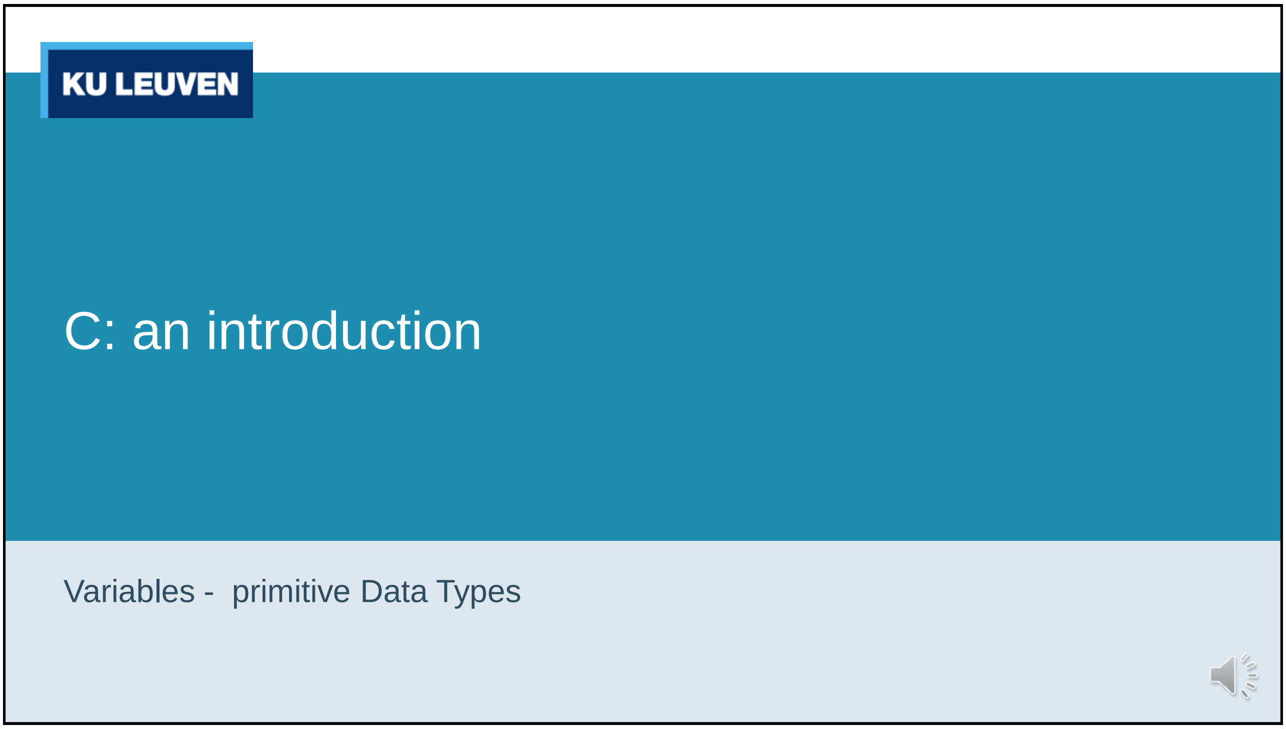
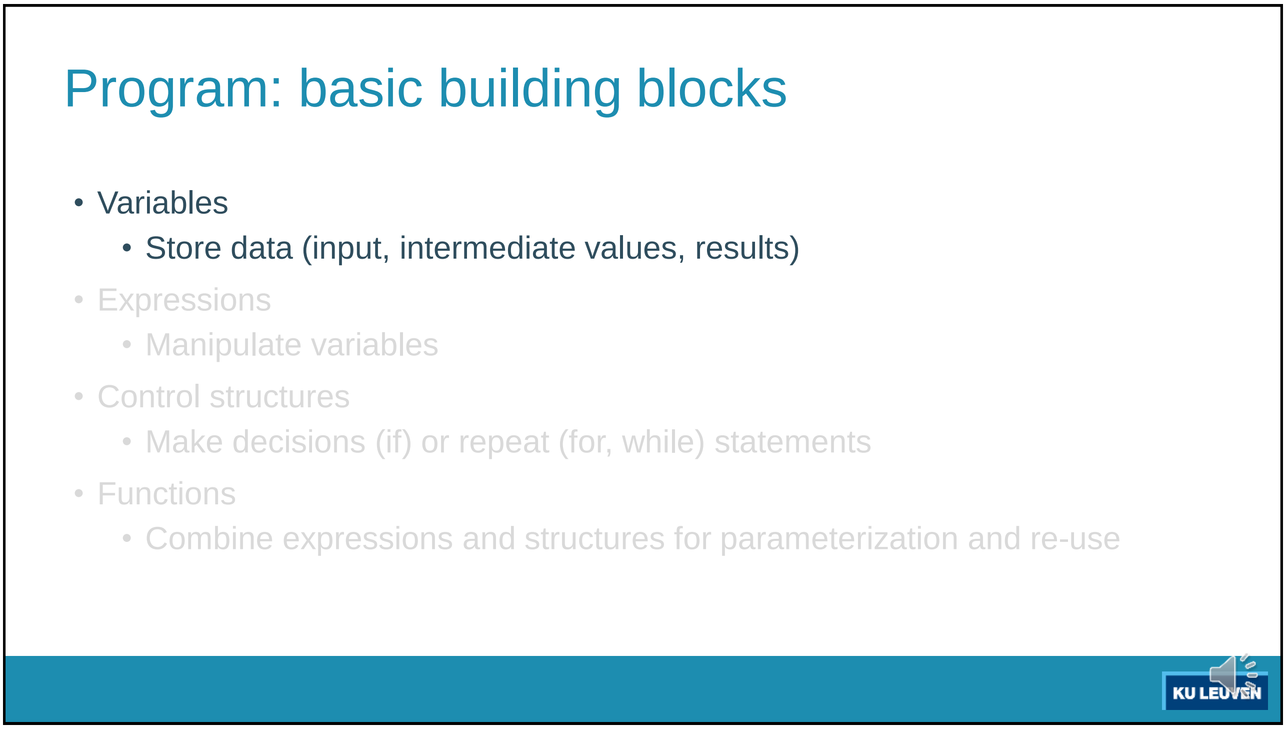


