Character, String, and I/O

Review and Motivation

What are data types that you have learnt?

- Numbers:
  - integers (int, short, long, etc.) with signed/unsigned
  - floating point numbers (double, etc.)
- Booleans: zero and non-zero (in C)

What other important kind of data we need?

- Text – characters, alphabets, symbols, etc.

Outline

- What is a character?
- Character input and output
- Character operations
- What is a string?
- String operations

Character type – char

American **S**tandard **C**ode for **I**nformation **I**nterchange

- Characters are part of our human languages
- An interesting topic called "**character encoding**"

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

In ASCII, the character 'a' is represented as a hexadecimal value 0x61, i.e., decimal value 97 (which is 16x6+1)

Further reading: https://en.wikipedia.org/wiki/ASCII

Character type – char

American **S**tandard **C**ode for **I**nformation **I**nterchange

Let's say we have a piece of text:

```
Dear Class: Now is week 6 already.
Have you continued to work hard, read book,
and do more programming exercises.
```

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

How is it stored in the computer?

```
 0: 44 65 61 72 20 43 6C 61  73 73 3A 20 4E 6F 77 20
10: 69 73 20 77 65 65 6B 20  36 20 61 6C 72 65 61 64
20: 79 2E 0D 0A 48 61 76 65  20 79 6F 75 20 63 6F 6E
30: 74 69 6E 75 65 64 20 74  6F 20 77 6F 72 6B 20 68
40: 61 72 64 2C 20 72 65 61  64 20 62 6F 6F 6B 2C 0D
50: 0A 61 6E 64 20 64 6F 20  6D 6F 72 65 20 70 72 6F
60: 67 72 61 6D 6D 69 6E 67  20 65 78 65 72 63 69 73
70: 65 73 2E
```

Encoding

E.g., 44 (hex) is 0100 0100 (binary) and 65 is 0110 0101

Character type – char

American **S**tandard **C**ode for **I**nformation **I**nterchange

```
Dear Class: Now is week 6 already.
Have you continued to work hard, read book,
and do more programming exercises.
```

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

```
 0: 44 65 61 72 20 43 6C 61  73 73 3A 20 4E 6F 77 20
10: 69 73 20 77 65 65 6B 20  36 20 61 6C 72 65 61 64
20: 79 2E 0D 0A 48 61 76 65  20 79 6F 75 20 63 6F 6E
30: 74 69 6E 75 65 64 20 74  6F 20 77 6F 72 6B 20 68
40: 61 72 64 2C 20 72 65 61  64 20 62 6F 6F 6B 2C 0D
50: 0A 61 6E 64 20 64 6F 20  6D 6F 72 65 20 70 72 6F
60: 67 72 61 6D 6D 69 6E 67  20 65 78 65 72 63 69 73
70: 65 73 2E
```

Decoding

Further reading -> why hex format? https://hackaday.com/tag/hex-editor/

Character type – char

- In C, character type is for storing integers!

```
char c ;
c = 'a' ;
```

The character **a** is assigned to variable **c**.

The alphabet that is enclosed by a pair of single quotes means a **character constant**.

Unlike double quotes, **only one alphabet** can be enclosed between single quotes.

Last but not least, `sizeof(char)` is 1.

Character type – char

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- As a matter of fact, **a character is a number**.

```c
int main( void )
{
    char c   ;
    int  num ;

    c   = 'a' ;
    num =  c  ;

    printf( "%c\n" , c   );
    printf( "%d\n" , num );

    return 0 ;
}
```

This line is absolutely correct because…
"*a character is a number!*"

```
Output:
a
97
```

Character type – char

- As a matter of fact, **a character is a number**.

```
1  int main( void )
2  {
3      char c   ;
4      int  num ;
5
6      c   = 'a' ;
7      num =  c  ;
8
9      printf( "%c\n" , c );
10     printf( "%d\n" , c );
11
12     return 0 ;
13 }
```

Since "*a character is a number!*", we may print out the value of "c"

```
Output:
a
97
```

Character type – char

- As a matter of fact, **a character is a number**.

```
1  int main( void )
2  {
3      char c   ;
4      int  num ;
5
6      c = 'a'  ;  // character literal (plain value)
7      c = 97   ;  // integer literal
8      c = 0x61 ;  // hexidecimal format: 6x16 + 1 -> 97
9      c = 0141 ;  // octal format: 1x64 + 4x8 + 1 -> 97
10
11     printf( "%c\n" , c );
12     printf( "%d\n" , c );
13     return 0 ;
14 }
```

These four statements are the same!!!

