Welcome to LABC

A few points about the exercises

- YOU need to debug your code
  - Not all tests will be published
- Run on school computers
- Organize your time
- Pre-submission script
  - Start with it
- Design and style
- Use Google for any question you have

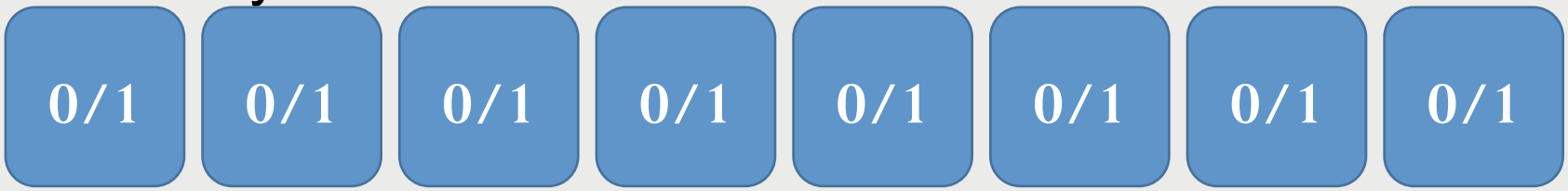
Tirgul 1 - Agenda

- Basic syntax
- Primitive types
- Expressions and operators
- Basic Compilation

Memory – few definitions

- Bit – a binary digit - zero or one

- Byte – 8 bits



- Not C specific

C and Java

- Basic Types are similar to JAVA
  - int, double, char, …
- Basic flow-control is similar –
  - `if (condition) {… }`
  - `for (start; condition; increment){… }`
  - `while (condition) {… }`
  - etc…
- No Classes!
  - no methods, no packages….
- No exceptions
  - programs usually simply crash! "התוכנית עפה"

Types

- C is a “statically-typed language”
  - The compiler must know the type of each variable at compile-time
- The variable type determine:
  - Its **size**
  - Its **internal representation** – the exact bit pattern
  - The **operators** that can be used with it
- Part of the above is defined by the *C standard*, while part of it is *implementation dependent*

Integral data types

Size rules (C's standard):

1. Size of char = 1 byte
2. Size of short ≤ size of int ≤ size of long

Variables types in C

- Simple types:
  - Arithmetic types
  - Pointers
  - Enumeration types
- Aggregate types:
  - Arrays
  - Structs

- User can define new types (e.g. structs)
- User can define new names for each type
  - and use '*typedef' to make things nicer*

Arithmetic data types

- Arithmetic data types are similar to JAVA:
- Integral types:
  - `char`
  - `int`
  - `short`
  - `long`
- Floating-point types:
  - `float`
  - `double`

- However, unlike in JAVA, the types sizes are machine dependant!

Undetermined type sizes

- Advantage:
  - Better hardware support for arithmetic operations
- Disadvantage:
  - Might cause problems with porting codes from one machine to another
- **Knowing type size is important**
  - For dynamic memory allocation
  - For knowing what value range it can store
  - Watch overflow!

The sizeof() operator

- The operator sizeof (type) returns the size of the type.
- It is evaluated at compile time!

Example:

```
printf("%lu", sizeof(char)); //charSize == 1
```

Integers binary representation (the two-complement representation)

Bit Pattern	Unsigned	2's Complement
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
•	•	•
•	•	•
0111 1110	126	126
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
•	•	•
•	•	•
1111 1110	254	-2
1111 1111	255	-1

signed VS. unsigned

- Each type could be signed (represent both negative and positive numbers) or unsigned (only positive numbers).

- Examples:
  - `int negNum = -2;`
  - `int posNum = 2;`
  - `unsinged int posNum = 2;`
  - `unsinged int posNum = -2;`

```
// this would result in implicit conversion
```

Integers internal representation

- *unsigned integers* types are represented with the *binary representation* of the number they stand for.
- The representation range therefore is: $[0,2^x-1]$,

  Where x is the size in bits of the type.
  - For example,

    if `sizeof(unsigned short) == 2` unsigned short: $[0,2^{16}-1]$

Integers binary representation

- The exact representation of signed integers types is machine dependant
- The most popular representation is the two-complement representation
- The representation will cause different behavior in case of overflow and conversions
- The representation range is: $[-2^{x-1},2^{x-1}-1]$

  Where x is the size in bits of the type.
  - For example, if sizeof(short) == 2
    - signed short: $[-2^{15},2^{15}-1]$

Integers binary representation

```
short num = -1;
printf("%u",num);
```

- The output on the cs school machines is: 65535
- What happened?
- -1 -> 11111111 11111111 (signed short representation).
- 1111111...1->65535 when interpreted as **unsigned** short.

ascii table

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Integral types - characters

- chars can represent small integers or a character code.

