

Western University Computer Science Department

Part Six: Simple I/O

A series of horizontal lines of varying lengths and shades of gray, stacked on the right side of the slide.

SIMPLE C PROGRAM

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char a;      /* 1 byte   */
    int b;       /* 4 bytes  */
    float c;     /* 4 bytes  */
    double d;    /* 8 bytes  */

    a = 'K';
    b = 37;
    c = 2.5;
    d = 75.3;

    printf( "1st value of a is : %c \n" , a );
    printf( "2nd value of b is : %d \n" , b );
    printf( "3rd value of c is : %f \n" , c );
    printf( "4rd value of d is : %lf \n" , d );

    return 0 ;
}
```

OUTPUT:

1st value of a is : K
2nd value of b is : 37
3rd value of c is : 2.50000000
4th value of d is : 75.50000000

PRINTING in C

In printing, C will view the **content** of a **variable** as a (**generic**) sequence of bits

C does not know (nor care) about the **data type** of the variable

You must tell (instruct) C on how to **interpret** the bit pattern !!!

The **printf()** function is used to print values of all built-in data types in C.

Syntax of the printf() function:

```
printf ( " format string " , value1, value2, ..... );
```

```
printf( "2nd value of b is : %d \n" , b );
```

The "format string" contains instructions on how to interpret each of the values in the parameter list

PRINTING in C

FORMAT STRING:

The format string in the printf() function contains **formatting characters** that instruct the C compiler to print a value in the given format



Formatting Character

Meaning

<code>%d</code>	Print the (next) value as a signed integer value
<code>%u</code>	Print the (next) value as a unsigned integer value
<code>%ld</code>	Print the (next) value as a long signed integer value
<code>%lu</code>	Print the (next) value as a long unsigned integer value
<code>%f</code>	Print the (next) value as a floating point value
<code>%lf</code>	Print the (next) value as a double precision floating point value
<code>%c</code>	Print the (next) value as a character (ASCII code)
<code>%s</code>	Print the (next) value as a string (to be explained later)

PRINTING in C

```
int main( int argc, char* argv[] )
{
    int i = 65, j = 'B';          /* ASCII code for 'B' = 66 */
    float x = 65.0;

    printf( "signed integer i: %d and signed integer j: %d\n", i, j );

    printf( "signed integers i: %c and j: %c as characters ", i, j );
    printf( "using ASCII code.\n" );

    printf( "\n" );
    printf( "float x: %f\n", x );

    return (0);
}
```

signed integer i: 65 and signed integer j: 66
 signed integers i: A and j: B as characters using ASCII code.

float x: 65.000000

PRINTING in C

```
int main( int argc, char* argv[] )
{
    int i = 65, j = 'B';          /* A
    float x = 65.0;

    printf( "signed integer i: %d a

    printf( "signed integers i: %c
    printf( "using ASCII code.\n");

    printf( "\n" );
    printf( "float x: %f\n", x );

    return (0);
}
```

signed integer i: 65 and signed integer j: 66
signed integers i: A and j: B as characters

float x: 65.000000

Label	Address	Value	Binary
	399		
i	400	65	0000 0000
	401		0000 0000
	402		0000 0000
	403		0100 0001
j	404	66	0000 0000
	405		0000 0000
	406		0000 0000
	407		0100 0010
x	408	65.0	0000 0010
	409		0000 0000
	410		0000 0000
	411		0000 0000
	412		0000 0000
	413		0000 0000
	414		0000 0000
	415		0000 0000
	416		
	417		
	418		
	...		

PRINTING in C

WARNING:

The **C compiler** do *not* perform any **type checks** in the **printf()** function call
You must make sure that the **data type** of the **variables** correspond to **formatting character**

```
int main( int argc, char* argv[] )
{
    int i = 65, j = 'B';          /* ASCII code for 'B' = 66 */
    float x = 65.0;

    printf( "signed integer i: %f and signed integer j: %f\n", i, j );

    printf( "signed integers i: %lu and j: %u as characters ", i, j );
    printf( "using ASCII code.\n" );

    printf( "\n" );
    printf( "float x: %d\n", x );

    return (0);
}
```

signed integer i: 0.000000 and signed integer j: 0.000000
 process returned -1073741819 (0xC000000005)

PRINTING in C

WARNING:

The **C compiler** do *not* perform any type checking.
You must make sure that the data type of the variable is correct.