KU LEUVEN

C: an introduction

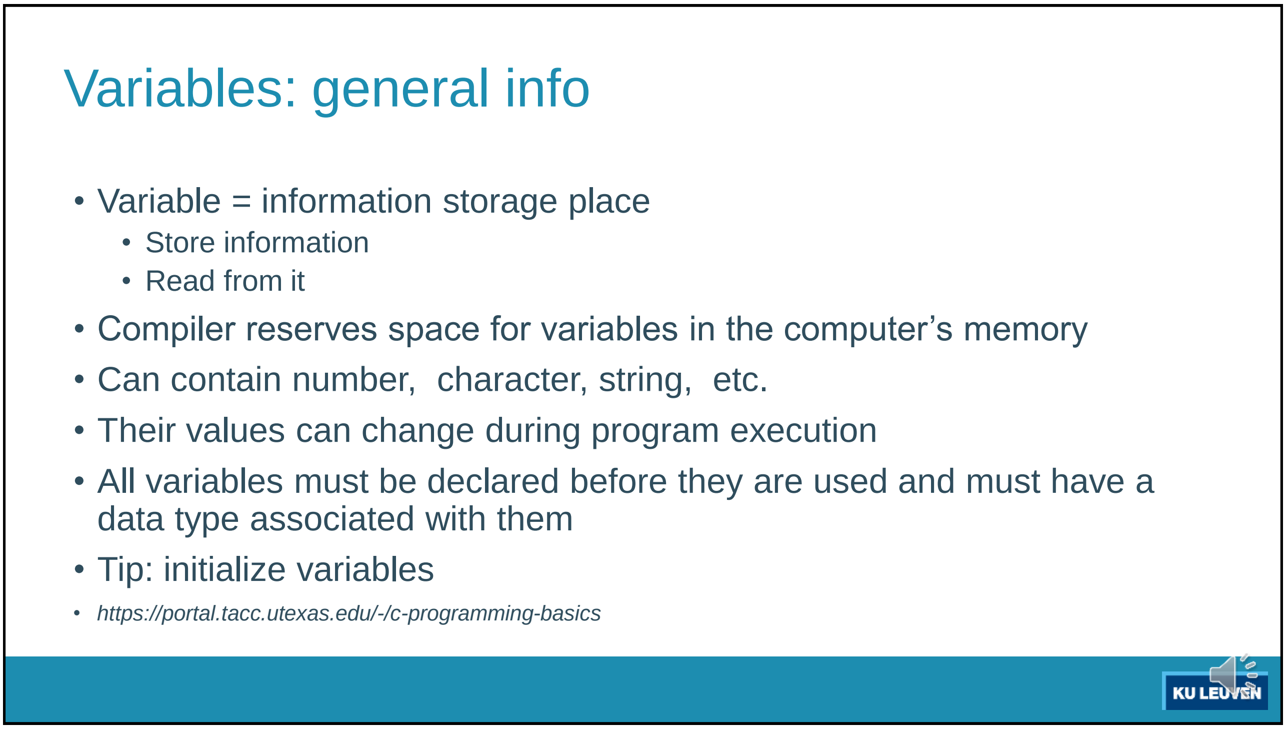
Variables - primitive Data Types



Program: basic building blocks

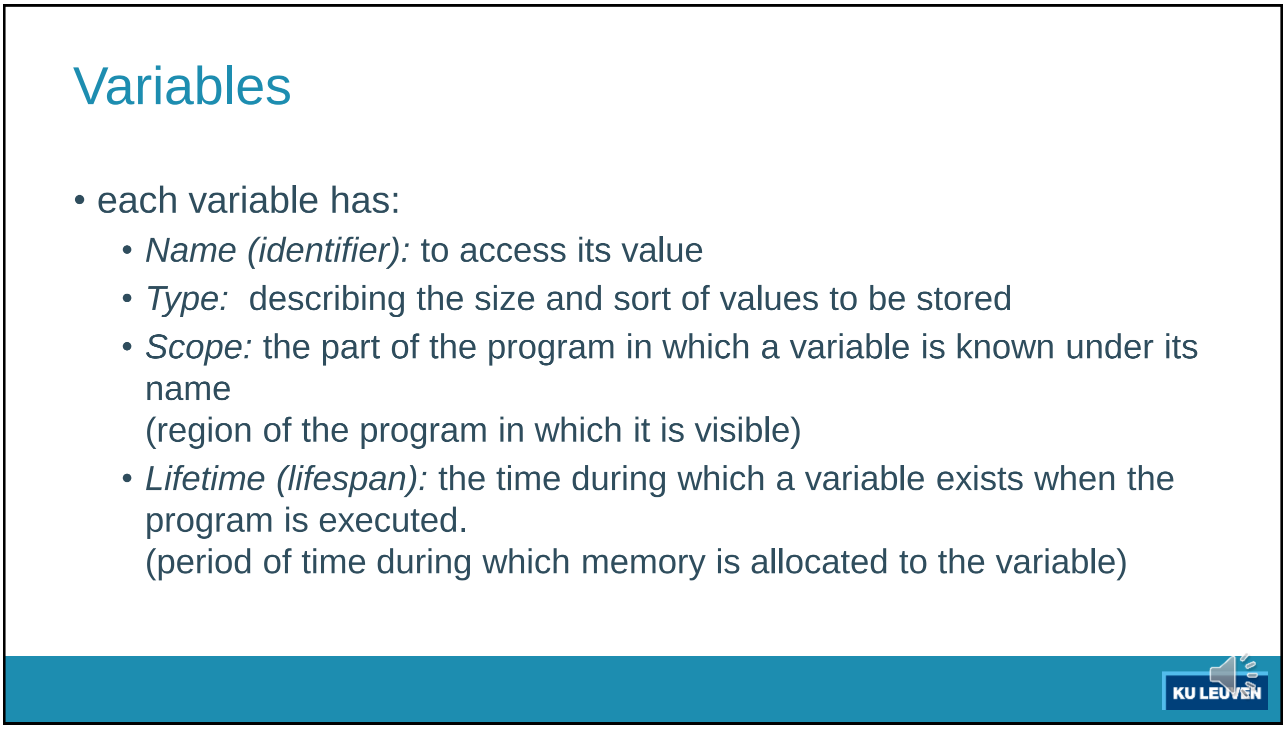
- Variables
  - Store data (input, intermediate values, results)
- Expressions
  - Manipulate variables
- Control structures
  - Make decisions (if) or repeat (for, while) statements
- Functions
  - Combine expressions and structures for parameterization and re-use

KU LEUVEN



Variables: general info

- Variable = information storage place
  - Store information
  - Read from it
- Compiler reserves space for variables in the computer's memory
- Can contain number, character, string, etc.
- Their values can change during program execution
- All variables must be declared before they are used and must have a data type associated with them
- Tip: initialize variables
- *https://portal.tacc.utexas.edu/-/c-programming-basics*



Variables

- each variable has:
  - *Name (identifier):* to access its value
  - *Type:* describing the size and sort of values to be stored
  - *Scope:* the part of the program in which a variable is known under its name
    (region of the program in which it is visible)
  - *Lifetime (lifespan):* the time during which a variable exists when the program is executed.
    (period of time during which memory is allocated to the variable)

Variable Names

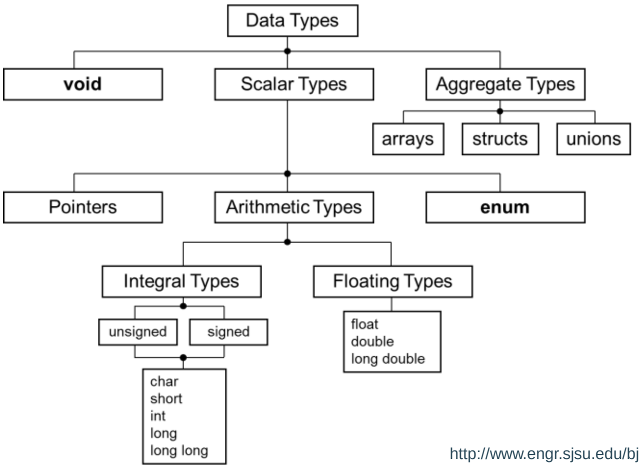
- Names are composed of alphanumeric characters (letters and numbers), and _ (underscore)
  - Names may not start with a number
  - Names are case sensitive; `st` and `St` are different variables
  - can have at least 31 characters
  - do not use keywords (reserved words)
- Convention:
  - variable names begin with a lower-case letter,
  - symbolic constant: capital letters
  - Try to use meaningful names

```
 step, top, i, j, x, fahr, flag
lookup_table_index, nameListHead
```

The C Type System

- C is a “typed” language.
  - explicitly define variables with a specific type.
- By specifying types, the compiler can perform a number of tasks:
  - Size: Compiler allocates fixed amount of memory for a given type.
  - Usage: A variable’s type determines what value it can represent and what operations may be performed on it.
  - Compiler can detect type-mismatch errors.

Data types



Data Types

void

Scalar Types

Aggregate Types

arrays

structs

unions

Pointers

Arithmetic Types

enum

Integral Types

Floating Types

unsigned

signed

char
short
int
long
long long

float
double
long double

http://www.engr.sjsu.edu/bjfurman/courses/ME30/lecture_notes.htm

C primitive data types

Type		Size(bytes)	Size(bits)
`char`	a single byte, capable of holding one character	1	8
`int`	an integer	4	32
`float`	single-precision floating point number	4	32
`double`	double-precision floating point number	8	64

Qualifier	
`short`	
`long`	
`signed`	type may represent a negative number
`unsigned`	cannot represent a negative number

C primitive data types

- Basic data types
  1. Integer types: `char`, `int`
  2. Floating point types: `float`, `double`
- Different types have different properties:
  - Values they can represent
    These types have *finite* precision and range.
    - There is a limit to the size of a number (min, max)
    - Floating point values have a limit to the number of significant figures.
  - Operations that can be performed on them.

Variable declaration

- format
  `type varname1, varname2, ...;`
- ex:

```
int  count;
float aa;
double percent, total;
unsigned char x,y,z;
long int aLongInt;
```

- each variable must be declared before use
- Tip: put some comments at declaration time

Variable declaration

- initialise a variable before using it:
  - skipping initialisation can lead to problems
  - C standard does not mention the value of non-initialised variables

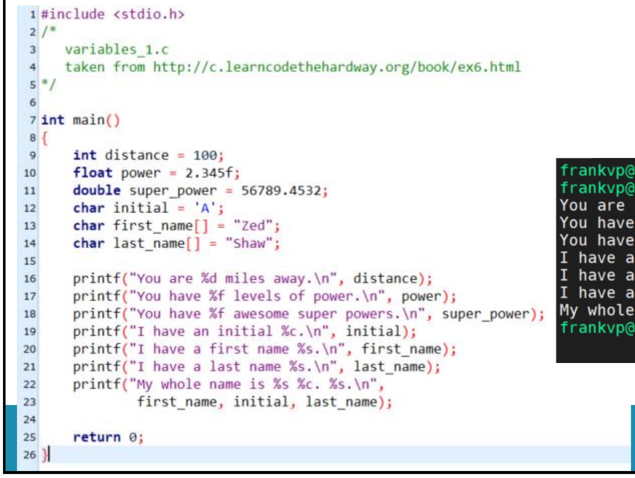
```
int counter;
counter=0;
```

or

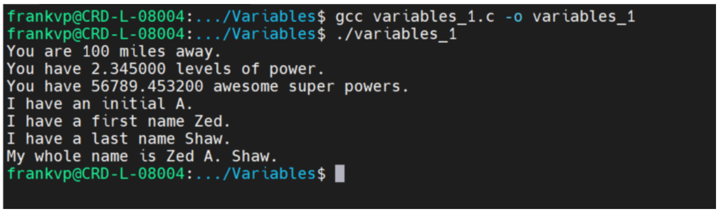
```
int counter=0;
```

- *File: variables_1.c*
- *File: demo_init.c*

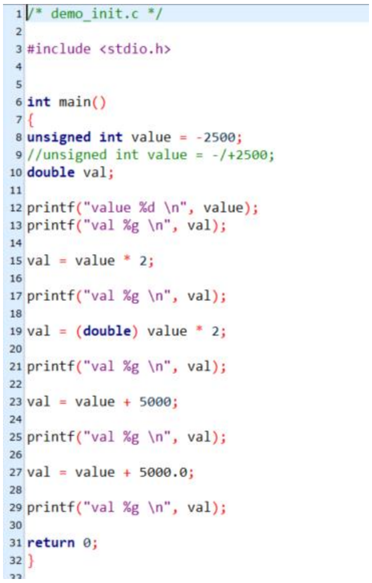
- variables_1.c



```c
#include <stdio.h>
/*
   variables_1.c
   taken from http://c.learncodethehardway.org/book/ex6.html
*/

int main()
{
    int distance = 100;
    float power = 2.345f;
    double super_power = 56789.4532;
    char initial = 'A';
    char first_name[] = "Zed";
    char last_name[] = "Shaw";

    printf("You are %d miles away.\n", distance);
    printf("You have %f levels of power.\n", power);
    printf("You have %f awesome super powers.\n", super_power);
    printf("I have an initial %c.\n", initial);
    printf("I have a first name %s.\n", first_name);
    printf("I have a last name %s.\n", last_name);
    printf("My whole name is %s %c. %s.\n",
            first_name, initial, last_name);

    return 0;
}
```

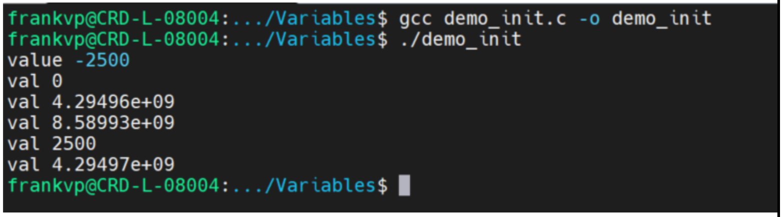


```
frankvp@CRD-L-08004:.../Variables$ gcc variables_1.c -o variables_1
frankvp@CRD-L-08004:.../Variables$ ./variables_1
You are 100 miles away.
You have 2.345000 levels of power.
You have 56789.453200 awesome super powers.
I have an initial A.
I have a first name Zed.
I have a last name Shaw.
My whole name is Zed A. Shaw.
frankvp@CRD-L-08004:.../Variables$ 
```