Hex: leading 0x
Oct: leading 0

Special characters shown on the ASCII table

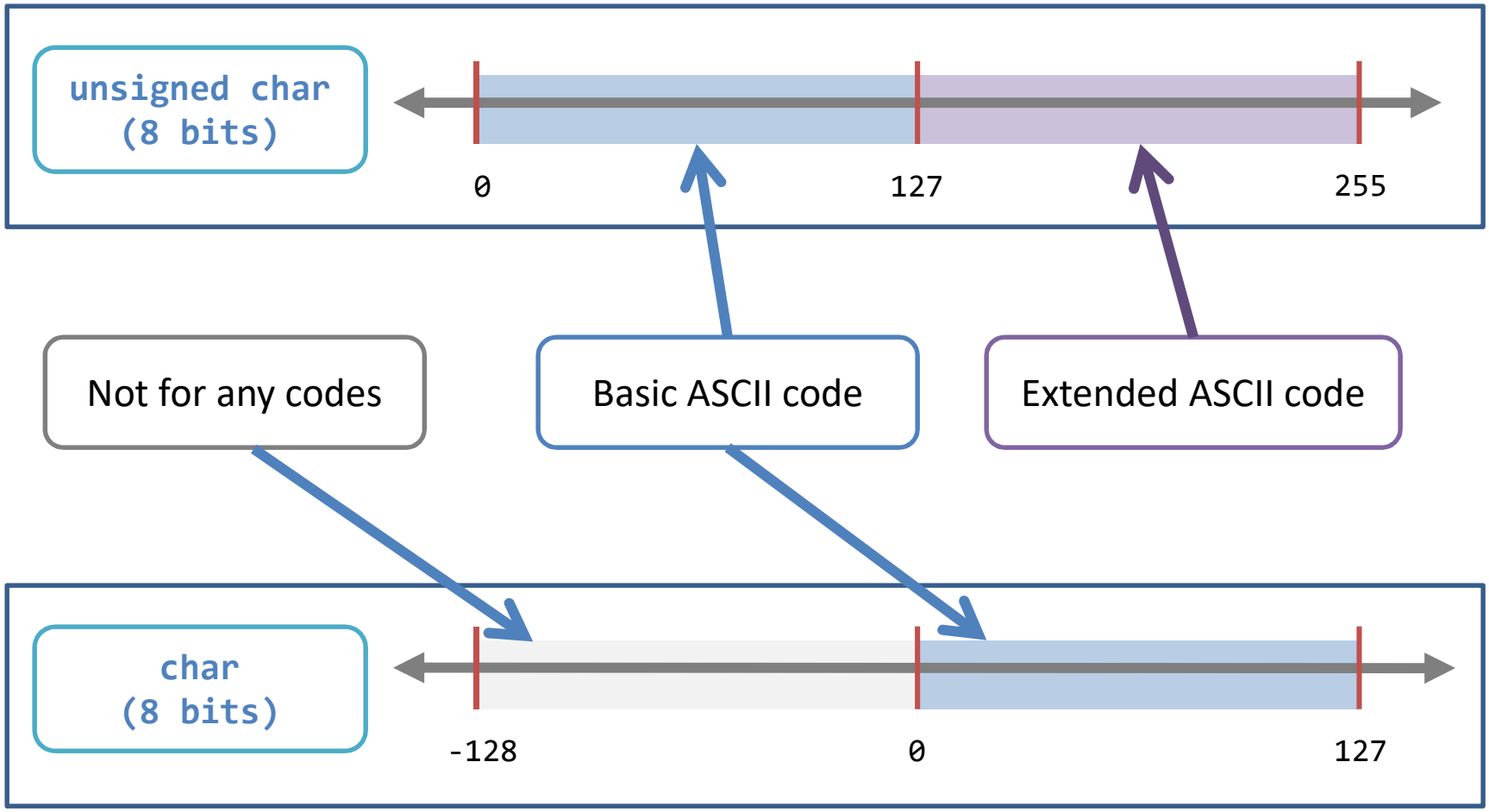
Symbol	ASCII	Description	Escaped characters in C
NUL	**0**	**null character**	`'\0'`
BEL	7	Bell (Cause a beep)	`'\a'`
BS	8	Backspace	`'\b'`
HT	**9**	**Horizontal Tab**	`'\t'`
LF	**10**	**Line feed (New line)**	`'\n'`
VT	11	Vertical Tab	`'\v'`
FF	12	Formfeed	`'\f'`
CR	13	Carriage Return	`'\r'`
ESC	27	Escape	
SP	**32**	**Space**	`' '`
DEL	127	Delete	

Character type – Encoding

Encoding Scheme	Byte for encoding a character	Description
ASCII **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange	1 byte	- ASCII code is a standard for representing basic characters in all computers, regardless of the programming languages - Basic ASCII code uses 7 bits, i.e., the most significant bit of the byte is 0 - Extended ASCII code uses 8 bits, i.e., the most significant bit of the byte is 1
BIG5	2 bytes	- BIG5 code represents (traditional) Chinese characters used in Taiwan and Hong Kong.

In addition, there are other encoding schemes such as Unicode, etc.
http://www.unicode.org/charts/

ASCII code


unsigned char
(8 bits)
0
127
255
Not for any codes
Basic ASCII code
Extended ASCII code
char
(8 bits)
-128
0
127

Chinese Encoding, e.g., BIG5

- How does the computer represent Chinese?
  - Encoded using **two consecutive bytes**
  - The first byte must be in [128, 255]
  - The second byte is in [0, 255]

A Big5 Example

```
int main( void )
{
    unsigned char byte1 = 0xA7 ;
    unsigned char byte2 = 0xDA ;
    printf( "%c%c\n" , byte1 , byte2 );
    return 0 ;
}
```

why using “unsigned”?

```
Output:
我
```

Note: you have to enable Chinese interface in your command line to see the character

Short Summary

- ASCII code is every where:
  - It is in C, in HTML, … etc.

