Variables
00000

Data types
000

Variable I/O
0000

C introduction

Variables

Variables
00000

Data types
000

Variable I/O
0000

Contents

Variables

Data types

Variable I/O

Variables
●○○○○

Data types
○○○

Variable I/O
○○○○

Usage

Declaration:

```
type identifier;
```

Assignment:

```
identifier = value;
```

Definition (all at once):

```
type identifier = value;
```

Example:

```
int number;                  /* declaration */
number = 42;                 /* assignment */
int another_number = 23;     /* definition */
```

Variables
○●○○○

Data types
○○○

Variable I/O
○○○○

Saving lines

Multiple declarations

```
int number, another_number;
```

Multiple Definitions

```
int number = 42, anothernumber = 23;
```

But be careful:

```
int a = 23, b = 23;
```
$\neq$
```
int a, b = 23;
```

$\rightarrow$ Avoid multiple variable definitions at one line!

Variables
○○●○○

Data types
○○○

Variable I/O
○○○○

Valid identifiers

- ▶ Consist of English letters (no *ß*, *ä*, *ö*, *ü*), numbers and underscore (_)
- ▶ Start with a letter or underscore
- ▶ Are case sensitive (*number* differs from *Number*)
- ▶ Must not be reserved words:

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

Style:

- ▶ Stay in one language (English recommended)
- ▶ Decide whether to use *camelCaseIdentifiers* or *snake_case_identifiers*.

Speaking identifiers

```
1  /* calculate volume of square pyramid */
2  int a, b, c;
3  a = 3;
4  b = 2;
5  c = (1 / 3) * a * a * b;
```

⇓

```
1  /* calculate volume of square pyramid */
2  int length, height, volume;
3  length = 3;
4  height = 2;
5  volume = (1 / 3) * length * length * height;
```

Variables
○○○○●

Data types
○○○

Variable I/O
○○○○

Use speaking identifiers.

Please, use speaking identifiers.[1]

---
[1]Seriously, use speaking identifiers.

Variables
00000

Data types
●○○

Variable I/O
○○○○

Integer numbers

- ▶ Keywords: *int*, *short*, *long*
- ▶ Stored as a binary number with fixed length
- ▶ Can be *signed*(default) or *unsigned*
- ▶ Actual size of *int*, *short*, *long* depends on architecture

Example (64 Bit):

```
int a;              /* Range: −2.147.483.648 to 2.147.483.647 */
unsigned short b;                        /* Range: 0 to 65.535 */
```

Variables
00000

Data types
0●0

Variable I/O
0000

Floating point numbers

- ▶ Keywords: *float*, *double*, *long double*
- ▶ Stored as specified in *IEEE 754 Standard* TL;DR
- ▶ Special values for ∞, $-\infty$, NaN
- ▶ Useful for fractions and very large numbers
- ▶ Type a decimal point instead of a comma!

Example:

```
float x = 0.125;              /* Precision: 7 to 8 digits */
double y = 111111.111111;     /* Precision: 15 to 16 digits */
```

Variables
00000

Data types
00●

Variable I/O
0000

Characters

- ▶ Keyword: *char*
- ▶ Can be *signed*(default) or *unsigned*
- ▶ Size: 1 Byte (8 Bit) on almost every architecture
- ▶ Intended to represent a single character
- ▶ Stores its *ASCII* number (e.g. 'A' $\Rightarrow$ 65)

You can define a *char* either by its ASCII number or by its symbol:

```
char  a = 65;
char  b = 'A';     /* use single quotation marks */
```

Variables
00000

Data types
000

Variable I/O
●000

printf() with placeholders

The string you pass to *printf* may contain placeholders:

```c
int a = 3;
int b = 5;
float c = 7.4;
printf("a: %d\n", a);
printf("b: %d\nc: %f\n", b, c);
```

Output:

```
a: 3
b: 5
c: 7.4
```

You can insert any amount of placeholders. For each placeholder, you have to pass a value of the corresponding type.

Variables
00000

Data types
000

Variable I/O
0●00

Example placeholders

The placeholder determines how the value is interpreted. To avoid compiler warnings, only use the following combinations:

type	description	type of argument
%c	single character	char, int (if $<= 255$)
%d	decimal number	char, int
%u	unsigned decimal number	unsigned char, unsigned int
%X	hexadecimal number	char, int
%ld	long decimal number	long
%f	floating point number	float, double

Variables
00000

Data types
000

Variable I/O
00●0

Variable input

scanf() is another useful function from the standard library.

- ▶ Like printf(), it is declared in stdio.h
- ▶ Like printf(), it has a format string with placeholders
- ▶ You can use it to read values of primitive datatypes from the command line

Example:

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

After calling scanf(), the program waits for the user to input a value in the command line. After pressing the return key, that value is stored in i.

Note:

- ▶ *scanf()* uses the same placeholders as *printf()*
- ▶ You must type an & before each variable identifier
  (more about this later)
- ▶ If you read a number (using %d, %u etc.), interpretation
  - ▶ Starts at first digit
  - ▶ Ends before last non digit character
- ▶ If you use %c, the first character of the user input is interpreted
  (this may be a ' ' as well!)

Never trust the user: they may enter a blank line while you expect a
number, which means your input variable is still undefined!