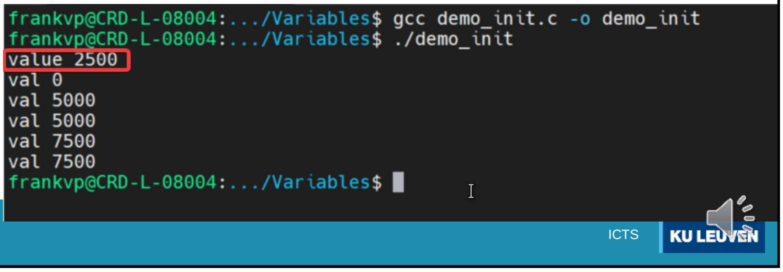


```
/* demo_init.c */

#include <stdio.h>


int main()
{
unsigned int value = -2500;
//unsigned int value = -/+2500;
double val;

printf("value %d \n", value);
printf("val %g \n", val);

val = value * 2;

printf("val %g \n", val);

val = (double) value * 2;

printf("val %g \n", val);

val = value + 5000;

printf("val %g \n", val);

val = value + 5000.0;

printf("val %g \n", val);

return 0;
}
```

- *demo_init.c*



```
frankvp@CRD-L-08004:.../Variables$ gcc demo_init.c -o demo_init
frankvp@CRD-L-08004:.../Variables$ ./demo_init
value -2500
val 0
val 4.29496e+09
val 8.58993e+09
val 2500
val 4.29497e+09
frankvp@CRD-L-08004:.../Variables$
```



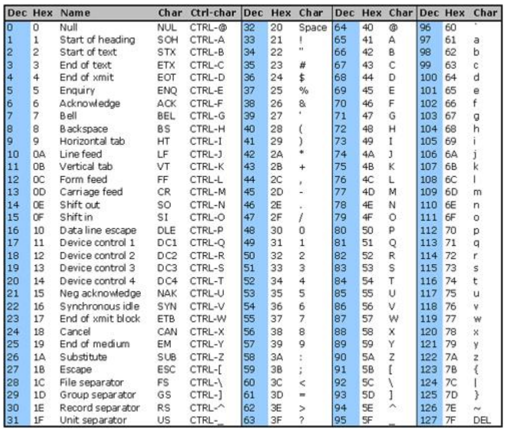
```
frankvp@CRD-L-08004:.../Variables$ gcc demo_init.c -o demo_init
frankvp@CRD-L-08004:.../Variables$ ./demo_init
value 2500
val 0
val 5000
val 5000
val 7500
val 7500
frankvp@CRD-L-08004:.../Variables$
```

void

- `void` type means no value.
- Usually used to specify the type of functions which returns nothing.

character

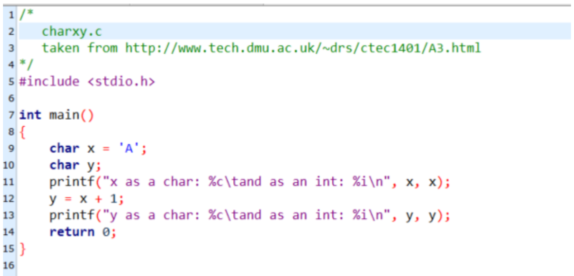
- Commonly used to represent ASCII characters
- To represent characters, C uses a lookup table
    - Each character has a unique integer code
    - ASCII table is most common code
- Eg., Letter *R* can be represented by
    - Its ASCII integer code: `82` or `0122` or `0x52`
    - Or, use character constant: `'R'`



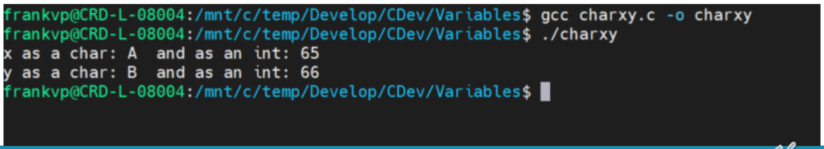
Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D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

character: examples

charxy.c



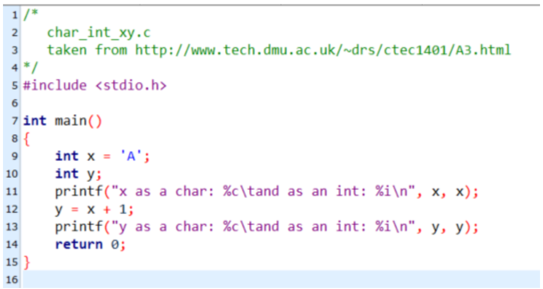
```c
/*
   charxy.c
   taken from http://www.tech.dmu.ac.uk/~drs/ctec1401/A3.html
*/
#include <stdio.h>

int main()
{
    char x = 'A';
    char y;
    printf("x as a char: %c\tand as an int: %i\n", x, x);
    y = x + 1;
    printf("y as a char: %c\tand as an int: %i\n", y, y);
    return 0;
}

```



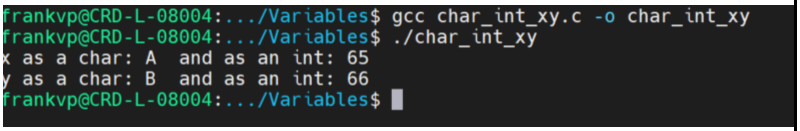
```
frankvp@CRD-L-08004:/mnt/c/temp/Develop/CDev/Variables$ gcc charxy.c -o charxy
frankvp@CRD-L-08004:/mnt/c/temp/Develop/CDev/Variables$ ./charxy
x as a char: A  and as an int: 65
y as a char: B  and as an int: 66
frankvp@CRD-L-08004:/mnt/c/temp/Develop/CDev/Variables$ █
```

character: examples

char_int_xy.c



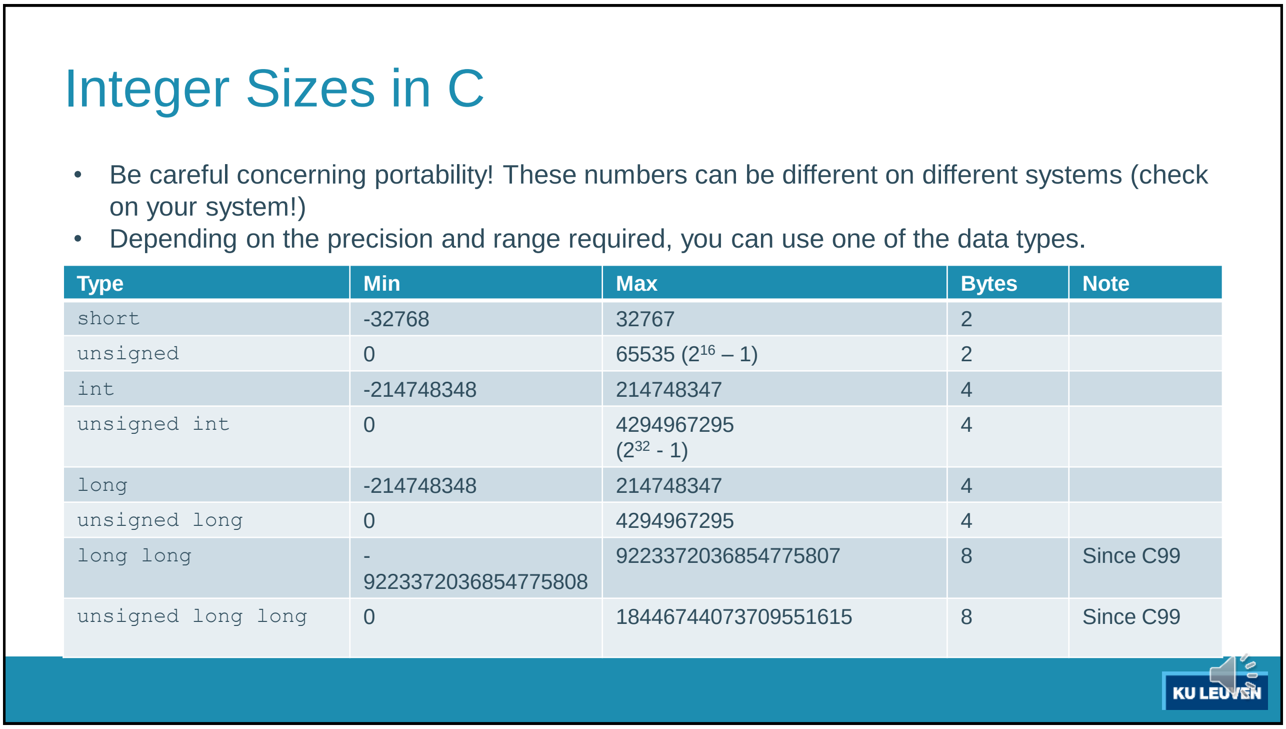
```c
/*
   char_int_xy.c
   taken from http://www.tech.dmu.ac.uk/~drs/ctec1401/A3.html
*/
#include <stdio.h>

int main()
{
    int x = 'A';
    int y;
    printf("x as a char: %c\tand as an int: %i\n", x, x);
    y = x + 1;
    printf("y as a char: %c\tand as an int: %i\n", y, y);
    return 0;
}

```



```
frankvp@CRD-L-08004:.../Variables$ gcc char_int_xy.c -o char_int_xy
frankvp@CRD-L-08004:.../Variables$ ./char_int_xy
x as a char: A  and as an int: 65
y as a char: B  and as an int: 66
frankvp@CRD-L-08004:.../Variables$ 
```

Integer Sizes in C

- Integers are a type that can only hold one of a finite range of whole number values.
- The standard does not specify exact sizes. The amount of memory storage allocated for a particular type will be different on different systems.
- ISO C standard requires *only* that
  - `short` is at least 16 bits
    - Often used to conserve memory
  - `long` is at least 32 bits
  - `short <= int <= long`
- Range of permissible values of integers is defined in `limits.h`

