CSC180: Lecture 9

Wael Aboulsaadat

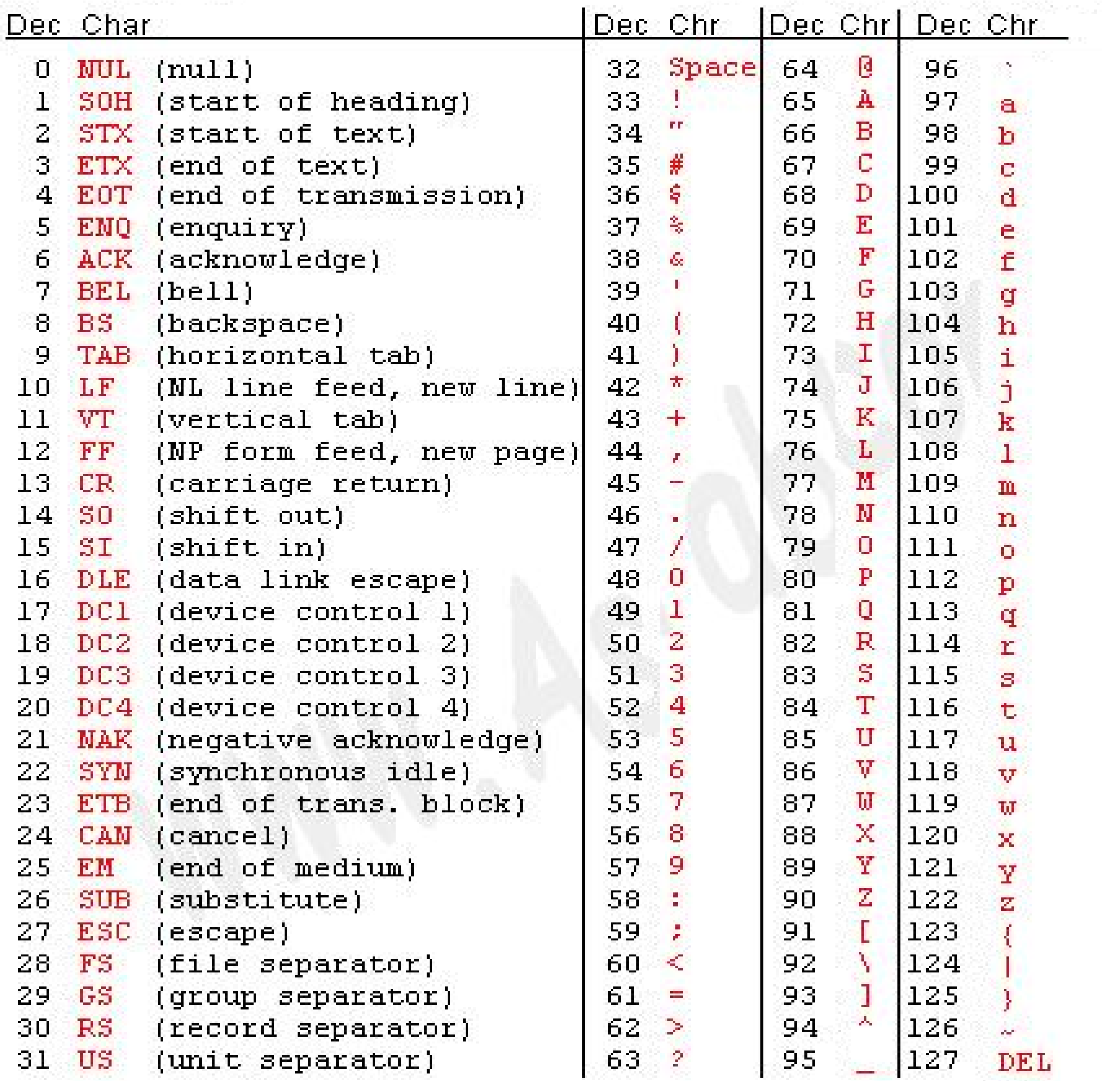
wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Data types

ASCII Table



Dec	Char	Dec	Chr	Dec	Chr	Dec	Chr
0	NUL (null)	32	Space	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

char ← → int

- The following actions are possible but generally not recommended!
- It is possible to store char values in integer variables

```
int value = 'A';
```

value will contain an integer representing 'A'