- For more details about encoding, please visit the following great article:
  - http://www.joelonsoftware.com/articles/Unicode.html

Outline

- ~~What is a character?~~
- Character input and output
- Character operations
- What is a string?
- String operations

Character Processing – I/O

I/O: Input / Output

- Input – reading one character at a time

Function	Description
`char input = getchar() ;`	"**`getchar()`**" returns one character. Note that, later in the course, we will use "**`int`**" type to store the return value of "**`getchar()`**".
`char input ;` `scanf( "%c" , & input );`	**`scanf( "%c" , ... )`** when the pattern **`"%c"`** appears, **`scanf`** expects an input of size one byte (i.e., **`char`** type).

Character Processing – I/O

- Output – printing one character at a time

Function	Description
`char input = 'a' ;` `putchar( input );`	"**`putchar()`**" expects a single number in [0, 255] and prints it out.
`char input = 'a' ;` `printf( "%c\n" , input );`	**`printf( "%c" , ... )`** when the pattern **`"%c"`** appears, **`printf`** expects a single number in the range [0, 255].

Character Processing – I/O

- Simple I/O examples

```c
int main( void )
{
   putchar( 'a'  );
   putchar( '\n' );
   return 0 ;
}
```

```c
int main( void )
{
  printf( "%c\n" , 'a' );
  return 0 ;
}
```

  – '**\n**' means a single newline <u>character</u>
  – "**\n**" means a character <u>string</u> containing only one character, which is '**\n**'

Character Processing – I/O

- I/O examples with if-then-else:

```c
int main( void )
{
    char input = getchar() ;
    if ( input == 0 )
        printf( "It is zero\n" );
    else
        printf( "It is non-zero\n" );
    return 0 ;
}
```

  – What is the meaning of the above code?

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Character Processing – I/O

- I/O examples with if-then-else:

'0' and 0 are very different!

```c
int main( void )
{
    char input = getchar() ;
    if ( input == '0' )
        printf( "It is zero\n" );
    else
        printf( "It is non-zero\n" );
    return 0 ;
}
```

	0	1	2	3
0	NUL	SOH	STX	ET
1	DLE	DC1	DC2	DC
2		!	"	#
3	0	1	2	3
4	@	A	B	C
5	P	Q	R	S
6	`	a	b	c
7	p	q	r	s

– Since '`0`' (48 in ASCII) and `input` are numbers, then...

– We can compare them using relational operators.

  - Not only `==`, we may also use `>`, `>=`, `<`, `<=`, and `!=` later

Character Processing – I/O

- I/O examples with loops

```c
int main( void )
{
    char c ;
    while ( 1 )
    {
        c = getchar() ;
        putchar( c );
    }
    return 0 ;
}
```

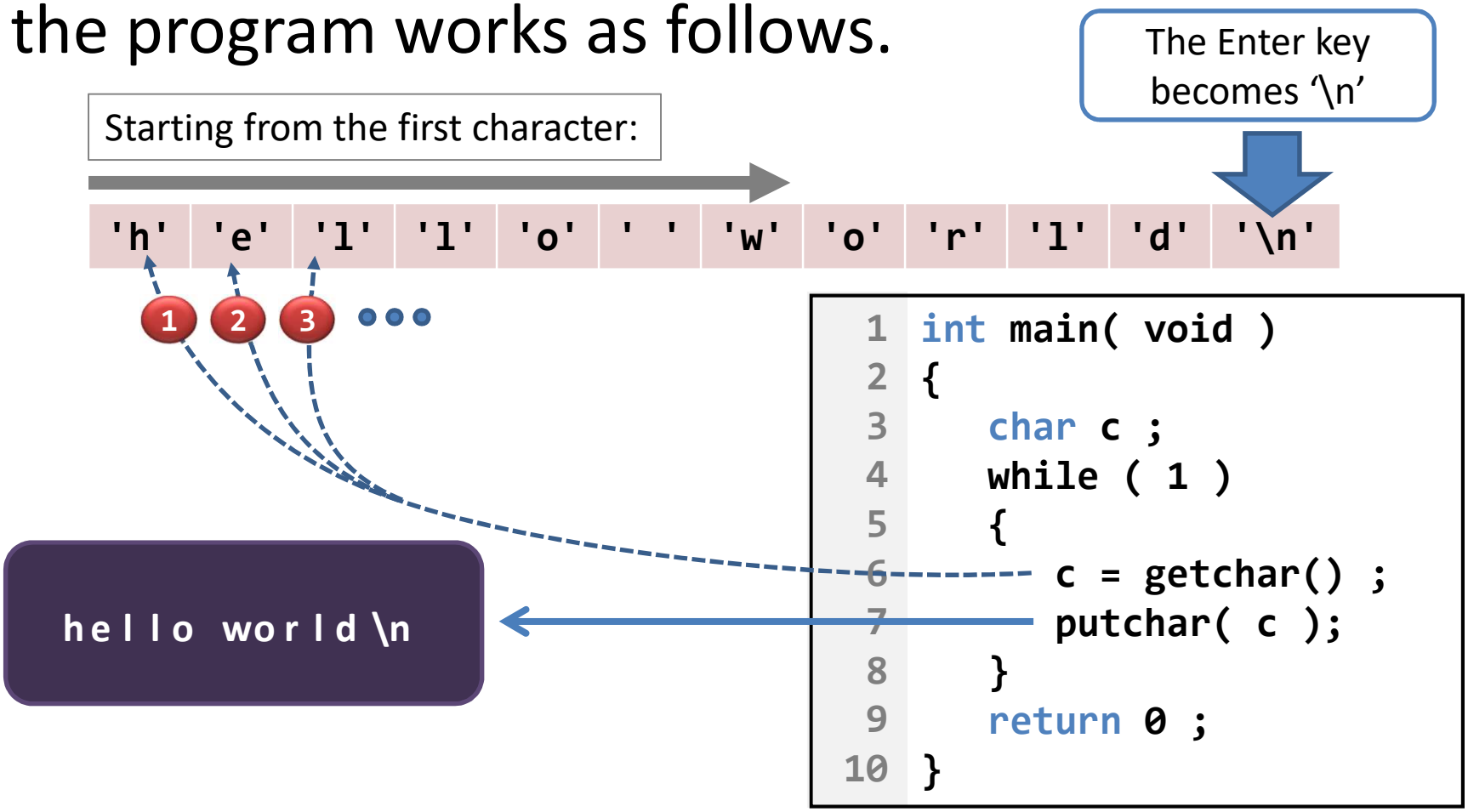
```c
int main( void )
{
    char c ;
    while ( 1 )
    {
        scanf(  "%c" , &c );
        printf( "%c" ,  c );
    }
    return 0 ;
}
```