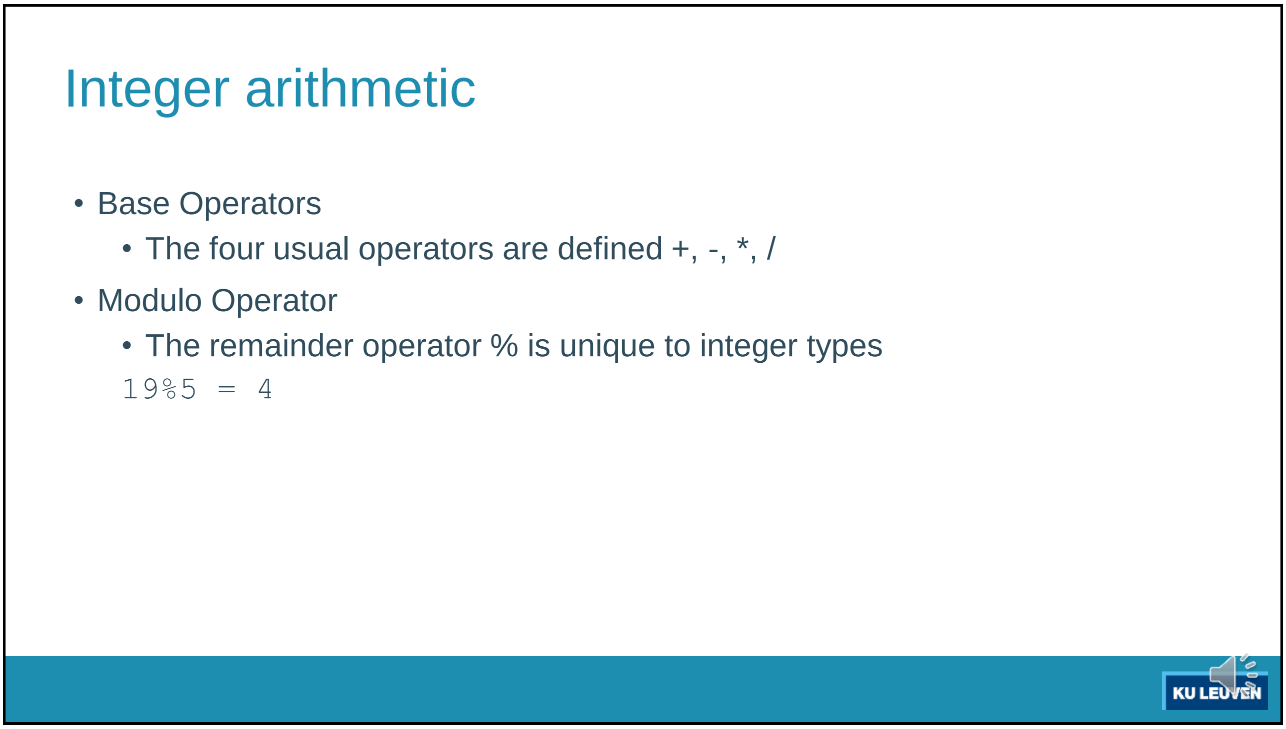


Integer Sizes in C

- Be careful concerning portability! These numbers can be different on different systems (check on your system!)
- Depending on the precision and range required, you can use one of the data types.

Type	Min	Max	Bytes	Note
`short`	-32768	32767	2	
`unsigned`	0	65535 (2^{16} – 1)	2	
`int`	-214748348	214748347	4	
`unsigned int`	0	4294967295 (2^{32} - 1)	4	
`long`	-214748348	214748347	4	
`unsigned long`	0	4294967295	4	
`long long`	-9223372036854775808	9223372036854775807	8	Since C99
`unsigned long long`	0	18446744073709551615	8	Since C99

KU LEUVEN



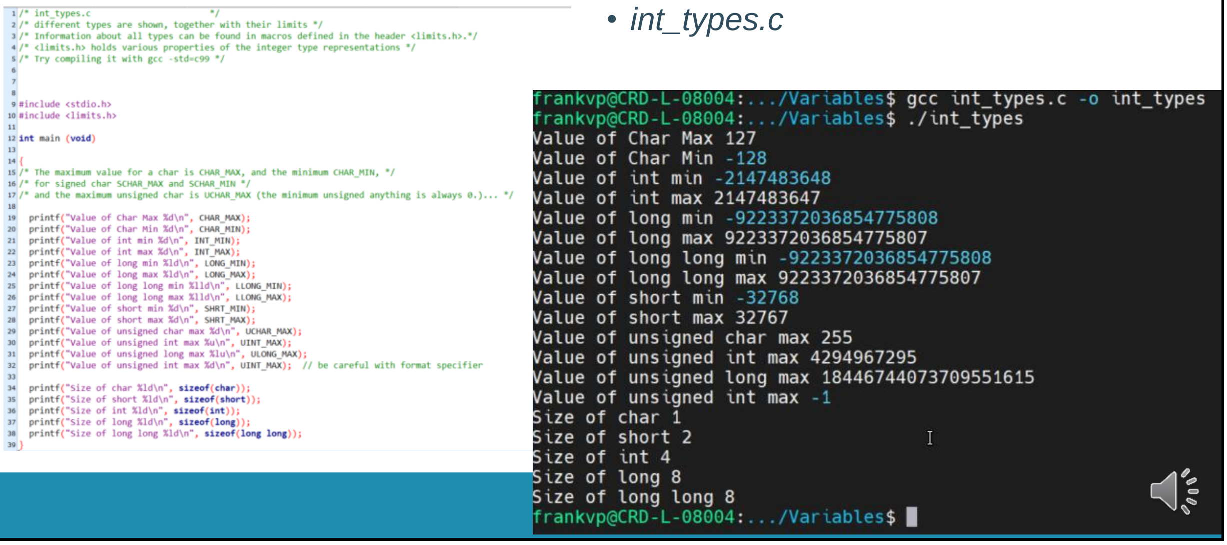
Integer arithmetic

- Base Operators
  - The four usual operators are defined +, -, *, /
- Modulo Operator
  - The remainder operator % is unique to integer types

```
19%5 = 4
```

KU LEUVEN

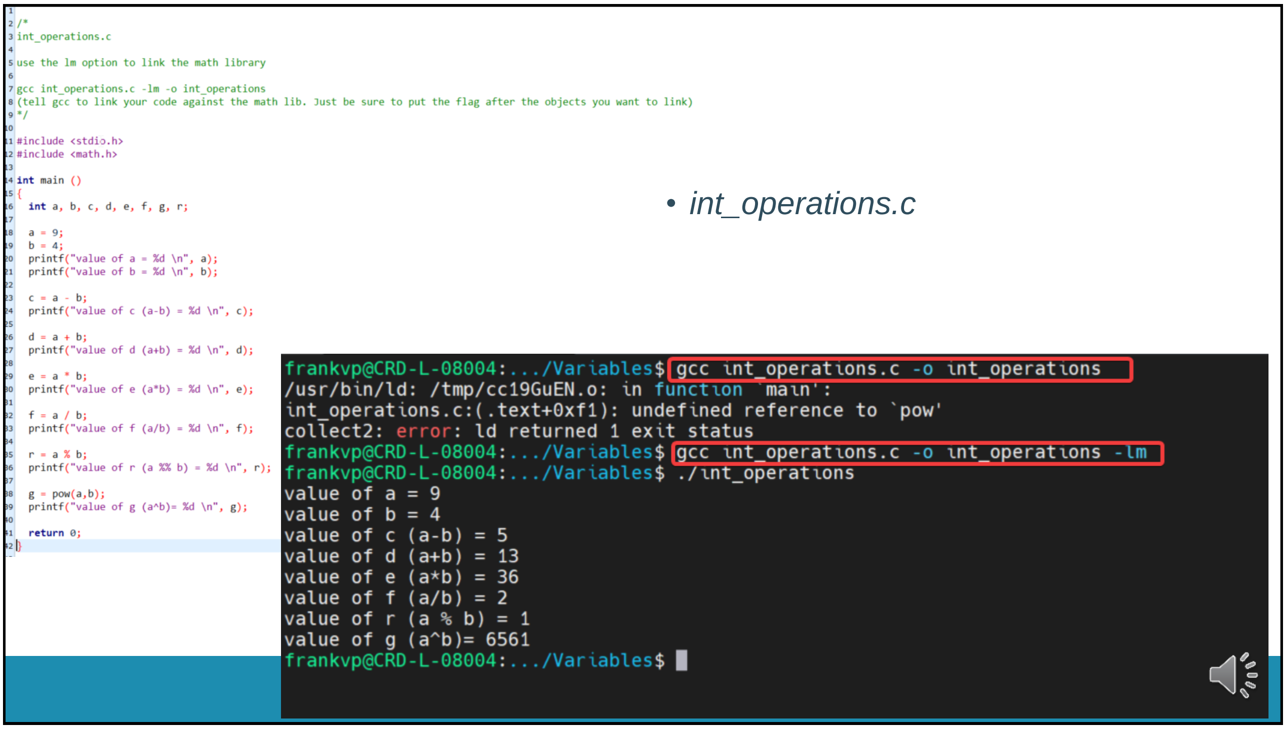
- *int_types.c*



```c
/* int_types.c                        */
/* different types are shown, together with their limits */
/* Information about all types can be found in macros defined in the header <limits.h>.*/
/* <limits.h> holds various properties of the integer type representations */
/* Try compiling it with gcc -std=c99 */



#include <stdio.h>
#include <limits.h>

int main (void)

{
/* The maximum value for a char is CHAR_MAX, and the minimum CHAR_MIN, */
/* for signed char SCHAR_MAX and SCHAR_MIN */
/* and the maximum unsigned char is UCHAR_MAX (the minimum unsigned anything is always 0.)... */

  printf("Value of Char Max %d\n", CHAR_MAX);
  printf("Value of Char Min %d\n", CHAR_MIN);
  printf("Value of int min %d\n", INT_MIN);
  printf("Value of int max %d\n", INT_MAX);
  printf("Value of long min %ld\n", LONG_MIN);
  printf("Value of long max %ld\n", LONG_MAX);
  printf("Value of long long min %lld\n", LLONG_MIN);
  printf("Value of long long max %lld\n", LLONG_MAX);
  printf("Value of short min %d\n", SHRT_MIN);
  printf("Value of short max %d\n", SHRT_MAX);
  printf("Value of unsigned char max %d\n", UCHAR_MAX);
  printf("Value of unsigned int max %u\n", UINT_MAX);
  printf("Value of unsigned long max %lu\n", ULONG_MAX);
  printf("Value of unsigned int max %d\n", UINT_MAX);  // be careful with format specifier

  printf("Size of char %ld\n", sizeof(char));
  printf("Size of short %ld\n", sizeof(short));
  printf("Size of int %ld\n", sizeof(int));
  printf("Size of long %ld\n", sizeof(long));
  printf("Size of long long %ld\n", sizeof(long long));
}
```

```
frankvp@CRD-L-08004:.../Variables$ gcc int_types.c -o int_types
frankvp@CRD-L-08004:.../Variables$ ./int_types
Value of Char Max 127
Value of Char Min -128
Value of int min -2147483648
Value of int max 2147483647
Value of long min -9223372036854775808
Value of long max 9223372036854775807
Value of long long min -9223372036854775808
Value of long long max 9223372036854775807
Value of short min -32768
Value of short max 32767
Value of unsigned char max 255
Value of unsigned int max 4294967295
Value of unsigned long max 18446744073709551615
Value of unsigned int max -1
Size of char 1
Size of short 2
Size of int 4
Size of long 8
Size of long long 8
frankvp@CRD-L-08004:.../Variables$ 
```