- It is possible to store int values in char variables

```
char letter = 65;
```

I/O

Output via `printf`

In C, we output to standard output using a `printf` statement:

```
printf("This will be output to stdout.\n");
```

A `printf` statement can output a string literal, but it can also output the value of a variable, a literal constant or a named constant:

```
printf("%d", number_of_students);
```

The statement above outputs to `stdout` (the terminal screen) the value of a variable named `number_of_students` of type `int`

Placeholders

```
printf("%d", number_of_students);
```

The %d is known as a ***placeholder***: it holds the place of the value of the variable that we actually want to output.

Formatted Output with printf

- Placeholders:

%d -- displays an integer

%u -- displays Unsigned integer

%l -- displays a "long“

%e -- displays a floating point value in exponential notation

%f -- displays a floating point value

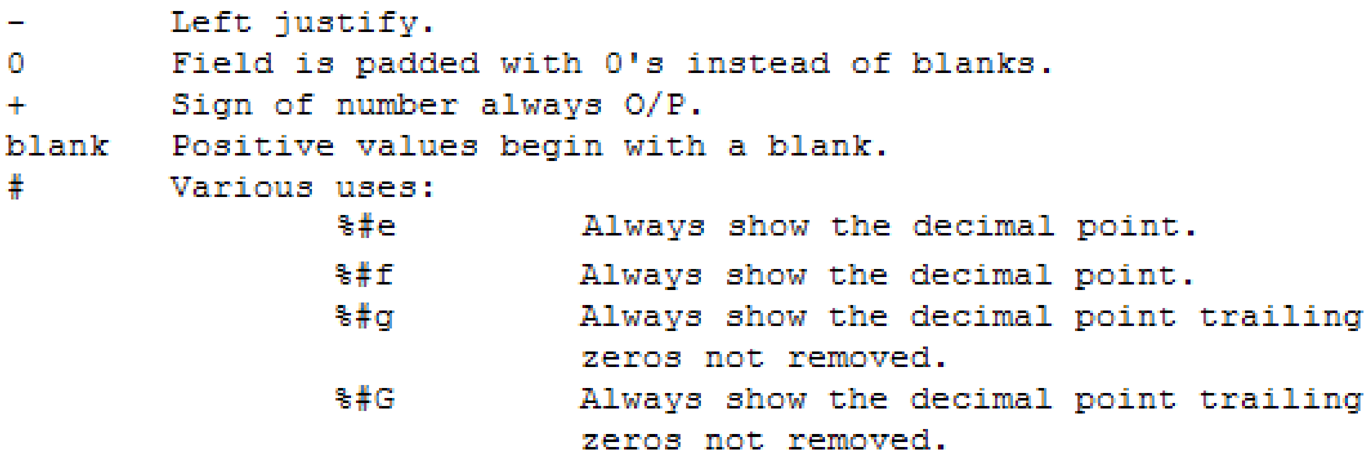
%c -- displays a single character

%s -- displays a string of characters

Formatted Output with printf

- Flags used with place holders:

  flags are placed between the % and the field width or format identifier (more than one flag can be used) e.g. $\%\underbrace{\qquad}_{\textit{flag}}d$



```
-        Left justify.
0        Field is padded with 0's instead of blanks.
+        Sign of number always O/P.
blank    Positive values begin with a blank.
#        Various uses:
                %#e          Always show the decimal point.
                %#f          Always show the decimal point.
                %#g          Always show the decimal point trailing
                             zeros not removed.
                %#G          Always show the decimal point trailing
                             zeros not removed.
```

Mixing Literal Text and Variables' Values

```
printf("The %d federal income tax on $%f\n",
         tax_year, income);
```

means:

- Output to `stdout` (the terminal screen)
- the literal text `"The "`, and then
- the value of the `int` variable named `tax_year`, and then
- the literal text `" federal income tax on $"`, and then
- the value of the `float` variable named `income`, and then
- a newline.

