C-Lessons

Variables

Lecturers: Mirko Jantschke, Pascal Scholz

Created by: Richard Mörbitz, Manuel Thieme

6. November 2018

Overview

1. Data types
2. Variable I/O
3. Related Task

Usage

Declaration:

```
type identifier;
```

Assignment:

```
identifier = value;
```

Definition (all at once):

```
type identifier = value;
```

Example:

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

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;
```

→ Avoid multiple variable definitions at one line!

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

Style:

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

Speaking identifiers

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

⇓

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

Use speaking identifiers.

Please, use speaking identifiers.[1]

[1] Seriously, use speaking identifiers.

Data types

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 */
```

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 */
```

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' ⇒ 65)

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

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

Variable I/O

printf() with placeholders

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

```
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.

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

Have a look at the Cheatsheet!

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!

Related Task

Task: Rot13

Task as online:
Rot13 is an ancient cryptography procedure. It encrypts a message by shifting each character 13 letters in alphabetical order (modulo z).

Write a program that takes a printable character and prints its Rot13 encrypted version. Assume that the input is always a small character.

Experts: Get it working with keeping the character case (a ->, B ->O). Keep it simple. Do not use stuff that has not been introduced yet.