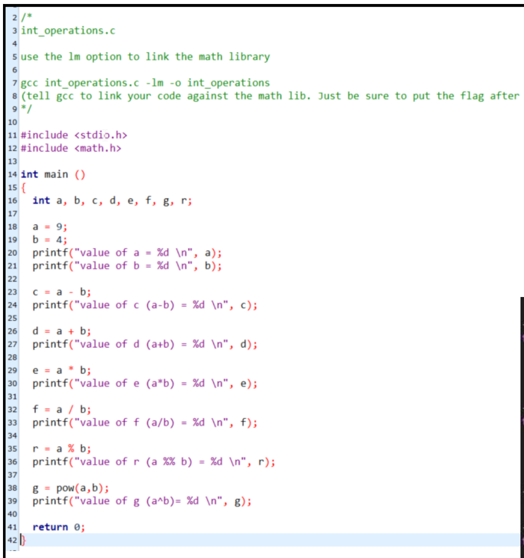
- *int_operations.c*



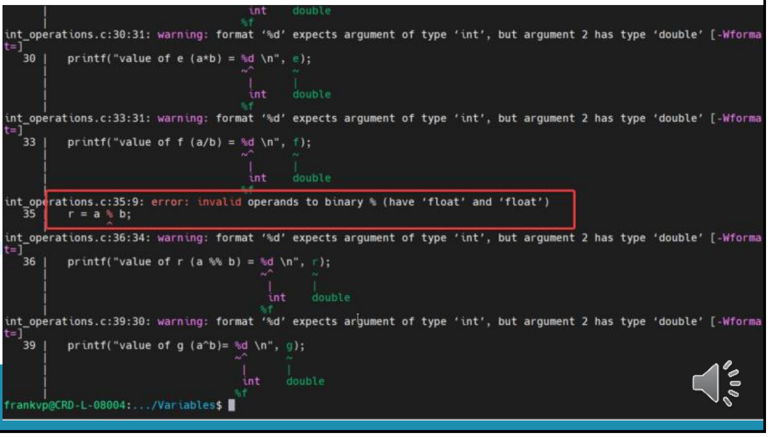
```c

/*
int_operations.c

use the lm option to link the math library

gcc int_operations.c -lm -o int_operations
(tell gcc to link your code against the math lib. Just be sure to put the flag after the objects you want to link)
*/

#include <stdio.h>
#include <math.h>

int main ()
{
  int a, b, c, d, e, f, g, r;

  a = 9;
  b = 4;
  printf("value of a = %d \n", a);
  printf("value of b = %d \n", b);

  c = a - b;
  printf("value of c (a-b) = %d \n", c);

  d = a + b;
  printf("value of d (a+b) = %d \n", d);

  e = a * b;
  printf("value of e (a*b) = %d \n", e);

  f = a / b;
  printf("value of f (a/b) = %d \n", f);

  r = a % b;
  printf("value of r (a %% b) = %d \n", r);

  g = pow(a,b);
  printf("value of g (a^b)= %d \n", g);

  return 0;
}
```

```
frankvp@CRD-L-08004:.../Variables$ gcc int_operations.c -o int_operations
/usr/bin/ld: /tmp/cc19GuEN.o: in function `main':
int_operations.c:(.text+0xf1): undefined reference to `pow'
collect2: error: ld returned 1 exit status
frankvp@CRD-L-08004:.../Variables$ gcc int_operations.c -o int_operations -lm
frankvp@CRD-L-08004:.../Variables$ ./int_operations
value of a = 9
value of b = 4
value of c (a-b) = 5
value of d (a+b) = 13
value of e (a*b) = 36
value of f (a/b) = 2
value of r (a % b) = 1
value of g (a^b)= 6561
frankvp@CRD-L-08004:.../Variables$ 
```

- *int_operations.c*
- *Change `int` into `float`*

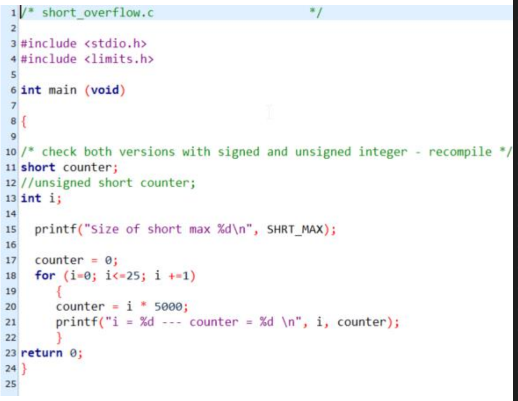


```c
/*
int_operations.c

use the lm option to link the math library

gcc int_operations.c -lm -o int_operations
(tell gcc to link your code against the math lib. Just be sure to put the flag after the objects you want to link)
*/

#include <stdio.h>
#include <math.h>

int main ()
{
  int a, b, c, d, e, f, g, r;

  a = 9;
  b = 4;
  printf("value of a = %d \n", a);
  printf("value of b = %d \n", b);

  c = a - b;
  printf("value of c (a-b) = %d \n", c);

  d = a + b;
  printf("value of d (a+b) = %d \n", d);

  e = a * b;
  printf("value of e (a*b) = %d \n", e);

  f = a / b;
  printf("value of f (a/b) = %d \n", f);

  r = a % b;
  printf("value of r (a %% b) = %d \n", r);

  g = pow(a,b);
  printf("value of g (a^b)= %d \n", g);

  return 0;
}
```

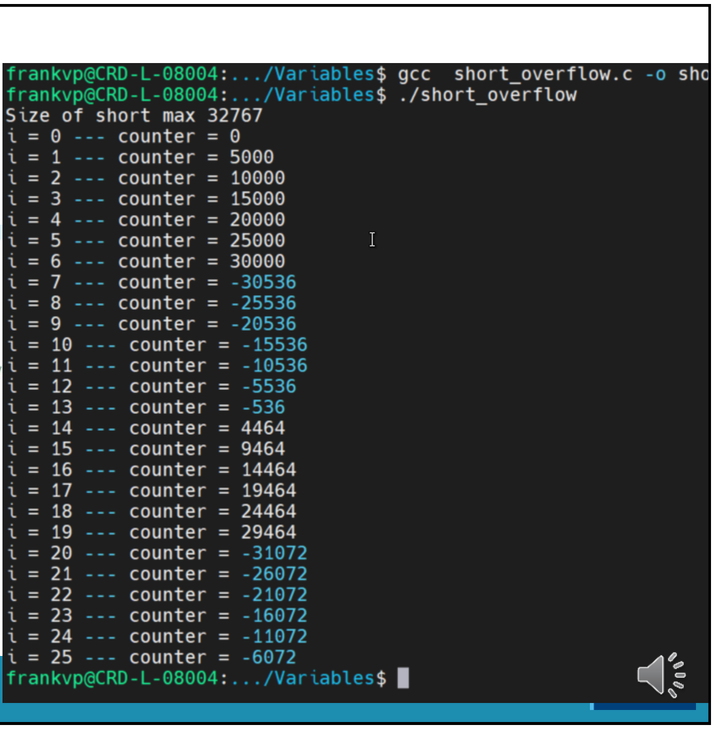


```
      |                          int    double
      |                         %f
int_operations.c:30:31: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘double’ [-Wforma
t=]
   30 |   printf("value of e (a*b) = %d \n", e);
      |                              ~^      ~
      |                               |      |
      |                               int    double
      |                              %f
int_operations.c:33:31: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘double’ [-Wforma
t=]
   33 |   printf("value of f (a/b) = %d \n", f);
      |                              ~^      ~
      |                               |      |
      |                               int    double
      |                              %f
int_operations.c:35:9: error: invalid operands to binary % (have ‘float’ and ‘float’)
   35 |   r = a % b;
      |         ^
int_operations.c:36:34: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘double’ [-Wforma
t=]
   36 |   printf("value of r (a %% b) = %d \n", r);
      |                                 ~^      ~
      |                                  |      |
      |                                  int    double
      |                                 %f
int_operations.c:39:30: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘double’ [-Wforma
t=]
   39 |   printf("value of g (a^b)= %d \n", g);
      |                             ~^      ~
      |                              |      |
      |                              int    double
      |                             %f
frankvp@CRD-L-08004:.../Variables$ 
```