```
int main( int argc, char* argv[] )
{
    int i = 65, j = 'B';          /* ASCII code for 'B' is 66 */
    float x = 65.0;

    printf( "signed integer i: %f and signed integer j: %d\n", i, j );

    printf( "signed integers i: %lu and j: %lu\n", i, j );
    printf( "using ASCII code.\n" );

    printf( "\n" );
    printf( "float x: %d\n", x );

    return (0);
}
```

signed integer i: 0.000000 and signed integer j: 66
process returned -1073741819 (0xC0000000)

Label	Address	Value	Binary
	399		
i	400	65	0000 0000
	401		0000 0000
	402		0000 0000
	403		0100 0001
j	404	66	0000 0000
	405		0000 0000
	406		0000 0000
	407		0100 0010
x	408	65.0	0000 0010
	409		0000 0000
	410		0000 0000
	411		0000 0000
	412		0000 0000
	413		0000 0000
	414		0000 0000
	415		0000 0000
	416		
	417		
	418		
	...		

PRINTING in C

printf() special characters:

The following character sequences have a special meaning when used as printf format specifiers

Formatting Character	Meaning
<code>\a</code>	audible alert
<code>\b</code>	backspace
<code>\f</code>	form feed
<code>\n</code>	newline, or linefeed
<code>\r</code>	carriage return
<code>\t</code>	tab
<code>\v</code>	vertical tab
<code>\\</code>	backslash

Simple Input and Output

END OF PART 1

PRINTING in C

printf() special characters:
examples:

Description	Code	Result
Insert a tab character in a string	<code>printf("Hello\tworld");</code>	Hello world
Insert a newline character in a string	<code>printf("Hello\nworld");</code>	Hello world
Typical use of the newline character	<code>printf("Hello world\n");</code> <i>newline</i>	Hello world <i>~~~~~</i>
A DOS/Windows path with backslash characters	<code>printf("C:\\ \\ Windows\\ \\ System32\\ \\");</code>	C:\\Windows\\System32\\

PRINTING in C

Controlling integer width with printf

The %3d specifier is used with integers, and means a minimum width of three spaces, which, by default, will be right-justified:

printf("%3d", 0);	0
printf("%3d", 123456789);	123456789
printf("%3d", -10);	-10
printf("%3d", -123456789);	-123456789

"%3d", 0 => 000
 ↓

保证输出位置 3.

PRINTING in C

Left-justifying printf integer output

To left-justify integer output with printf, just add a minus sign (-) after the % symbol, like this:

integer may change.

<code>printf("%-3d", 0);</code>	0□□.
<code>printf("%-3d", 123456789);</code>	123456789
<code>printf("%-3d", -10);</code>	-10
<code>printf("%-3d", -123456789);</code>	-123456789

PRINTING in C

The printf integer zero-fill option

To zero-fill your printf integer output, just add a zero (0) after the % symbol, like this:

fill the blank space with 0

<code>printf("%03d", 0);</code>	000
<code>printf("%03d", 1);</code>	001
<code>printf("%03d", 123456789);</code>	123456789
<code>printf("%03d", -10);</code>	-10
<code>printf("%03d", -123456789);</code>	-123456789

PRINTING in C

printf integer formatting

As a summary of printf integer formatting, here's a little collection of integer formatting examples. Several different options are shown, including a minimum width specification, left-justified, zero-filled, and also a plus sign for positive numbers.

Description	Code	Result
At least five wide	<code>printf("%5d", 10);</code>	' 10'
At least five-wide, left-justified	<code>printf("%-5d", 10);</code>	'10 '
At least five-wide, zero-filled	<code>printf("%05d", 10);</code>	'00010'
At least five-wide, with a plus sign	<code>printf("%+5d", 10);</code>	' +10'
Five-wide, plus sign, left-justified	<code>printf("%-+5d", 10);</code>	'+10 '

PRINTING in C

formatting floating point numbers with printf

Here are several examples showing how to format floating-point numbers with printf:

Description	Code	Result
Print one position after the decimal	<code>printf("%.1f", 10.3456);</code>	'10.3'
Two positions after the decimal	<code>printf("%.2f", 10.3456);</code>	'10.35'
Eight-wide, two positions after the decimal	<code>printf("%8.2f", 10.3456);</code>	' 10.35'
Eight-wide, four positions after the decimal	<code>printf("%8.4f", 10.3456);</code>	' 10.3456'
Eight-wide, two positions after the decimal, zero-filled	<code>printf("%08.2f", 10.3456);</code>	'00010.35'
Eight-wide, two positions after the decimal, left-justified	<code>printf("%-8.2f", 10.3456);</code>	'10.35 '
Printing a much larger number with that same format	<code>printf("%-8.2f", 101234567.3456);</code>	'101234567.35'

Simple Input and Output

END OF PART 2

PRINTING in C

printf string formatting

Here are several examples that show how to format **string** output with printf:

Example:

