JTSK-320111

Programming in C I

C-Lab I

Lecture 1 & 2

Dr. Kinga Lipskoch

Fall 2017

Who am I?

- PhD in Computer Science at the Carl von Ossietzky University of Oldenburg
- University lecturer at the Computer Science Department
- Joined Jacobs University in January 2013
- Office: Research I, Room 94
- Telephone: +49 421 200-3148
- E-Mail: k.lipskoch@jacobs-university.de
- Office hours: Mondays 10:00 - 12:00

Course Goals

- Learn the basic aspects of procedural programming
- Learn the basic details of the C programming language
- Write and test simple programs
- "Hands on": not just theory, but practice lab sessions to apply what you have learned in the lectures

Course Details

- 3 weeks = 24 hours
- Every week will consist of
  - 2 lectures: Thu/Fri afternoon, 14:15 - 16:00
  - 2 lab sessions: Thu/Fri afternoon, 16:15 - 18:30
- During each lab session you will have to solve a programming assignment sheet (consisting of multiple exercises) related to the corresponding lecture
- Tutorial offered by TAs as help session during the weekend before the deadline

Course Resources

- Textbook:
  "The C programming language" by B. Kernighan and D. Ritchie, second edition, Prentice Hall, 1988
- Homepage of course: https://grader.eecs.jacobs-university.de/courses/320111/2017_2gA/
  Slides and practice sheets will be posted there
- Program assignments will be received during the lab
  Presence exercises have to be submitted during the lab
- Offline questions: Office hours on Mondays 10:00 - 12:00 or ask for another appointment individually
- Do not hesitate, and do not wait until you are left too much behind

Grading Policy

- Each exercise will be graded (percentages)
- 35% - programming assignments
- 65% - final exam
- In the (written) final exam you will be asked to solve exercises similar to ones in the assignments
- The final exam will take place sometime in October

Programming Assignments

- There will be presence assignments that need to be solved in the lab
- Other assignments are due on the following Tuesday and Wednesday morning at 10:00 sharp
- These assignments are structured in a way that you can solve them during the lab session
- Solutions have to be submitted via web interface to Grader `https://grader.eecs.jacobs-university.de`

Grading Criteria for Programming Assignments

- Assignments are graded by the TAs
- Not just solution itself counts, but also programming style and form
- TAs prepared a grading criteria document
- https://grader.eecs.jacobs-university.de/courses/320111/2017_2gA/Grading-Criteria-C.pdf
- They will use rules to grade your solutions
- Extension of deadline possible only with an official excuse

Lab Sessions

- TAs will be available to help you in case of problems
- Optimize your time: solve the assignments during lab sessions
- Do not copy the solutions for the assignments, it is not only illegal, but you will get 0% and later you will certainly fail the written final exam without practice in programming

Missing Homework, Quizzes, Exams according to AP

- https://www.jacobs-university.de/sites/default/files/bachelor_policies_v1.1.pdf (page 9)
- Illness must be documented with a sick certificate
- Sick certificates and documentation for personal emergencies must be submitted to the Student Records Office by the third calendar day
- Predated or backdated sick certificates will be accepted only when the visit to the physician precedes or follows the period of illness by no more than one calendar day
- Students must inform the Instructor of Record before the beginning of the examination or class/lab session that they will not be able to attend
- The day after the excuse ends, students must contact the Instructor of Record in order to clarify the make-up procedure
- Make-up examinations have to be taken and incomplete coursework has to be submitted by no later than the deadline for submitting incomplete coursework as published in the Academic Calendar