Placeholder & Variable in Same Statement

```
/* These printfs are GOOD GOOD GOOD! */
 printf("f1=%f, ", f1);
 printf("i1=%d, GOOD!\n", i1);


/* These printfs are BAD  BAD  BAD!  */
 printf("f2=%f, i2=%d, ");
 printf("BAD!\n", f2, i2);
```

Input via scanf

The **`printf`** statement **outputs** to **`stdout`** (the terminal screen).

Likewise, the **`scanf`** statement **inputs** from **`stdin`** (a user typing at the keyboard).

The `scanf` statement has a somewhat strange syntax:

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

Input via scanf

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

This statement says:

- input from `stdin` (a user typing at the keyboard)
- an `int` value
- and place it into the memory location associated with the `int` variable named `height_in_cm`.

Input via `scanf`: Ampersand Before Variable

The `scanf` statement has a somewhat strange syntax:

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

Notice the **ampersand** & before the name of the variable that you're inputting into.

Input via scanf Example

```
#include <stdio.h>

int main ()
{
    int height_in_cm;

    printf("What’s my height in centimeters?\n");
    scanf("%d", &height_in_cm);
    printf("My height is %d cm.\n", height_in_cm);
}

% gcc -o read_variable read_variable.c
% read_variable
What’s my height in centimeters?
160
My height is 160 cm.
```

Reading Multiple Variables with a Single `scanf`

C allows inputting multiple variables per scanf statement. **At runtime,** when the user types in the input values, they can separate the individual input values

- by blank spaces, and/or
- by tabs, and/or
- by carriage returns (newlines).

Blank spaces, tabs and carriage returns, as a group, are known as ***white space***.

Multiple Variables per `scanf` Example #1

```c
#include <stdio.h>

int main ()
{
    float average_height_in_m;
    int number_of_people;


    printf("How many people are there in CS180,\n");
    printf("and what is their average height ?\n");

    scanf("%d %f",&number_of_people, &average_height_in_m);

    printf("There are %d people\n", number_of_people);
    printf(" with an average height of %f m.\n",
                                average_height_in_m);


}
```

`printf vs scanf`

- `printf`
  - outputs
  - to `stdout`
  - **CAN** (and typically does) contain literal text as well as placeholders
  - typically **DOES** end with a newline
  - variable names after the string literal **CANNOT** be preceded by &

- `scanf`
  - inputs
  - from `stdin`
  - **CANNOT** contain literal text, other than spaces to separate the placeholders (which are **REQUIRED**)
  - **CANNOT** contain a newline
  - variable names after the string literal **MUST** be preceded by &

Arrays

Introduction to Arrays

- An array is used to process a collection of data of the same type
  - Examples: A list of names
    A list of temperatures
- Why do we need arrays?
  - Imagine keeping track of 5 test scores, or 100, or 1000 in memory
    - How would you name all the variables?
    - How would you process each of the variables?

Declaring an Array

- An array, named score, containing five variables of type int can be declared as

  ```
  int score[ 5 ];
  ```

- This is like declaring 5 variables of type int:

  score[0], score[1], … , score[4]

- The value in brackets is called
  - A subscript
  - An index

The Array Variables

- The variables making up the array are referred to as
  - Indexed variables
  - Subscripted variables
  - Elements of the array
- The number of indexed variables in an array is the declared size, or size, of the array
  - The largest index is one less than the size
  - The first index value is zero

Array Variable Types

- An array can have indexed variables of any type

- All indexed variables in an array are of the same type
  - This is the base type of the array

- An indexed variable can be used anywhere an ordinary variable of the base type is used

Using [ ] With Arrays

- In an array declaration, [ ]'s enclose the size of the array such as this array of 5 integers:

```
int score [5];
```

- When referring to one of the indexed variables, the [ ]'s enclose a number identifying one of the indexed variables
  - score[3] is one of the indexed variables
  - The value in the [ ]'s can be any expression that evaluates to one of the integers
    0 to (size -1)

Indexed Variable Assignment

- To assign a value to an indexed variable, use the assignment operator:

```
int n = 2;
score[n + 1] = 99;
```

  - In this example, variable score[3] is assigned 99

Loops And Arrays

- for-loops are commonly used to step through arrays

  First index is 0

  Last index is (size – 1)

  - Example:

    ```
    for (i = 0; i < 5; i++)
    {
            printf( score[i] );
    }
    ```