(int i=0;i<10;i++)
    printf("String %d: %s", str_array[i];
```

Array of Strings

- **Array of dynamically allocated strings**

```
char** str_array=NULL;
int str_num=10, str_len=32;

str_array=(float**)malloc(sizeof(float*)*str_num);
for(int i=0;i<str_num;i++)
    str_array[i]=(float*)malloc(sizeof(float)*str_len);

for(int i=0;i<str_num;i++)
    scanf("%s", str_array[i]);

for(int i=0;i<str_num;i++)
    printf("String %d: %s", str_array[i]);

for(int i=0;i<str_num;i++)
    free(str_array[i]);
free(str_array);
```