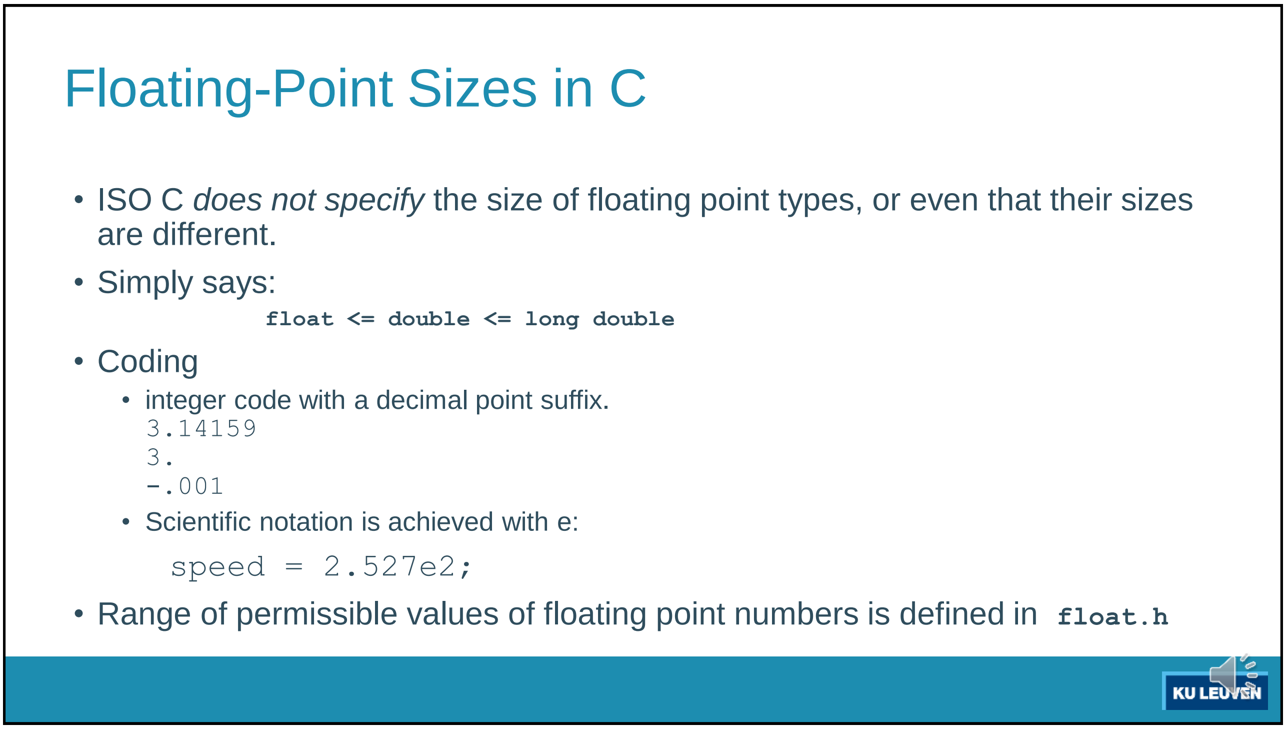
- *short_overflow.c*



```c
/* short_overflow.c                   */

#include <stdio.h>
#include <limits.h>

int main (void)

{

/* check both versions with signed and unsigned integer - recompile */
short counter;
//unsigned short counter;
int i;

  printf("Size of short max %d\n", SHRT_MAX);

  counter = 0;
  for (i=0; i<=25; i +=1)
     {
     counter = i * 5000;
     printf("i = %d --- counter = %d \n", i, counter);
     }
return 0;
}
```



```
frankvp@CRD-L-08004:.../Variables$ gcc  short_overflow.c -o sho
frankvp@CRD-L-08004:.../Variables$ ./short_overflow
Size of short max 32767
i = 0 --- counter = 0
i = 1 --- counter = 5000
i = 2 --- counter = 10000
i = 3 --- counter = 15000
i = 4 --- counter = 20000
i = 5 --- counter = 25000
i = 6 --- counter = 30000
i = 7 --- counter = -30536
i = 8 --- counter = -25536
i = 9 --- counter = -20536
i = 10 --- counter = -15536
i = 11 --- counter = -10536
i = 12 --- counter = -5536
i = 13 --- counter = -536
i = 14 --- counter = 4464
i = 15 --- counter = 9464
i = 16 --- counter = 14464
i = 17 --- counter = 19464
i = 18 --- counter = 24464
i = 19 --- counter = 29464
i = 20 --- counter = -31072
i = 21 --- counter = -26072
i = 22 --- counter = -21072
i = 23 --- counter = -16072
i = 24 --- counter = -11072
i = 25 --- counter = -6072
frankvp@CRD-L-08004:.../Variables$ 
```



Floating-Point Sizes in C

- ISO C *does not specify* the size of floating point types, or even that their sizes are different.
- Simply says:

```
float <= double <= long double
```

- Coding
  - integer code with a decimal point suffix.

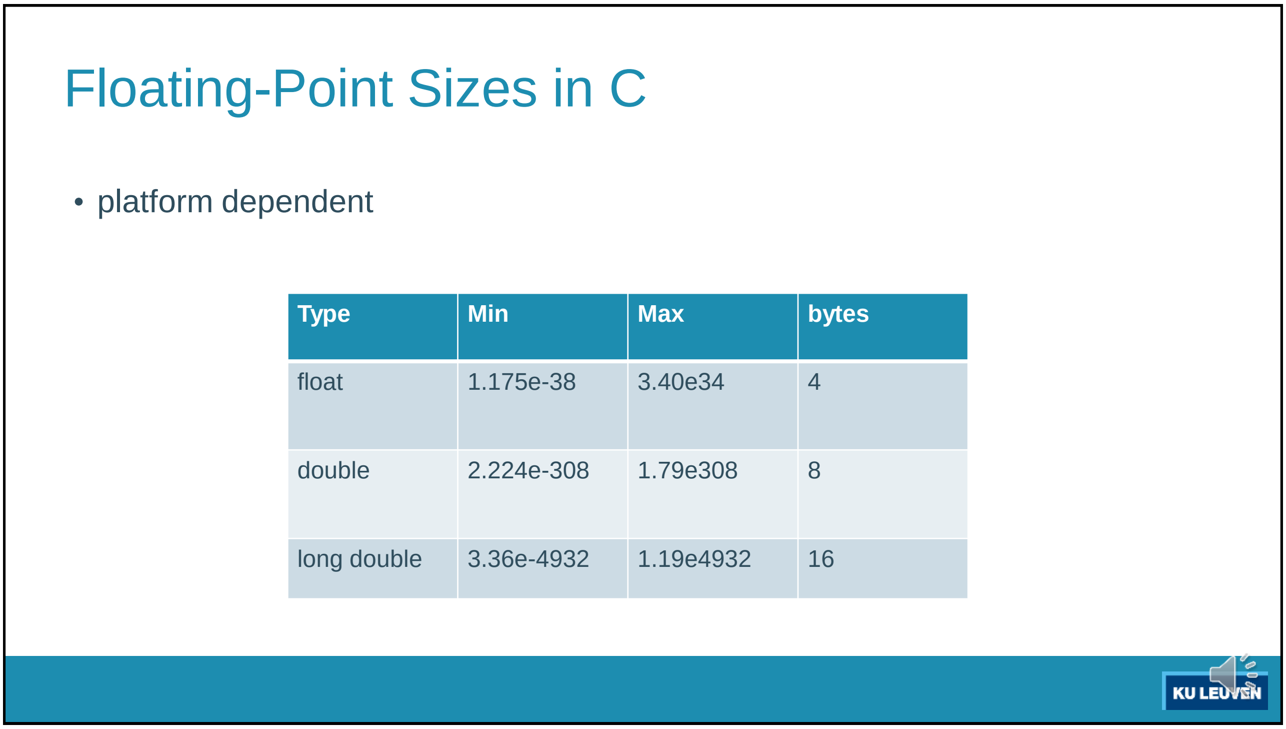
```
3.14159
3.
-.001
```

  - Scientific notation is achieved with e:

```
speed = 2.527e2;
```

- Range of permissible values of floating point numbers is defined in **`float.h`**

KU LEUVEN

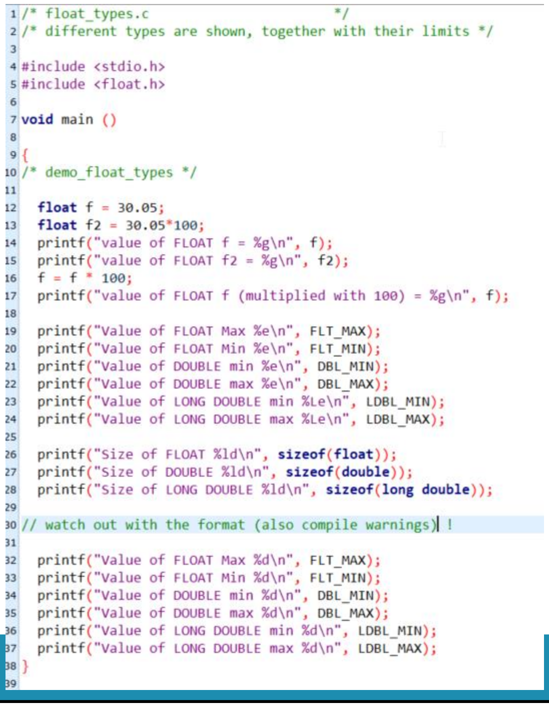


Floating-Point Sizes in C

- platform dependent

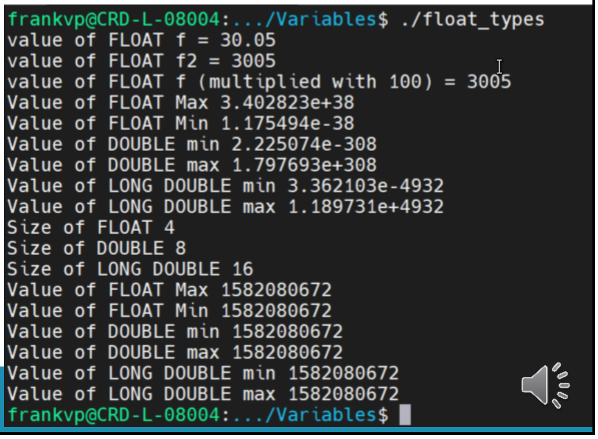
Type	Min	Max	bytes
float	1.175e-38	3.40e34	4
double	2.224e-308	1.79e308	8
long double	3.36e-4932	1.19e4932	16

KU LEUVEN



```c
/* float_types.c                           */
/* different types are shown, together with their limits */

#include <stdio.h>
#include <float.h>

void main ()

{
/* demo_float_types */

  float f = 30.05;
  float f2 = 30.05*100;
  printf("value of FLOAT f = %g\n", f);
  printf("value of FLOAT f2 = %g\n", f2);
  f = f * 100;
  printf("value of FLOAT f (multiplied with 100) = %g\n", f);

  printf("Value of FLOAT Max %e\n", FLT_MAX);
  printf("Value of FLOAT Min %e\n", FLT_MIN);
  printf("Value of DOUBLE min %e\n", DBL_MIN);
  printf("Value of DOUBLE max %e\n", DBL_MAX);
  printf("Value of LONG DOUBLE min %Le\n", LDBL_MIN);
  printf("Value of LONG DOUBLE max %Le\n", LDBL_MAX);

  printf("Size of FLOAT %ld\n", sizeof(float));
  printf("Size of DOUBLE %ld\n", sizeof(double));
  printf("Size of LONG DOUBLE %ld\n", sizeof(long double));

// watch out with the format (also compile warnings) !

  printf("Value of FLOAT Max %d\n", FLT_MAX);
  printf("Value of FLOAT Min %d\n", FLT_MIN);
  printf("Value of DOUBLE min %d\n", DBL_MIN);
  printf("Value of DOUBLE max %d\n", DBL_MAX);
  printf("Value of LONG DOUBLE min %d\n", LDBL_MIN);
  printf("Value of LONG DOUBLE max %d\n", LDBL_MAX);
}
```