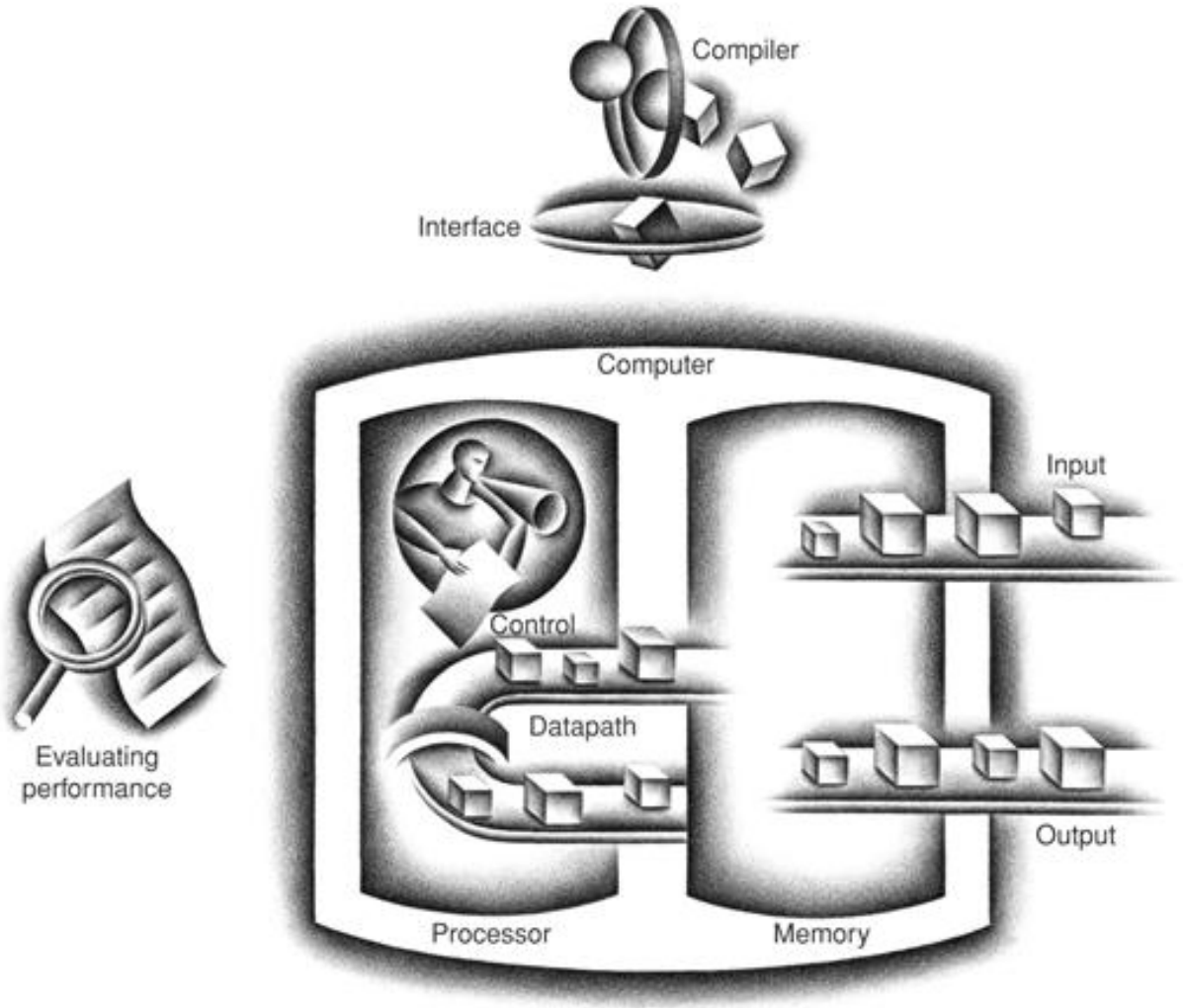
About C

- Widely used general purpose language
- Advantages: small, efficient, portable, structured
- Disadvantages: not user-friendly
- C is an imperative language. You will find many of its characteristics in other imperative languages, such as Pascal or Fortran, but also in scripting languages such as Perl, PHP, Python, etc.

Imperative Languages

- Computation is described in terms of:
  - State (variables)
  - Operations to change this state (assignments, loops, etc.)
- Imperative: first do this, than do that, ...
- There exists other approaches (functional programming, logic programming, object oriented programming, etc.)