  - What do the above programs do?
  - Any differences?
  - How to stop them from running?

Further reading: https://stackoverflow.com/questions/2507082/getc-vs-getchar-vs-scanf-for-reading-a-character-from-stdin (you may read it after you finished the whole lecture material)

Character Processing – I/O

- When you **type "hello world"** and **hit enter**, the program works as follows.




```
1  int main( void )
2  {
3      char c ;
4      while ( 1 )
5      {
6          c = getchar() ;
7          putchar( c );
8      }
9      return 0 ;
10 }
```

Character Processing – I/O

- But, how to stop this program?

```c
int main( void )
{
    char c ;
    while ( 1 )
    {
        c = getchar() ;
        putchar( c );
    }
    return 0 ;
}
```

```
hello world
hello world
## Right here waiting ##
```

Ctrl + C

Try a live demo.

Method #1: **Ctrl + C**

Ctrl + C is to kill (stop) the program immediately.

You can also **use Ctrl + C to stop any running program**, not just infinitely looping program.

Character Processing – I/O

- But, how to stop this program?
  - When **`getchar()`** meets the **end of the input**, **`getchar()`** returns **EOF**.

Program Updated

```
1  int main( void )
2  {
3     int c = getchar() ;
4     while ( c != EOF )
5     {
6        putchar( c );
7        c = getchar() ;
8     }
9     printf( "Bye\n" );
10    return 0 ;
11 }
```

Method #2: End the input.

Windows:
- In the beginning of a new line,
- Press **Ctrl + Z**, and release
- Press enter.

Mac & Linux:
- In the beginning of a new line,
- Press **Ctrl + D**.

Beware of End of line characters in your code: https://en.wikipedia.org/wiki/End-of-file

Character Processing – I/O

Key: white - keyboard input green - program output

```c
int main( void )
{
    int c = getchar() ;
    while ( c != EOF )
    {
        putchar( c );
        c = getchar() ;
    }
    printf( "Bye\n" );
    return 0 ;
}
```

Method 1

```
hello world
hello world
[Ctrl + C]
Program Terminated
```

In this example, Ctrl + C terminates the program while it is waiting for new input. As a matter of fact, **Ctrl + C can terminates a program at any time**.

Note: in Windows 10, “Bye” will also be printed before the program terminates…

Character Processing – I/O

Key: white - keyboard input green - program output

```
1  int main( void )
2  {
3     int c = getchar() ;
4     while ( c != EOF )
5     {
6        putchar( c );
7        c = getchar() ;
8     }
9     printf( "Bye\n" );
10    return 0 ;
11 }
```

c = EOF;

Method 2

```
hello world
hello world
[Ctrl + Z] [Enter]
Bye
Program Terminated
```

[Ctrl + Z, Enter] (in Windows) together stops the input. "**getchar()**" will know and return **EOF (End of File)**. The loop ends and "Bye" is printed.

Short Summary

- Characters are numbers!
  - Comparison can be done using relational operators.
- EOF is not equal to [Ctrl+Z, Enter]
  - It is just an **indicator** returned from **`getchar()`**;
  - EOF means End Of File (no data anymore); you will see its practice al usage when working taking text files as inputs.
- Ending input stream is used more often than Ctrl + C.
  - Ctrl+C is too violent, right?
  - Our online judge stops the input by ending the input stream!!!

Short Summary

- How `scanf( "%c" , & input )` reacts to the end of the input stream?
  - It behaves slightly different, since it does not output and write EOF to variable `input`.

- Instead, use the character value from `scanf`
  - For "%c", it always returns only **one character** value.
  - So, you may decide a special ending character.