```
char str[ ] = "A message to display";
printf ("%s\n", str);
```

printf expects to receive a string as an additional parameter when it sees **%s** in the format string

Can be from a character array. *char str[]*

Can be another literal string. *"abcd"*

Can be from a character pointer (more on this later).

printf knows how much to print out because of the NULL character at the end of all strings.

When it finds a **\0**, it knows to stop.

char array[] = "a, b, c, d"

array[0] = "a"

array[4] = "\0"

interpret as NULL

所以字符串必须以32位-12

PRINTING in C

printf string formatting

Here are several examples that show how to format **string** output with `printf()`:

```
char str[10]="unix and c";
```

```
printf("%s", str);
printf("\n");
str[6]='\0';
printf("%s", str);
printf("\n");
```

```
printf("\n");
printf(str);
printf("\n");
str[2]='%';
printf(str);
printf("\n");
```

C语言数组加入 “\0” 的问题

转载 wasabikun 2018-04-24 15:27:06 5723 收藏 4

对于字符串后面是否要加'\0'，有以下几种情况：

第一种：char str[10] = {'h','e','l','l','o'};

如果大括号中的字符个数小于字符串数组的长度，则把大括号中的字符赋给数组中前面的元素，其余元素自动赋值为空字符（即'\0'）。

第二种：char str[5] = {'h','e','l','l','o'};

对于大括号中的字符个数和数组长度相同，系统不会自动添加'\0'。对于这种赋值方式，需要手动添加'\0'；

即：char str[6] = {'h','e','l','l','o','\0'};

第三种：char str[] = "hello";

系统会自动在后面添加'\0'；

对于字符串，系统会自动在其后面加'\0'，因此 `const char str[5] = "hello"`，在VS环境下会提示错误：`const char[6]` 不能赋值给 `const char[5]`，但是在GCC不报错；建议最好不要这样写，以及对字符串定义时采用第三种方式，或者使用 `char *str = "hello"`；

PRINTING in C

printf string formatting
Here are several examples that show how to format **string** output with `printf()`:

printf string formatting
Here are several examples that show how to format **string** output with `printf()`:

```
char str[10]="unix and c";
```

```
printf("%s", str);
```

```
printf("\n");
```

```
str[6]='\0';
```

```
printf("%s", str);
```

```
printf("\n");
```

```
printf("\n");
```

```
printf(str);
```

```
printf("\n");
```

```
str[2]='%';
```

```
printf(str);
```

```
printf("\n");
```

[illegible]

PRINTING in C

printf string formatting

Here are several examples that show how to format **string** output with `printf()`:

```
char str[11]="unix and c";
```

```
printf("%s", str);
printf("\n");
str[6]='\0';
printf("%s", str);
printf("\n");
```

```
printf("\n");
printf(str);
printf("\n");
str[2]='%';
printf(str);
printf("\n");
```

Label	Address	Value
str[0]	400	u
str[1]	401	n
str[2]	402	i
str[3]	403	x
str[4]	404	[space]
str[5]	405	a
str[6]	406	n
str[7]	407	d
str[8]	408	[space]
str[9]	409	c
str[10]	410	\0

PRINTING in C

printf string formatting

Here are several examples that show how to format **string** output with `printf()`:

```
char str[11]="unix and c";
```

```
printf("%s", str);
```

```
printf("\n");
```

```
str[10]='\X';
```

```
printf("%s", str);
```

```
printf("\n");
```

```
str[6]='\0'; → null
```

```
printf("%s", str); unix ad c
```

```
printf("\n");
```

```
printf("\n");
```

```
printf(str);
```

```
printf("\n");
```

```
str[2]='%';
```

```
printf(str);
```

```
printf("\n");
```

Label	Address	Value
str[0]	400	u
str[1]	401	n
str[2]	402	i
str[3]	403	x
str[4]	404	[space]
str[5]	405	a
str[6]	406	n
str[7]	407	d
str[8]	408	[space]
str[9]	409	c
str[10]	410	X

PRINTING in C

- printing with puts()

The `puts` function is a much simpler output function than `printf()` for string printing.

Prototype of `puts` is defined in `stdio.h`

