Week 1

- Structure of a simple C program
- Input and output
- Some specifications (reserved words, arithmetic operators,…)

A simple C program

```
/* A comment */
#include <stdio.h>

int main(void) {
 printf("Hello");
 return(0);
}
```

Notes on a simple program

All statements are terminated by semicolons`;`

Comments

`/ * --- * /`

- multiline comment

`include <stdio.h>` corresponds to the Python's `import` statement

- In C, the **preprocessor** processes lines beginning with the # character
- The preprocessor replaces the line `#include <stdio.h>` with the textual content of the file `'stdio.h'`.

Notes on a simple program

The header file `stdio.h` stands for **standard input** `(stdin)` and **standard output** `(stdout).` It contains definitions of input/output functions, such as:

- `printf()` is used to print the strings, integer, character etc on the output screen.
- `scanf()` reads the character, string, integer etc from the keyboard.
- `fopen()` opens files
- etc…

Notes on a simple program

`int main(void)`

- each program must contain a *main program*

Each function has a **type**.

- Functions return a value of function's type when they terminate

The type of `main` function is an integer (`int`).

- The main function typically returns 0

`return(0)`

- stops the program and returns 0 to the operating system

Notes on a simple program

Code block is the code between “{” and “}”

- In Python, a code block begins with “:” and consist of the *indented lines*
- In C, a block is defined in parentheses, but indentation increases readability

```
printf("Hello")
```

- prints the string that is a parameter
- A sequence of characters in double quotes, like "hello, world\n", is called a **character string**
- “\n” is the notation for the **newline** (one character)

Compiling a program

Editor: VS Code
We compile programs in Linux (WSL) “inside” VS Code
Installation of the tools will be explained in this week’s **tutoring sessions**

In Moodle, there is a section **Lecture example programs**. Under it you find the folders:
Lecture 1, **Lecture 2**, …

- Each lecture’s example programs are in the corresponding folder

File: `hello.c`
Compile and execute:

```
$ gcc hello.c -o hello
$ ./hello
Hello
```

About data types

C language is a strongly typed language. Each variable must be declared before use. The type of the variable cannot be changed (c.f. Python)

Each type has “inner representation” in memory

Types for numbers

- `int` - integer
- `long` - long integer
- `float` - floating point, 32 bits
- `double` - double precision, 64 bits

About data types

Characters and strings

`char` – one character / one byte = 8 bits

`char[20]` – a string is an array of characters for 19 characters that are treated as a single data item. One place is for the null character '`\0`’

`void` is considered a data type, but it is basically a keyword used as a placeholder, where you would put a data type, to represent "no data".

For instance, the function `main` does not take any parameters.

Declaration of variables

A declaration announces the properties of variables.

It consists of a name and a list of variables, such as

```
int fahr, celsius;
float distance;
char name[30];
```

Outputting

`printf()` is a general-purpose **output formatting** function

The first parameter is a string that can contain conversion specifications beginning with "`%`"

The conversion specifications are "placeholders" for arguments separated by commas.

Between the % and the conversion character there may be specifications for *width* and *precision*, like %5.2f

- The first number is the minimal width
- The second is the number of decimals

Types: integer (%d), long (%l), float (%f), char (%c), string(%s)

File: `Example1.c`

Inputting

The function `scanf()` is the input analog of `printf()`

Storing a value in a variable requires the symbol & in front of the variable name.

Every variable is a memory location and every memory location has its address defined which can be accessed using “&”

```
int var;
scanf("%d",&var);
```

Here `var` is value and `&var` is address. The above statement says: read `%d` (integer) type of data into `&var` address.

Inputting strings

```
char s[20];
scanf("%s",s);
```

Here `s` is address not the value because `s` is a character array (string). An array name itself indicates its address: `s` and `&s[0]` are both the same.

The above statement says: read `%s` (array of characters) type of data into address location starting from `s`.

Input and output buffers

C uses a buffer to output or input variables. The buffer stores the variable that is supposed to be taken in (input) or sent out (output).