Short Summary

- We can re-write the loop program as follows:

```
1 int main( void )
2 {
3    int c = 0;
4    while ( ( c = getchar() ) != EOF )
5      putchar( c );
6    printf( "Bye\n" );
7    return 0 ;
8 }
```

  – Any change in the behavior?
  – What is the purpose of the highlighted bracket pair?
  – What if that pair of brackets is removed?

See lecture notes 03a for the priority between != and =

Outline

- ~~What is a character?~~
- ~~Character input and output~~
- Character operations
- What is a string?
- String operations

Character operations

- Since characters are integers, we apply:
  - arithmetic operators over them, i.e., `+ - * / %`
  - relational operators over them , i.e., `> >= == !=` etc.

- Let's look at some well-known operations.

Character operations – Examples

- **Ex. #1: Character type: digits? Or alphabet?**
  - The design of ASCII code is amazing.

Digits	0	1	2	3	4	5	6	7	8	9
ASCII Code (dec)	48	49	50	51	52	53	54	55	56	57

Upper-case Character	A	B	C	D	E	...	W	X	Y	Z
ASCII Code (dec)	65	66	67	68	68	...	87	88	89	90

Lower-case Character	a	b	c	d	e	...	w	x	y	z
ASCII Code (dec)	97	98	99	100	101	...	119	120	121	122

Character operations – Examples

- **Ex. #1: Character type: digits? Or alphabet?**
  - The design of ASCII code is amazing.

Digits	0	1	2	3	4	5	6	7	8	9
ASCII Code (dec)	48	49	50	51	52	53	54	55	56	57

```c
int main( void )
{
    int c = getchar() ;
    if ( c >= 48 && c <= 57 )
        printf( "digit\n" );
    else
        printf( "Not a digit\n" );
    return 0 ;
}
```

We rely on the property that:
- The ASCII characters for digits are in the range [48-57], and
- there is no non-digit characters inside that range.

Yet, this example requires us to remember the ASCII codes...

Character operations – Examples

- **Ex. #1: Character type: digits? Or alphabet?**
  - The design of ASCII code is amazing.

Digits	0	1	2	3	4	5	6	7	8	9
ASCII Code (dec)	48	49	50	51	52	53	54	55	56	57

```c
int main( void )
{
    int c = getchar() ;
    if ( c >= 0 && c <= 9 )
        printf( "digit\n" );
    else
        printf( "Not a digit\n" );
    return 0 ;
}
```

'0' and 0 are very different!

Character operations – Examples

- **Ex. #1: Character type: digits? Or alphabet?**
  - The design of ASCII code is amazing.

Digits	0	1	2	3	4	5	6	7	8	9
ASCII Code (dec)	48	49	50	51	52	53	54	55	56	57

```c
int main( void )
{
    int c = getchar() ;
    if ( c >= '0' && c <= '9' )
        printf( "digit\n" );
    else
        printf( "Not a digit\n" );
    return 0 ;
}
```

Nice! This is better!
More **readable** than 48 & 57

So, can you extend this example into:

- Decide if a character is a alphabet?

Character operations – Examples

- **Ex. #2: lower-case → upper-case, and vice versa?**

Lower-case	a	b	...	z
ASCII Code: c	97	98	...	122
c - 'a'	0	1	...	25

Upper-case	A	B	...	Z
ASCII Code (dec)	65	66	...	90

+ 'A'

+ 'A'

```c
int main( void )
{
    int c = getchar() ;
    if ( c >= 'a' && c <= 'z' )
        c = c - 'a' + 'A' ;
    printf( "result = %c\n" , c );
    return 0 ;
}
```

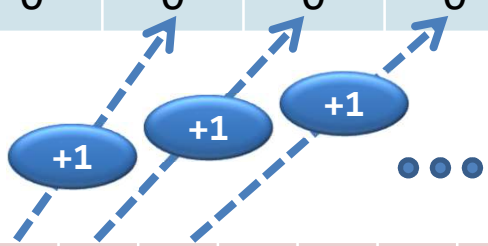
Great! Transformation is easy!

We rely on the property that:
- the ASCII alphabets are in continuous ranges
- but note [91,96] is not alphabets!!!

Character operations – Examples

- **Ex. #3: Counting digits**
  - **Goal**: for each digit **c**, how many **c** can be found from a stream of input characters?
  - **Idea**: initialize an array with zeros and sum up the frequency to array item accordingly

Digits	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
frequency	0	0	0	0	0	0	0	0	0	0



Input	1	2	3		3	2	1		1	2	3	4	5	6	7	\n

Character operations – Examples

- **Ex. #3: Counting digits**
  - **Code**: for ease of programming, it is good to transform: **character → array index**.