- *File: float_types.c*



```
frankvp@CRD-L-08004:.../Variables$ ./float_types
value of FLOAT f = 30.05
value of FLOAT f2 = 3005
value of FLOAT f (multiplied with 100) = 3005
Value of FLOAT Max 3.402823e+38
Value of FLOAT Min 1.175494e-38
Value of DOUBLE min 2.225074e-308
Value of DOUBLE max 1.797693e+308
Value of LONG DOUBLE min 3.362103e-4932
Value of LONG DOUBLE max 1.189731e+4932
Size of FLOAT 4
Size of DOUBLE 8
Size of LONG DOUBLE 16
Value of FLOAT Max 1582080672
Value of FLOAT Min 1582080672
Value of DOUBLE min 1582080672
Value of DOUBLE max 1582080672
Value of LONG DOUBLE min 1582080672
Value of LONG DOUBLE max 1582080672
frankvp@CRD-L-08004:.../Variables$
```

Float arithmetic

- Base Operations: +, -, *, /
- Math functions come with the Standard C Library, which contains some mathematical functions.
- Use it with:

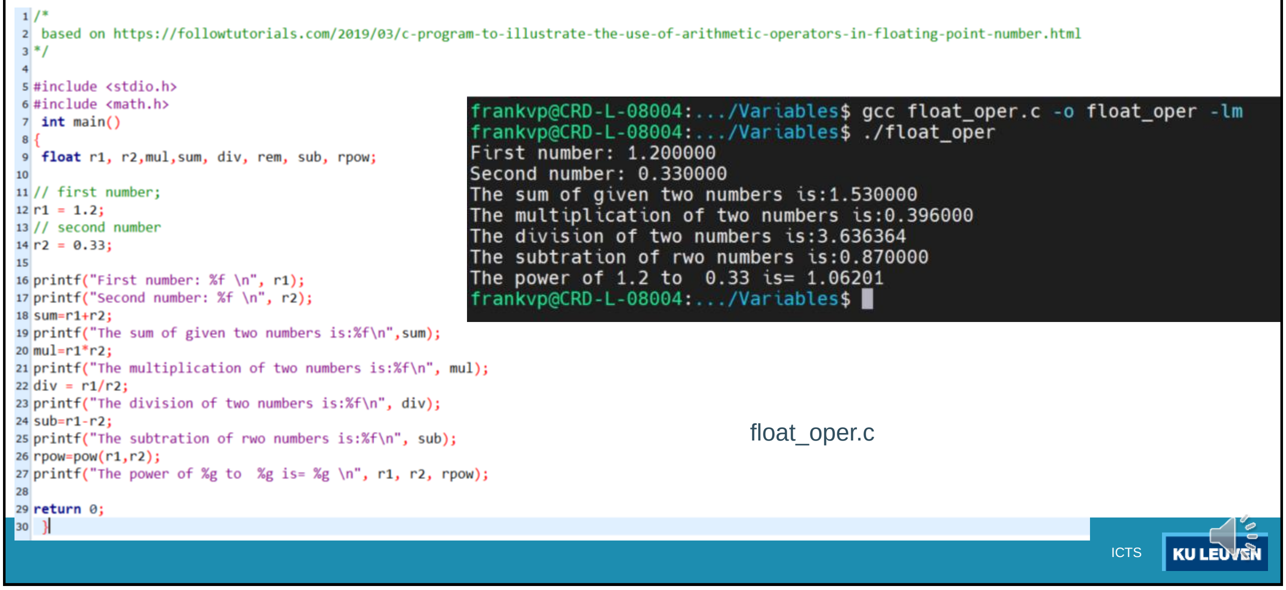
```
#include <math.h>
```

Link with -lm option!

- Power operator (^or **)
  - Not available
  - Use pow function

    ```
    r = pow(x, y)
    ```

  - assumes x and y are of type double.
  - File: pow_exp.c



```c
/*
 based on https://followtutorials.com/2019/03/c-program-to-illustrate-the-use-of-arithmetic-operators-in-floating-point-number.html
*/

#include <stdio.h>
#include <math.h>
 int main()
{
 float r1, r2,mul,sum, div, rem, sub, rpow;

// first number;
r1 = 1.2;
// second number
r2 = 0.33;

printf("First number: %f \n", r1);
printf("Second number: %f \n", r2);
sum=r1+r2;
printf("The sum of given two numbers is:%f\n",sum);
mul=r1*r2;
printf("The multiplication of two numbers is:%f\n", mul);
div = r1/r2;
printf("The division of two numbers is:%f\n", div);
sub=r1-r2;
printf("The subtration of rwo numbers is:%f\n", sub);
rpow=pow(r1,r2);
printf("The power of %g to  %g is= %g \n", r1, r2, rpow);

return 0;
 }
```

```
frankvp@CRD-L-08004:.../Variables$ gcc float_oper.c -o float_oper -lm
frankvp@CRD-L-08004:.../Variables$ ./float_oper
First number: 1.200000
Second number: 0.330000
The sum of given two numbers is:1.530000
The multiplication of two numbers is:0.396000
The division of two numbers is:3.636364
The subtraction of rwo numbers is:0.870000
The power of 1.2 to  0.33 is= 1.06201
frankvp@CRD-L-08004:.../Variables$
```

float_oper.c

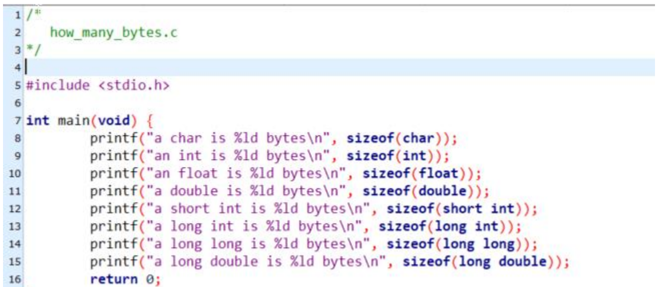
floating point: tips

- value kept in memory is usually not exact.
  2/3 will be saved as 0.6666667
- when comparing non-integer numbers, try to use a range instead of an exact equality
  ex.
  in between -0.1E-30 +0.1E-30
  instead of =0
- double will be used more often than float

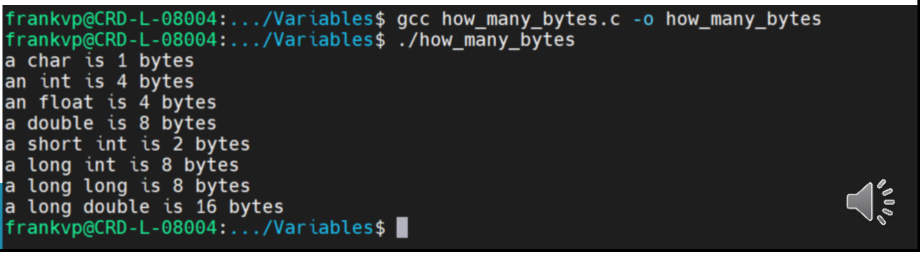
sizeof

- **`sizeof`** operator, returning the size of the types in bytes
  - useful when using complex data types
  - ex. dynamic memory allocation
- Format
  `sizeof` *`identifier`*
  `sizeof` *`type`*
  - () not necessary, but useful
  - `sizeof(int)`
    `sizeof(double)`
    `sizeof(long double)`
    `sizeof(x)`

- *how_many_bytes.c*



```
/*
    how_many_bytes.c
*/

#include <stdio.h>

int main(void) {
        printf("a char is %ld bytes\n", sizeof(char));
        printf("an int is %ld bytes\n", sizeof(int));
        printf("an float is %ld bytes\n", sizeof(float));
        printf("a double is %ld bytes\n", sizeof(double));
        printf("a short int is %ld bytes\n", sizeof(short int));
        printf("a long int is %ld bytes\n", sizeof(long int));
        printf("a long long is %ld bytes\n", sizeof(long long));
        printf("a long double is %ld bytes\n", sizeof(long double));
        return 0;
}
```



```
frankvp@CRD-L-08004:.../Variables$ gcc how_many_bytes.c -o how_many_bytes
frankvp@CRD-L-08004:.../Variables$ ./how_many_bytes
a char is 1 bytes
an int is 4 bytes
an float is 4 bytes
a double is 8 bytes
a short int is 2 bytes
a long int is 8 bytes
a long long is 8 bytes
a long double is 16 bytes
frankvp@CRD-L-08004:.../Variables$
```

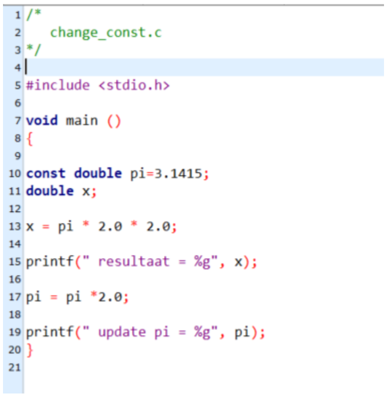
More Type Qualifiers

`const`

- indicates that a variable is intended to remain constant and should not be changed.