Five Classic Components of the Computer



Memory Organization

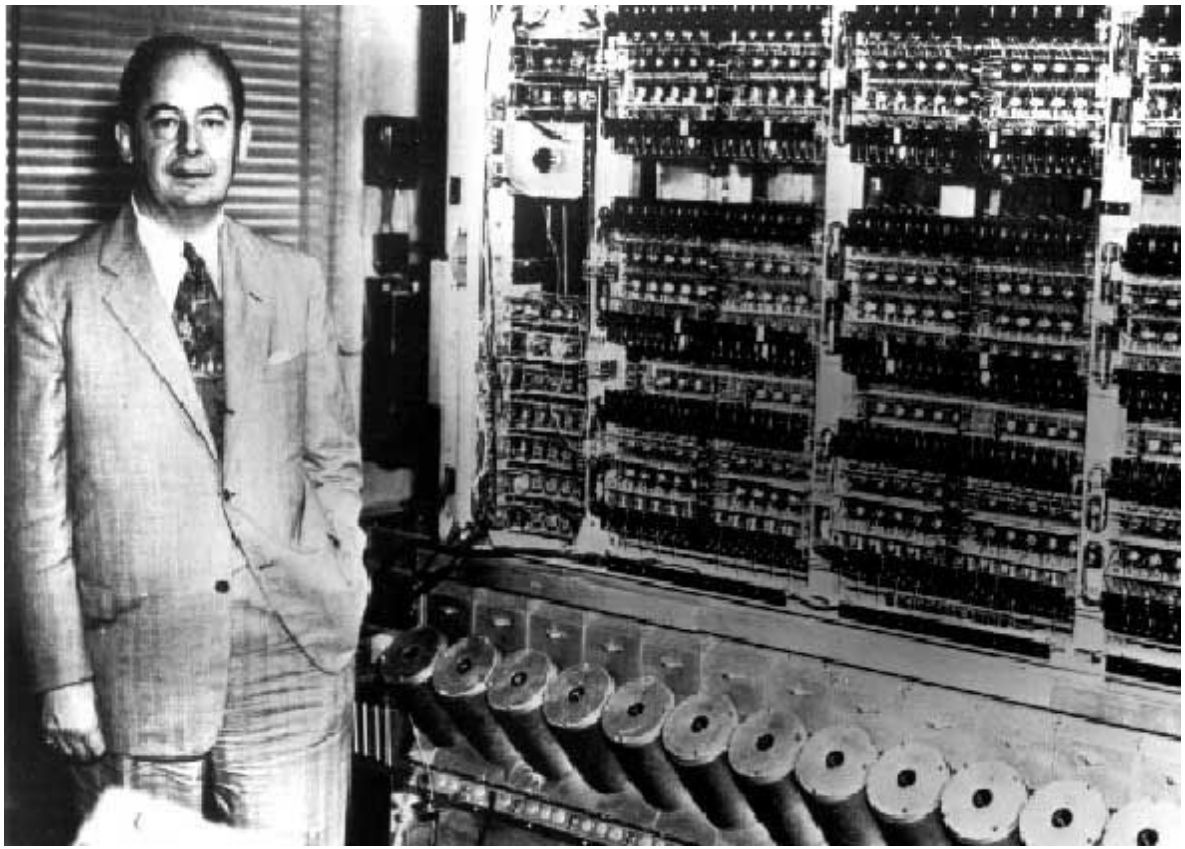
- Viewed as a large, single-dimension array, with an address
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory
- Memory holds data as well as instructions

Address	8 bits of data
0x0001	01111010
0x0002	01101011
0x0003	01100010
0x0004	01101010
0x0005	01111110
0x0006	00101010
...	
0x0008	01101010

Machine-oriented Presentation

- Assembler code: Mnemonic notation
  `add $t0, $s0, $s1`
  Mapping of instruction to the machine level
- Binary representation
- Sequence of 32 '1's and '0's
  00000010001100100100000000100000
  00000010001100100100000000100000
- Well structured for minimal effort when finally mapped to hardware

Electronic Numerical Integrator And Computer (ENIAC) – John von Neumann



Programming Languages

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Assembly language program (for MIPS)

```
swap:
    muli    $2, $5, 4
    add     $2, $4, $2
    lw      $15, 0($2)
    lw      $16, 4($2)
    sw      $16, 0($2)
    sw      $15, 4($2)
    jr      $31
```