Digits	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
Array index	0	1	2	3	4	5	6	7	8	9
frequency	0	0	0	0	0	0	0	0	0	0

Character operations – Examples

- **Ex. #3: Counting digits**
  - By the way, we will have a related exercise in Lab 6

```
1  int main( void )
2  {
3     int c , i ;
4     int freq[ 10 ] = { 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 };
5
6     while ( ( c = getchar() ) != EOF )
7     {
8        if ( c >= '0' && c <= '9' )
9           freq[   ????   ]++ ;
10    }
11    /*  print the frequency array */
12    for ( i = 0 ; i < 10 ; i++ )
13       printf( "frequency of %c = %d\n" ,   ????   , freq[ i ] );
14    return 0 ;
15 }
```

Transformation:
character → array index

Character operations – Functions

- Remember to add "`#include <ctype.h>`"

Function name	Description (return 0 or 1)
`isascii`	If input integer is in the range [0,127], return `1 (true)`; Else, return `0 (false)`;
`isdigit`	If input integer is in the range ['0', '9'], return `1 (true)`; Else, return `0 (false)`;
`islower`	If input integer is in the range ['a', 'z'], return `1 (true)`; Else, return `0 (false)`;
`isupper`	If input integer is in the range ['A', 'Z'], return `1 (true)`; Else, return `0 (false)`;
`isspace`	If input integer is a whitespace, a tab, or a newline character, return `1 (true)`; Else, return `0 (false)`;

Character operations – Functions

- Using functions: Example

```
1  int main( void )
2  {
3     char input = getchar() ;
4     if ( isascii( input ) )
5     {
6        if ( isdigit( input ) )
7           printf( "Input is a digit\n" );
8        else
9        if ( isupper( input ) || islower( input ) )
10          printf( "Input is a character\n" );
11       else
12       if ( isspace( input ) )
13          printf( "Input is a space\n" );
14       else
15          printf( "I don't know what it is\n" );
16    }
17    else
18       printf( "Input is not an ASCII character\n" );
19    return 0 ;
20 }
```

Character operations – Functions

- Remember to add "`#include <ctype.h>`"

Function name	Description (return 0 or 1)
`isalpha`	Same as: `( islower( input ) || isupper( input ) )`

`isalnum`	Same as:
	`( islower( input ) || isupper( input )`
	`|| isdigit( input ) )`

`tolower`	If input integer is in the range ['A' , 'Z'], return `lower-case representation of the input;` Else, return the input
`toupper`	If input integer is in the range ['a' , 'z'], return `upper-case representation of the input;` Else, return the input

Outline

- ~~What is a character?~~
- ~~Character input and output~~
- ~~Character operations~~
- What is a string?
- String operations

Note: strings in C is simply a 1D array of char.
It is very different from high-level string data type available in C++ or in other programming lang.

What is a string?

- A string is **<u>an ordered list of characters</u>**.
- There are two methods in representing a string:
  - **Method 1**: string **literal** (or string **constant**):

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

A string constant begins and ends with a pair of double quotes.

Example #1. `"Hello world\n"` is a string with 12 characters.
- Note that '\n' forms one character, but in general.... (see footnote below)

Example #2. `""` is a string with no character, and we call it an **empty string**.

A literal is a plain value in a program.
Pay attention to '\n' in different operating system: https://en.wikipedia.org/wiki/Newline

What is a string?

Note: A string constant is **read-only**, cannot be modified.

– **Method 2**: a mutable string with a character array.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Array Content	'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'\n'	'\0'

- The last character `'\0'` is called the <u>null terminator</u>.
- It terminates a string (or means the end of the string).
- <u>**Every string must be ended with an `'\0'`!**</u> **Why?**

Because a string is an array of char!

String – Basics

What will happen if I do this after?

```
array[5] = '!';
```

- Initialization:

```
char array[ 10 ] = "Hello" ;
```

Index	0	1	2	3	4	5	6	7	8	9
array	'H'	'e'	'l'	'l'	'o'	'\0'	??	??	??	??

  - **Note 1**: you must allocate a big-enough array for a string.
  - **Note 2**: this copy operation is only valid when declaring a char array.
  - **Note 3**: in this example, the **<u>6th slot (i.e., array[5]) is automatically filled with '\0'</u>**, but not for the slots after it.

String – Basics

- Reading:

```
1  int main( void )
2  {
3     char array[ 20 ] = "Hello" ;
4     int  i ;
5
6     for ( i = 0 ; array[ i ] != '\0' ; i++ )
7        putchar( array[ i ] );
8
9     puts( array );
10
11    printf( "array = [%s]\n" , array );
12
13    return 0 ;
14 }
```

This should be the first time you find a new way to write a **for-loop condition**

puts() prints a string. It automatically appends a newline character ('\n') after the string output

printf() prints a string with the format **"%s"**

String – Basics

- Updating:

```
1  int main( void )
2  {
3     char array[ 20 ] = "Hello" ;
4     int  i ;
5
6     for ( i = 0 ; array[ i ] != '\0' ; i++ )
7        array[i] = toupper( array[ i ] );
8
9     puts( array );
10
11    printf( "array = [%s]\n" , array );
12
13    return 0 ;
14 }
```

Since a string is an array, you can modify individual character

String – Basics

- Discussion: what will happen?

```c
int main( void )
{
    char array[ 20 ] = "Hello" ;

    array[ 0 ] = '\0' ;

    puts( array );

    printf( "array = [%s]\n" , array );

    return 0 ;
}
```

Important note

- Two troublesome (or tedious) things about C-String, which is just a “1D array of char”
  1. Make sure you **allocate enough memory** to C-String, such that it can hold all necessary data, including '\0‘
  2. Whenever you modify the contents of a C-String, make sure you **append '\0' after the last data byte**

```
char array[ 7 ] = "Hello" ;
array[ 5 ] = '!' ;
array[ 6 ] = '\0' ;  // important!
printf( "array = [%s]\n" , array );
```

Index	0	1	2	3	4	5	6
Array Content	'H'	'e'	'l'	'l'	'o'	'\0'	?