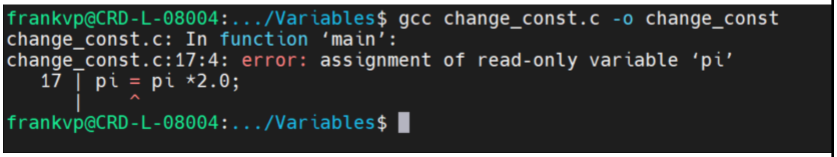
```
const int DoesNotChange = 5;
DoesNotChange = 6; /* will not compile */
```

- constant can be useful:
  - tells immediately that the value will not change (makes code more readable)
  - tells the compiler that the value will not change (can influence the speed)

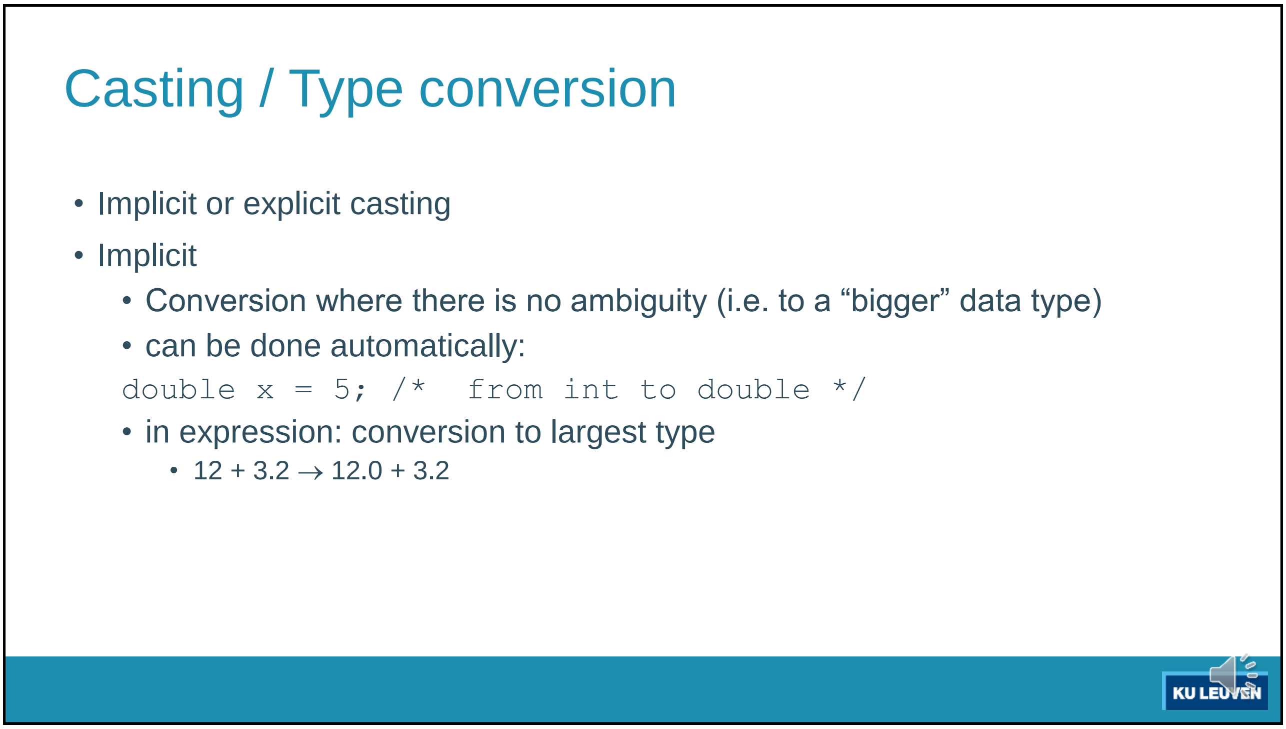


```
1  /*
2     change_const.c
3  */
4
5  #include <stdio.h>
6
7  void main ()
8  {
9
10 const double pi=3.1415;
11 double x;
12
13 x = pi * 2.0 * 2.0;
14
15 printf(" resultaat = %g", x);
16
17 pi = pi *2.0;
18
19 printf(" update pi = %g", pi);
20 }
21
```

- change_const.c



```
frankvp@CRD-L-08004:.../Variables$ gcc change_const.c -o change_const
change_const.c: In function ‘main’:
change_const.c:17:4: error: assignment of read-only variable ‘pi’
   17 | pi = pi *2.0;
      |    ^
frankvp@CRD-L-08004:.../Variables$ █
```



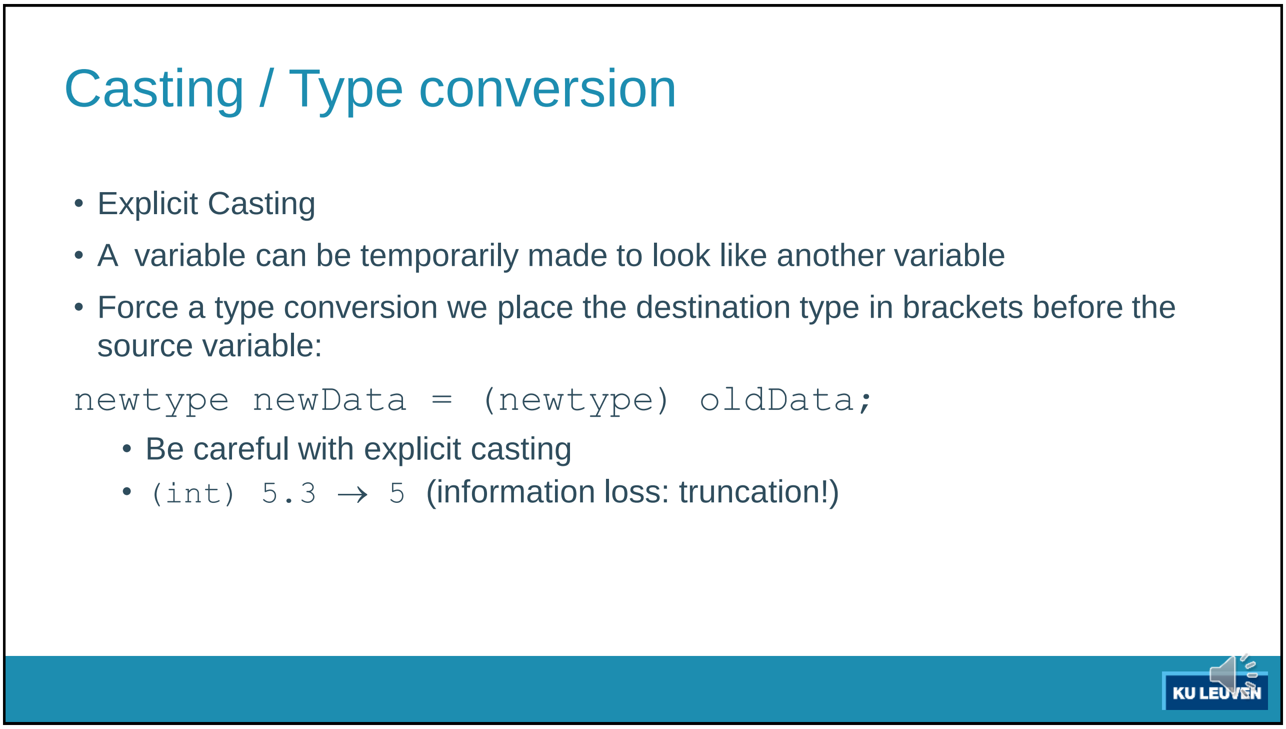
Casting / Type conversion

- Implicit or explicit casting
- Implicit
  - Conversion where there is no ambiguity (i.e. to a “bigger” data type)
  - can be done automatically:

```
double x = 5; /*  from int to double */
```

  - in expression: conversion to largest type
    - 12 + 3.2 → 12.0 + 3.2

KU LEUVEN



Casting / Type conversion

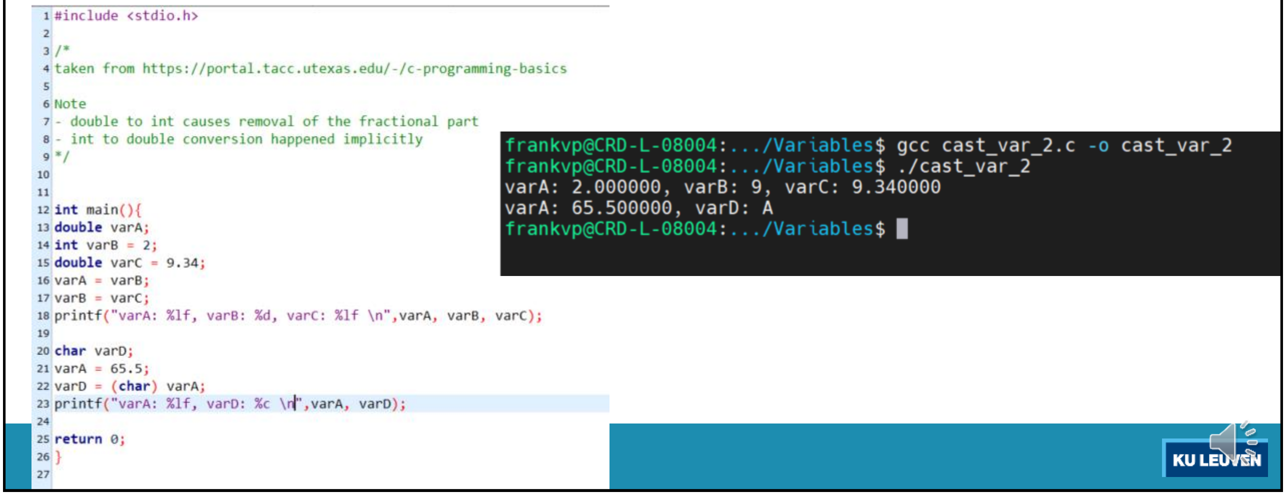
- Explicit Casting
- A variable can be temporarily made to look like another variable
- Force a type conversion we place the destination type in brackets before the source variable:

```
newtype newData = (newtype) oldData;
```

  - Be careful with explicit casting
  - `(int) 5.3 → 5` (information loss: truncation!)

KU LEUVEN

cast_var_2.c



```c
#include <stdio.h>

/*
taken from https://portal.tacc.utexas.edu/-/c-programming-basics

Note
- double to int causes removal of the fractional part
- int to double conversion happened implicitly
*/


int main(){
double varA;
int varB = 2;
double varC = 9.34;
varA = varB;
varB = varC;
printf("varA: %lf, varB: %d, varC: %lf \n",varA, varB, varC);

char varD;
varA = 65.5;
varD = (char) varA;
printf("varA: %lf, varD: %c \n",varA, varD);

return 0;
}

```

```
frankvp@CRD-L-08004:.../Variables$ gcc cast_var_2.c -o cast_var_2
frankvp@CRD-L-08004:.../Variables$ ./cast_var_2
varA: 2.000000, varB: 9, varC: 9.340000
varA: 65.500000, varD: A
frankvp@CRD-L-08004:.../Variables$
```