```
int puts(const char * str)
```

This is more efficient than `printf()`

Because your program doesn't need to analyze the format string at run-time.

For example:

```
char sentence[] = "The quick brown fox";  
puts(sentence);
```

Prints out:

The quick brown fox

PRINTING in C

printf string formatting

Here are several examples that show how to format string output with printf:

Description	Code	Result
A simple string	<code>printf("%s", "Hello");</code>	'Hello'
A string with a minimum length	<code>printf("%10s", "Hello");</code>	' Hello'
Minimum length, left-justified	<code>printf("%-10s", "Hello");</code>	'Hello '

Simple Input and Output

END OF PART 3

The String Library in C

String functions are provided in an ANSI standard string library.

Access this through the include file:

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

Includes functions such as:

- Computing length of string

- Copying strings

- Concatenating strings

This library is guaranteed to be there in any ANSI standard implementation of C.

The String Library in C

strlen returns the length of a NULL terminated character string:

```
int count;  
char d[8] = "Magic";  
/* char: 1 byte */
```

Defined in string.h

```
count = strlen(d);
```

- Returns 5 (even though there are 6 values if you

[illegible]

The String Library in C

`strcpy` copies a character string into another string:

A copy of `source` is made at `destination`

`source` should be NULL terminated

`destination` should have enough room

(its length should be at least the size of `source`)

The return value also points at the `destination`.

The String Library in C

strcpy example

output:

C programming

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

```
int main() {
    char str1[20] = "
    char str2[20];
```

```
    // copying str1 to str2
    strcpy(str2, str1);
```

```
    puts(str2); // C programming
```

```
    return 0;
```

```
}
```

Note: When you use strcpy()

the size of the destination string should be large enough to store the copied string.

Otherwise, it may result in **undefined behavior**

The String Library in C

`strcat` : included in `<string.h>`:

Appends a copy of `str2` to the end of `str1`

A pointer equal to `str1` is returned

Ensure that `str1` has sufficient space for the concatenated string!

Array index out of range will be the most popular bug in your C programming career.

The String Library in C

strcat example

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[100] = "This

    // concatenates str1 and str2
    // the resultant string is stored in str1.
    strcat(str1, str2);

    puts(str1);
    puts(str2);

    return 0;
}
```

output:

```
This is programiz.com
programiz.com
```

Note: When we use strcat()

the size of the destination string should be large enough to store the resultant string.

If not, we will get the segmentation fault error.

The String Library in C

C strings can be compared for equality or inequality

If they are equal - they are ASCII identical

If they are unequal the comparison function will return an int that is interpreted as:

$str1 - str2 \begin{cases} < 0 : \text{str1 is less than str2} \\ = 0 : \text{str1 is equal to str2} \\ > 0 : \text{str1 is greater than str2} \end{cases}$

The String Library in C

Basic comparison functions:

```
int strcmp (str1, str2);
```

Does an ASCII comparison one char at a time until a difference is found between two chars in the same position.

Return value is as stated before

If both strings reach a '\0' at the same time, they are considered equal.

```
int strncmp (str1, str2, n);
```

Compares `n` chars of `str1` and `str2`

Continues until `n` chars are compared or

The end of `str1` or `str2` is encountered

Also have `strcasecmp()` and `strncasecmp()`

which do the same as above, but ignore case in letters.

The String Library in C

strcmp example

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

int main()
{
    char str1[] = "abcd", str2[] = "ABCD", str3[] = "abcd";
    int result;

    // comparing strings str1 and str2
    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);

    // comparing strings str1 and str3
    result = strcmp(str1, str3);
    printf("strcmp(str1, str3) = %d\n", result);

    return 0;
}
```

output:

```
strcmp(str1, str2) = 32
strcmp(str1, str3) = 0
```

The first unmatched character between string str1 and str2 is third character.

The ASCII value of 'c' is 99
and the ASCII value of 'C' is 67.
so, when strings str1 and str2 are compared,
the return value is 32.

When strings str1 and str3 are compared,
the result is 0 because both strings are identical.

The String Library in C

There are a number of searching functions:

```
char * strchr (char * str, int ch) ;
```

`strchr` search `str` until `ch` is found or NULL character is found instead.

If found, a (non-NULL) pointer to `ch` is returned.

Otherwise, NULL is returned instead.

You can determine its location (index) in the string by:

Subtracting the value returned from the address of the start of the string

More pointer arithmetic ... more on this later!

The String Library in C

strchr example

output:

String after |.| is - |.tutorialspoint.com|

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

int main () {
    const char str[] = "http
const char ch = '.';
char *ret;

ret = strchr(str, ch);

printf("String after |%c| is - |%s|\n", ch, ret);

return(0);
}
```

strchr() finds the first '.' after www

strchr() returns a pointer

(pointer???)
- later ...