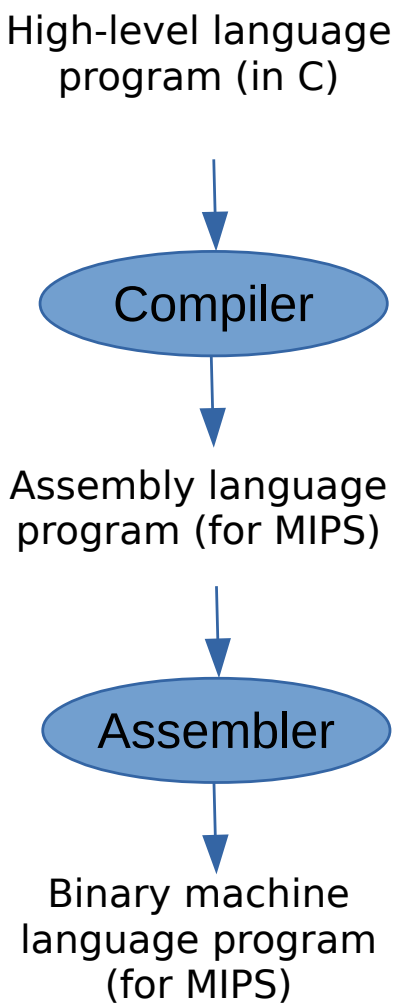
The language is tied to one specific processor type

Higher Programming Languages

- Use symbolic names
- Loops, conditions

High-level language program (in C)

```
1  swap(int v[], int k) {
2    int temp;
3    temp = v[k];
4    v[k] = v[k+1];
5    v[k+1] = temp;
6  }
```

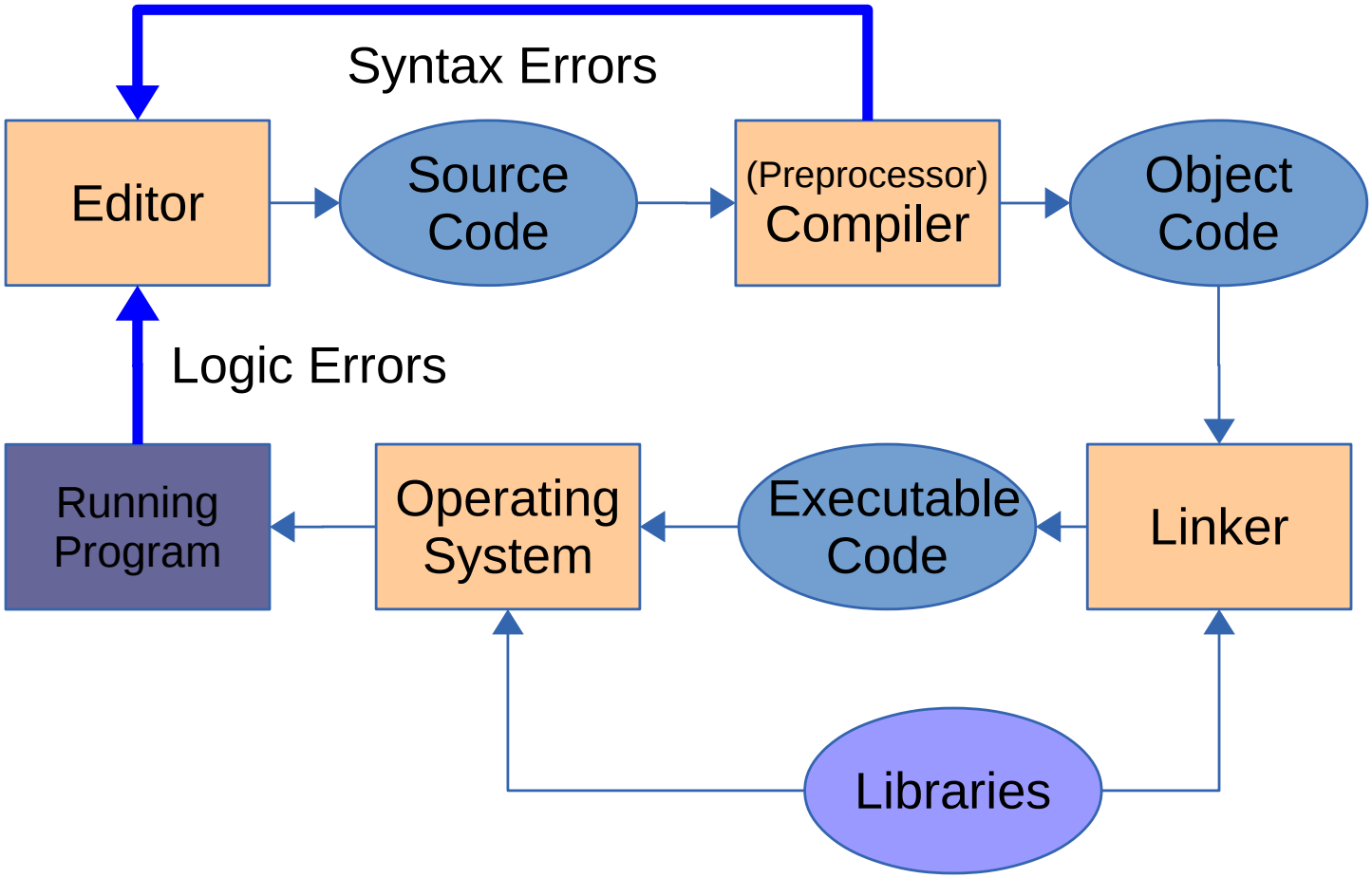



```
swap(int v[], int k) {
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