- Examples:

  - `char c = ‘A’;`
  - `char c = 65;`

Arithmetics with character variables

```
char ch = 'A';
printf("The character %c has the ASCII code %d.\n",
   ch, ch);
for ( ; ch <= 'Z'; ch++ )
{
     printf("%c", ch);
}
```

Characters input and output

```
#include <stdio.h>
  int main ()
  {
     char c;
     puts ("Enter character:");
     c=getchar();
     puts ("You entered:");
     putchar (c);
     return 0;
  }
```

Floating Point troubles

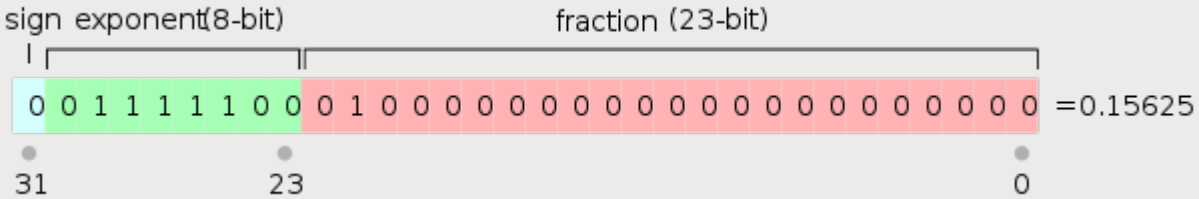
```
float f;
for (f=0;f<20000000;f+=0.001);
```

A bit more in your HW

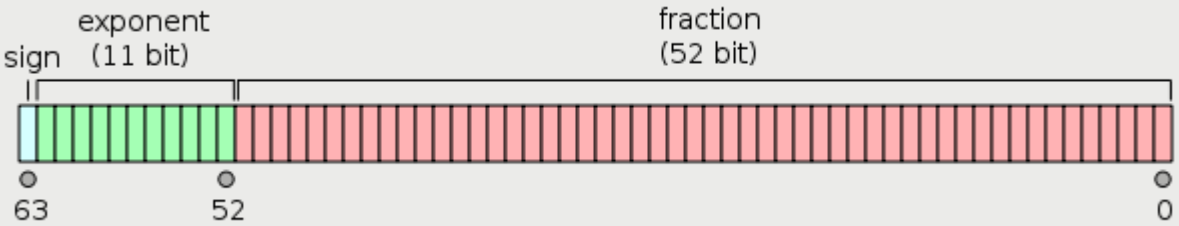
- This code will not stop. Why?
- You will learn about it in digicomp, but:
  - Floating point numbers have adaptive "resolution" - when dealing with large numbers small numbers are insagnificant, and discarded: $1*2^{50}+1*2^{-50} == 1*2^{50}$.
  - The resoultion of a float is about 7 digits, and of double is about 15.
    - **=> Don't do any calculations / comparisons that involve floating point numbers of different magnitude**

Floating point representation

- Unspecified by C standard.
- On most systems – IEEE-754.
- float:



- double:



- See nice explanation here:
  http://steve.hollasch.net/cgindex/coding/ieeefloat.html
- http://www.cprogramming.com/tutorial/floating_point/understanding_floating_point_representation.html

Casting and Type Conversion

- Casting possible between all primitive types.
- Casting up - usually automatic:
  - float => double
  - short => int => long

  etc.

```
int i;
short s;
long l;
i=s;    // no problem
l=i;    // no problem
s=l;    // might lose info,
        // warning not
guaranteed
```

Integer division

- Mathematical operators on (only) int operands
  - The result is int

```
float f=1/3;          //0
float f=1/3.0;        //0.3333..
float f=(float)1/3; //0.3333..
```

Expressions as Values

```
int i=0;
if (i=5)
{
   printf("hello\n");
}
```

- Will we enter the *if statement?*
- We probably meant “i==5” on line 2
- Completely legal!

Expressions as Values

- && - logical “and”, & - bitwise “and”
- || - logical “or”, | - bitwise “or”

```
int i=0;
if (i==1 && x=isValid())
{
    …
}
```

Might not be evaluated

Wish you a nice course and semester ☺

- Make sure to check the site
- Good luck!