The String Library in C

String functions are provided in an ANSI standard string library.

Access this through the include file:

```
#include <stdlib.h>
```

This header defines one variable type, one macro, and various functions for manipulating arrays of characters.

This library is guaranteed to be there in any ANSI standard implementation of C.

double atof(const char *str)

Converts the string pointed to, by the argument *str* to a floating-point number (type double).

int atoi(const char *str)

Converts the string pointed to, by the argument *str* to an integer (type int).

long int atol(const char *str)

Converts the string pointed to, by the argument *str* to a long integer (type long int).

Simple Input and Output

END OF PART 4

USER INPUT in C

Reading in value of the built-in data types

The **scanf()** function is used to read in **values** of *all* built-in data types in C.

Syntax of the scanf() function:

```
scanf ( " format string " , &var1, &var2, ..... );
```

```
scanf ( "%d" , &x );
```

The **format string** in the **scanf()** function contains **formatting characters** that instruct the **C compiler** to *read in* a **value** and **store** it in the given **representation (encoding memory)**

FORMAT STRING:

The format string in the scanf() function contains the exact same **formatting characters** that are used by the printf() function to print a value in the given format

USER INPUT in C

```
int main( int argc, char* argv[] )
{
    int a;
    float y;

    printf( "Enter an integer value:" );
    scanf( "%d", &a );
    printf( "a = %d\n", a );

    printf( "Enter a floating point value:" );
    scanf( "%f", &y );
    printf( "y = %f\n", y );
}
```

Enter an integer value: **37**

a = 37

Enter a floating point value: **3.14159**

y = 3.141590

USER INPUT in C

```
int main( int argc, char* argv[] )
{
    int a;
    float y;

    printf( "Enter an integer value\n" );
    scanf( "%d", &a );
    printf( "a = %d\n", a );

    printf( "Enter a floating point value\n" );
    scanf( "%f", &y );
    printf( "y = %f\n", y );
}
```

Enter an integer value: **37**

a = 37

Enter a floating point value: **3.14159**

y = 3.141590

Label	Address	Value	Binary
	399		
a	400	37	0000 0000
	401		0000 0000
	402		0000 0000
	403		001 0001
y	404	3.14159	0000 0100
	405		1100 1011
	406		0010 1111
	407		000 00000
	408		
	409		
	410		
	411		
	412		
	413		
	414		
	415		
	416		
	417		
	418		
	...		

USER INPUT in C

Reading in value of the built-in data types

The **scanf()** function is used to read in **values** of *all* built-in data types in C.

To read a string include:

%s scans up to but not including the “next” white space character

%ns scans the next **n** characters or up to the next white space character, whichever comes first

Example:

```
scanf ("%s%s%s", s1, s2, s3);
```

```
scanf ("%2s%2s%2s", s1, s2, s3);
```

Note: No ampersand(&) when inputting strings into character arrays!
(We'll explain why later ...)

USER INPUT in C

Reading in value of the built-in data types

The **gets()** function gets a **line** from standard input in C.

The prototype is defined in **<stdio.h>**

```
char *gets(char *str)
```

str is a pointer to the character array.
Returns NULL upon failure.

Otherwise, it returns the pointer to the character array.

Otherwise, it returns the pointer to the character array.

```
char your_line[100];
printf("Enter a line:\n");
gets(your_line);
puts("Your input follows:\n");
puts(your_line);
```

You can overflow your string buffer, so be careful!

Difference between gets() and scanf()

gets ()

read a line

scanf ("%s" ,...)

read up to the next space

USER INPUT in C

Reading in value of the built-in data types

The **scanf()** function is used to read in **values** of *all* built-in data types in C.

Syntax of the scanf() function:

```
scanf ( &var1, &var2, ..... );
```

The **&** character is the "**reference**" operator of the C programming language

The expression **&x** means: the **address** of the variable **x**

You **must** pass the **address** of a variable to the **scanf()** function for reading operations.

Simple Input and Output

END OF PART 5