```
swap:
    muli   $2, $5, 4
    add    $2, $4, $2
    lw     $15, 0($2)
    lw     $16, 4($2)
    sw     $16, 0($2)
    sw     $15, 4($2)
    jr     $31
```

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

C Program Development Cycle



Integrated Development Environment (IDE)

- You can use different IDEs
- You can use the editor of your choice and compile from the terminal
  - `gcc -Wall -out executable program.c`
- If you do not know any of the above, you can use Code::Blocks
- Download and install Code::Blocks from: `http://codeblocks.org/downloads/26`
- If you are a Windows user download `codeblocks-16.01mingw-setup.exe`

The First Program

- A true classic: Hello world
- Open editor → New file
- Type text below, then save as `hello.c`

```
1  /* This is my first C program */
2  #include <stdio.h>
3
4  int main() {
5     printf("Hello world\n");
6     return 0;
7  }
```

Compiling and Running

- The following command performs compilation and linking
  `$> gcc -o hello hello.c`
- If no compilation errors, an executable called `hello` will be created
- If you do not specify o the executable will be called `a.out`
- To execute the program just type its name
  `$> ./hello`

Comments

- It is highly advisable to insert comments into your programs
- Comments start with the couple of characters
  `/* and end with the couple */` or
  `// this is also a comment`
- Everything between these two couples and after
  `// on the same line is considered to be a comment`
- Comments are ignored by the compiler

Including a Library File

- Libraries provide general purpose pieces of software to be reused many times by programmers
- Printing to the screen, getting input from the keyboard, writing to a file, ...
- In order to use them, you have to include them into your program

Header Files

- A header file is a file which contains the description of the resources provided by a library
- Technically it includes the prototypes of the provided functions
- Before using a library you must include the corresponding header file

  ```
  #include <stdio.h>
  ```

- Issue

  ```
  $> gcc -E- o  hello.i hello.c
  ```

  look at the output file `hello.i`

Functions

- A C program is a collection of functions
- A function is a piece of code which can be executed
- Every function has a name
- Functions may return back a value
- Calling a function means to execute the function
- Functions are a wide subject and will be in depth covered in a later lecture

What are Libraries?

- Libraries provide hundreds of functions
- Some are part of the standard definition of the language, others depend on the underlying operating system (OS)
- You do not need to remember them all, but rather learn quickly how to find what you need

  ```
  $> man 3 printf
  ```
- In this course we will use just a few

The `main` Function

- Every C program must have a function called `main()`
- The main function is the logical starting point of the program
- Even if there are 200 functions before ...