This may result that a "new line" character remains in the input buffer (`stdin`) and when reading just one character, `scanf()` operation receives the "new line" character from the buffer.

We need to “empty” the input buffer sometimes.

File: `Example2.c`

Characteristics

The C language is a strictly typed language

The internal representations of data types differ:

- Floats and integers are two completely different things
- Characters and strings are two completely different things

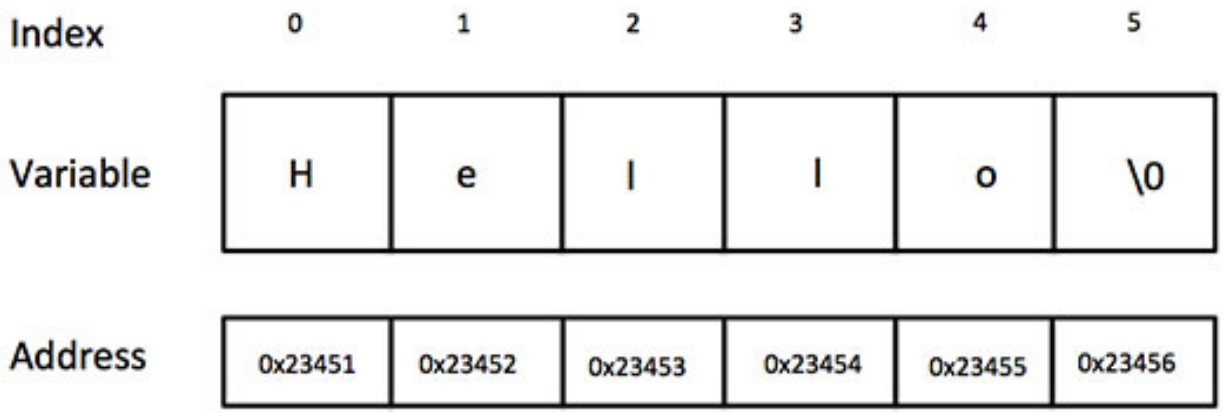
There is no automation, but the programmer does “everything”

The C programmer must know the boundaries and boundary conditions in the environment used. For example, `2147483647 + 1` can be `-2147483648`

Strings

Strings are actually one-dimensional array of characters terminated by a null character '\0'.

```
char greeting[] = "Hello";
```



C reserved keywords

These keywords cannot be used as variable names:

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while
_Packed

Some % specifiers

Specifier	Used For
%c	a single character
%s	a string
%hi	short (signed)
%hu	short (unsigned)
%Lf	long double
%n	prints nothing
%d	a decimal integer (assumes base 10)
%i	a decimal integer (detects the base automatically)
%o	an octal (base 8) integer
%x	a hexadecimal (base 16) integer
%p	an address (or pointer)
%f	a floating point number for floats
%u	int unsigned decimal
%e	a floating point number in scientific notation
%E	a floating point number in scientific notation
%%	the % symbol

Example

File: `Example3.c`

```
The number 2147483647 in HEX is 7fffffff.
The number -2147483648 in HEX is 80000000.
```

Hex `f` corresponds to `1111`

Hex `7` corresponds to `0111`

Hex `8` corresponds to `1000`

Arithmetic Operators

Operator	Meaning
+	addition or unary plus
−	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Increment and decrement operators

C has two operators increment `++a` and decrement `--a` to change the value of the variable `a` by 1. These two operators are unary operators, meaning they only operate on a single operand.

File: `Example4.c`

These two operators can also be used in this way: `a++` and `a--`

- Find out how they work

Relational operators

A relational operator checks the relationship between two operands. If the relation is true, it returns `1`; if the relation is false, it returns `0`.

Operator	Meaning	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Logical operators

Logical operator returns either `0` or `1` depending upon whether expression results true or false.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals 0
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals 1
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals 0

Bit-wise operators

In arithmetic-logic unit (which is within the CPU), mathematical operations like addition, subtraction, multiplication and division are done in bit-level.

In addition to this, C has the following bit-wise operators:

Operator	**Meaning**
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right