Outline

- ~~What is a character?~~
- ~~Character input and output~~
- ~~Character operations~~
- ~~What is a string?~~
- String operations

String operations – Basics

- Remember to add "`#include <string.h>`"

Functions	Descriptions
`strlen`	Pronunciation: string length **Return an integer**: the number of characters before the first null terminator
`strcmp`	Pronunciation: string compare **Return 0**: if the two input strings are the same **Return non-zero**: otherwise
`strcpy`	Pronunciation: string copy It copies a string from one to another in a character-by-character manner, **up to and including the first null terminator**.

String operations – Basics

- Example 1

```
1  int main( void )
2  {
3     char str1[ 20 ] = "hello" , str2[ 20 ] ;
4
5     printf( “[%s] has %d characters\n" ,
6             str1                       ,
7             strlen( str1 )             );
8
9     strcpy( str2 , str1 );
10
11    if ( strcmp( str1 , str2 ) == 0 )
12       printf( "Same\n" );
13    else
14       printf( "Different\n" );
15    return 0 ;
16 }
```

Length is integer.
Use **"%d"** in **printf**.

Format for **strcpy**:
strcpy(
destination,
source
);

Check if the two strings are identical

String operations – Basics

- Example 2

```
1  int main( void )
2  {
3     char str1[ 20 ] = "hello" , str2[ 20 ] ;
4
5     strcpy( str2 , str1 );
6     str2[4] = '\0' ;
7
8     if ( strcmp( str1 , str2 ) == 0 )
9        printf( "Same\n" );
10    else
11       printf( "Different\n" );
12    return 0 ;
13 }
```

What is the output?

String operations – Basics

- Example 3

```c
int main( void )
{
   char str1[20] , str2[20] ;
   int  result ;

   strcpy( str1 , "hello" );
   strcpy( str2 , "hEllo" );
   result = strcmp( str1 , str2 );

   if ( result > 0 )
      printf( "ASCII value of first unmatched char of str1 > str2" );
   else
   if ( result < 0 )
      printf( "ASCII value of first unmatched char of str1 < str2" );
   else
      printf( "str1 and str2 are the same" ) ;
   return 0 ;
}
```

What is the output?

String operation – I/O

- Using **`fgets( ... )`**:
  - It **reads and stores** from the input stream character by character **until and include the newline symbol**.
  - After **storing the newline character to the string**, it will append a null terminator **`'\0'`** at the end.

"fgets" Example 1

```
1  int main( void )
2  {
3     char str[ 20 ];
4     fgets( str , 20 , stdin );
5     printf( "Output = [%s]\n" ,  str );
6     return 0 ;
7  }
```

Input to **`fgets(...)`**:

1) target character array;
2) size of the character array;
3) "**`stdin`**" is called the "**standard input stream**". That means **`fgets`** will read from the keyboard.

String operation – I/O

- Using **`fgets( ... )`**:
  - Perfect for **<u>reading input line by line</u>**
  - Returns a special value (**NULL**) when encountering the end of the stream

"fgets" Example 2

```
int main( void )
{
   char str[ 100 ] ;
   int  i = 1      ;
   while ( fgets( str , 100 , stdin ) != NULL )
   {
      printf( "Line %d: [%s]" , i , str );
      i++ ;
   }
   return 0 ;
}
```

String operation – I/O

- Using **`fgets( ... )`**:
  - Perfect for **reading input line by line**
  - Returns a special value (**NULL**) when encountering the end of the stream

"fgets" Example 2.1

```
1  int main( void )
2  {
3     char str[ 100 ] ;
4     int  i = 1      ;
5     while ( fgets( str , sizeof(str) , stdin ) != NULL )
6     {
7        printf( "Line %d: [%s]" , i , str );
8        i++ ;
9     }
10    return 0 ;
11 }
```

String operation – I/O

- Using **`scanf( "%s" , ... )`**:
  - It **reads and stores** from the input stream character by character until, but **<u>not including</u>**:

    (1) space, (2) tab, and (3) newline characters.

"scanf reading strings" Example 1

```c
int main( void )
{
    char str[ 20 ];
    scanf( "%s" , str );
    printf( "Output = [%s]\n" , str );
    return 0 ;
}
```

Note to **`scanf( "%s" )`**:

1) "**%s**" is for string.
2) No need '**&**' before variable '**str**'

String operation – I/O

- Using **`scanf( "%s" , ... )`**:
  - Perfect for **<u>reading input word by word</u>**
  - Returns 0 when encountering the end of the stream

"scanf reading strings" Example 2

```
1  int main( void )
2  {
3     char str[ 100 ];
4     int  i  = 1 ;
5     while ( scanf( "%s" , str ) > 0 )
6     {
7        printf( "Word %d: [%s]\n" , i , str );
8        i++ ;
9     }
10    return 0 ;
11 }
```

A common mistake…

Note: CF = 13 and LF = 10

- Newline symbol in different operating systems: https://en.wikipedia.org/wiki/Newline
- Different platforms (for some historical reasons)