```
1  int main() {
2      ...
3      statements...;
4      ...
5  }
```

The `printf` Library Function

- `printf` is a library function used to output data
- To use `printf`, the header file `stdio.h` has to be included
- `stdio` stands for Standard I/O
- `stdio` contains the specification of many general purpose functions for I/O
- `printf` is a very rich and powerful function
- Basic use: printing a sequence of characters
- The following line calls the `printf` function
  `printf("Hello world\n");`
- The sequence of characters is called string
- The sequence is surrounded by quotes

Escape Characters

- `printf` prints the characters in the string
- If a character is preceded by a `\` character, then it is called an escape character
- Escape characters are printed differently and are used to format the output
- Example: `\n` means new line
- Although you type in two characters, internally they is only one character

Some Escape Characters

Character	Meaning
\n	Newline
\a	Bell
\r	Carriage Return
\t	Tabulator
\b	Backspace
\\	\ (Backslash)
\“	“ (Quote)
%% (in the printf control string)	% (Percent)

carriage return used in older bank softwares to overwrite

The `return` Keyword

- When the `return` statement is executed, the current function terminates
- In the example the main function, and then the program, terminates
- `return` can provide a value to be returned
- This will be studied when learning functions in detail

The Course of Semicolons

- Every statement must be terminated by a semicolon ;
- Statements:
  - Variable declarations
  - Function calls
  - Assignments
- Practice will help you . . .

Indentation

- The layout of your program is not important for the compiler
- The program below is correct
- Semicolons are used to determine where a statement ends and where the next starts
- But not only machines will read your programs

```
1    #include <stdio.h>
2    int main(){ printf("Hello\n"); return 0; }
```

- Indent your programs to make them easier to read
- Choose one style and be coherent
- We will look at details of different styles later

Data Types

- A program processes data
- Data can be combined by operators
- The type of a data defines
  - which values can be assumed
  - which operators can be applied
- C is a strongly typed language

the compiler doesn't let you make too many mistakes. strict compiler.

Basic Data Types of C

Data type	C identifier
Character	`char`
Integer number	`int`
Floating point number	`float`
Double precision number	`double`
No type	`void`

Moreover there exist some modifiers that can be applied to the basic data types

Characters

- `char` data type is used to store characters (ASCII code)
- A character is a symbol surrounded by a single quote like `'A'`, `'('` and so on
- C does note provide a string data type
- Strings are dealt as sequences of characters
- Details will follow in later lectures

ASCII Table

32:	48: 0	64: @	80: P	96: `	112: p
33: !	49: 1	65: A	81: Q	97: a	113: q
34: "	50: 2	66: B	82: R	98: b	114: r
35: #	51: 3	67: C	83: S	99: c	115: s
36: $	52: 4	68: D	84: T	100: d	116: t
37: %	53: 5	69: E	85: U	101: e	117: u
38: &	54: 6	70: F	86: V	102: f	118: v
39: '	55: 7	71: G	87: W	103: g	119: w
40: (	56: 8	72: H	88: X	104: h	120: x
41:)	57: 9	73: I	89: Y	105: i	121: y
42: *	58: :	74: J	90: Z	106: j	122: z
43: +	59: ;	75: K	91: [	107: k	123: {
44: ,	60: <	76: L	92: \	108: l	124: |
45: -	61: =	77: M	93:]	109: m	125: }
46: .	62: >	78: N	94: ^	110: n	126: ~
47: /	63: ?	79: O	95: _	111: o	127:

Integers

- Positive or negative numbers without fractional part
  `1, 2, 5, -999, 345302049`
- Maximum and minimum values depend on the system
- The standard does not define this aspect

Floating Point Numbers and Doubles

- Numbers with a fractional part

  `2.3, 3.14, 0.293939`

`float` used to represent real numbers, they offer just a mere approximation

- `double` uses twice the number of bytes to represent numbers
  - Increased precision
  - Increased memory size
  - Increased time to process

Variables (1)

- A variable is a named location in the computers memory used to store a certain data type
- Variables content vary over time: they can be read or written
- Variables must be declared before use
- Every time you need to store some data in your program a variable is needed
- The type of a variable is fixed
- Variables are created and destroyed on the fly (more in the future)
- The content of a variable is retained as long as the variable is present

Variables (2)

- Have a name
- Carry a type
- Hold a value
- Are located at a specific memory address

Declaring Variables

- To declare a variable there is a fixed syntax
  - First data type and then variable name
  - Variable declarations are statements and must be terminated by a semicolon
- Consider the following:

```
int firstVariable;
```

  - What is the value of `firstVariable`?
  - We do not know, and it cannot be known
  - Variable declaration just reserves enough space for the given type, and ties a name, but does not write anything to memory

Initialization of Variables

- Using a non-initialized variable is a common error
- In C it is possible to declare and initialize a variable at the same time