Operating system	Character encoding	Abbreviation	hex value	dec value	Escape sequence
Multics, Unix and Unix-like systems (Linux, macOS, FreeBSD, AIX, Xenix, etc.), BeOS, Amiga, RISC OS, and others[2]	ASCII	LF	0A	10	\n
Atari TOS, Microsoft Windows, DOS (MS-DOS, PC DOS, etc.), DEC TOPS-10, RT-11, CP/M, MP/M, OS/2, Symbian OS, Palm OS, Amstrad CPC, and most other early non-Unix and non-IBM operating systems		CR LF	0D 0A	13 10	\r\n
Commodore 8-bit machines (C64, C128), Acorn BBC, ZX Spectrum, TRS-80, Apple II family, Oberon, the classic Mac OS, MIT Lisp Machine and OS-9		CR	0D	13	\r

<u>Beware of these in lab. Ex.:</u>

(1) Need extra two bytes for **'\0' & '\n'**; // fgets -> example on P.46 becomes wrong

(2) Don't read char by char to bypass \n; better to use **fgets to read line by line** or **scanf to read word by word**; and

(3) Don't just check (ch == 13 OR ch == 10) to decide if it is a newline symbol

A common mistake…

- Hence, if you read char by char, and try to read one or two char to skip the '\n', what may happen?
  - If there are two chars, you will miss one!

- Note: you should use **`fgets & scanf:`**
  - **`fgets`** **reads and stores** from the input stream the whole line **<u>until & include the "newline" character</u>**
  - **`scanf`** **reads and stores** from the input stream the whole word **<u>until & exclude the "spaces"</u>** <u>(space, tab, and newline)</u>

Summary

- We introduced a new data type '**`char`**' (just a single byte integer) for storing characters.

- A string in C is an array of characters.
  - On top of that, we learn a new character '**`\0`**', for marking the end of a string data in the array; make sure enough space allocated for the string contents and '**`\0`**' and possibly the new line symbol (which can be 2 bytes)

- We enriched the set of formatted strings for both **printf** and **scanf**.

Summary

- Reading from input is always more complex than printing the output
  - It is because an input stream may end.

- We know how to read characters :
  - **Character by character**: `getchar()`, `scanf( "%c" );`
  - **Word by word**: `scanf( "%s" );`
  - **Line by line**: `fgets( ... );`

Summary

- The following four pages are cheat sheets for inputs and outputs:
  - input functions;
  - output functions;
  - **`scanf(...)`** format strings; and
  - **`printf(...)`** format strings.

- Further reading:
  - conio API for DOS and Windows

    https://en.wikipedia.org/wiki/Conio.h
  - curses library API for Linux and Unix

    https://en.wikipedia.org/wiki/Curses_(programming_library)

Cheat sheet: Input functions

Name	Purpose & Example
`getchar`	**Purpose**: Read **one character** from the input stream **Return EOF** when encountering the end of the input `char input = getchar() ;`
`scanf`	**Purpose:** Read **one word** (a list of characters) and process them **Return 0** when encountering the end of the input See the `scanf` check sheet.

`fgets`	**Purpose:** Read **one line** from the input
	Note: Read and store the ending newline character
	Return NULL when encountering the end of the input
	`char string[ 100 ];`
	`fgets( string , 100 , stdin );`

Cheat sheet: Output functions

Name	Purpose & Example
putchar	**Purpose:** Print one character `putchar( 'a' );`
printf	**Purpose:** Print and format the output See the **printf** check sheet

puts	**Purpose:** Print one string
	Notes: Add a newline character after printing the string
	`char string[ 100 ] = "hello\n" ;`
	`puts( string );  // this produces two lines.`

Cheat sheet – printf

Format string		Examples
%c	char	`char c = 'a' ; printf( "%c\n" , c );`
%d	integer	`int i = 10 ; printf( "%d\n" , i );`
%u	unsigned int	`unsigned int i = 10 ; printf( "%u\n" , i );`
%ld	long	`long i = 10 ; printf( "%ld\n" , i );`
%lu	unsigned long	`unsigned long i = 10 ; printf( "%lu\n" , i );`
%lld	long long	`long long i = 10 ; printf( "%lld\n" , i );`
%llu	unsigned long long	`unsigned long long i = 10 ; printf( "%llu\n" , i );`
%f	float, double	`float num = 10.0 ; printf( "%f\n" , num );`
%s	string	`char s[10] = "hello" ; printf( "%s\n" , s );`

Cheat sheet – scanf

Format string		Examples
%c	char	`char c ; scanf( "%c" , &c );`
%d	integer	`int i ; scanf( "%d" , &i );`
%u	unsigned int	`unsigned int i ; scanf( "%u" , &i );`
%ld	long	`long i ; scanf( "%ld" , &i );`
%lu	unsigned long	`unsigned long i ; scanf( "%lu" , &i );`
%lld	long long	`long long i ; scanf( "%lld" , &i );`
%llu	unsigned long long	`unsigned long long i ; scanf( "%llu" , &i );`
%f	float	`float num ; scanf( "%f" , &num );`
%lf	double	`double num ; scanf( "%lf" , &num );`
%s	string	`char s[ 10 ] ;` `scanf( "%s" , s );  /* no '&' */`