```
1    int firstVariable = 23;
2    float weight = 3.45;
3    char first = 'A', second = 'B';
```

Naming Variables

- Give variables meaningful names
  - Avoid too short or too long names
  - There are exceptions for loop variables (`i`, `j`, `k`, `m`)
- Rules:
  - First character must be a letter or the underscore
  - The remaining characters can be letters, numbers and underscores
  - No spaces are allowed
- A variable cannot have the same name as C keywords

Naming Variables: Example

- These are valid identifiers

```
1        int firstVariable;
2        float _startingWithUnderscore;
3        char _99adsfq_743m_;
```

- These are not valid identifiers

```
1        int first Variable;
2        float 945_temperature;
3        char float;
4        int some%data;
```

Example: Swapping Variables

```
1   #include <stdio.h>
2   int main() {
3     int a, b, swap;
4     a = 45;
5     b = 0;
6     printf("a=%d b=%d\n", a, b);
7     printf("Swap the contents of a and b\n");
8     swap = a;
9     a = b;
10    b = swap;
11    printf("a=%d b=%d\n", a, b);
12    return 0;
13  }
```

Binary Representation of Numbers

Power	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal	128	64	32	16	8	4	2	1
Binary number	0	1	0	1	1	1	0	1

- Different types have different binary representations
- Highest integer number if integer is
  - 1 byte wide?
  - 2 bytes wide?

Types Size: An Example

Data type	Size
`char`	1
`short int`	2
`int`	4
`long int`	4 or 8
`float`	4
`double`	8
`long double`	12 or 16

Modifiers

- C provides some modifiers that apply to the basic data types
- The `long` modifier can be applied to the `int` and `double` data types
- The `signed` and `unsigned` modifiers can be applied to `int` and `char` data types
- The `short` modifier can be applied to the `int` data type
- Modifiers must be put before the data type
- If the data type is `int`, it may (but should not) be omitted

```
1    unsigned int modifiedVariable;
2    long double somevar;
3    unsigned unsignedVariable;
```

Operators

- Operators perform mathematical or logical operations on data
- Can be roughly divided in arithmetic, relational and logical operators
- Apparently an easy subject, but there are many subtle details to know

Arithmetic Operators

Operator	Integer	Floating point
+	3 + 5 = 8	3.4 + 1.2 = 4.6
-	89 - 2 = 87	9.9 - 1.1 = 8.8
*	22 * 2 = 44	1.2 * 3.4 = 4.08
/	48 / 4 = 12	4.5 / 1.2 = 3.752
% (modulo)	49 % 4 = 1	n/a

Assignment Operator

- The assignment operator = is used to write data to variables
- `lvalue = rvalue`
- `lvalue` is what is on the left
- `rvalue` is what is on the right
  - Could be a variable, a constant, or an expression

Example

```
1  int main() {
2      int first = 4, second = 5;
3      int sum, difference;
4      int product;
5      product = first * second;
6      sum = first + second;
7      difference = first - second;
8      return 0;
9  }
```

- Note that this is a valid C program even if it does not print any of its results
- Modify the program, to print the results

Shorthand Operators

- The following patterns are very common

```
1        int x = 32, y = 50;
2        x = x + 10;
3        y = y * 3;
```

- To shorten the notation, it is possible to use the following abbreviations

```
1        +=        x += 10;        x = x + 10;
2        -=        x -= 10;        x = x - 10;
3        *=        x *= 10;        x = x * 10;
4        /=        x /= 10;        x = x / 10;
5        %=        x %= 10;        x = x % 10;
```

More on `printf`

- `printf` can print strings but also values of variables
- It can also mix the two
- The complete syntax of `printf` is pretty complex and many parameters can be specified

The Control String of `printf`

- The general syntax is the following

  ```
  printf("control string", arg1, arg2, ...);
  ```

- The control string specifies:
  - Which characters have to be printed,
  - How variables have to be formatted,
  - Number of decimal places, their type, etc.
- Note that `printf` accepts a variable number of arguments

Formatting Specification (1)

- Specify the type of the data to be printed

```
1       int a = 45;
2       printf("The value is %d\n", a);
```

- This will print the following:

  `The value is 45`

- Each formatting specification must be matched by a parameter
- To specify wrong control strings is another common error in C programs

Formatting Specification (2)

Formatting specification starts with a % character

Conversion	**Meaning**
%c	Single character
%d or %i	Signed decimal integer
%f	Floating point (decimal notation)
%e	Floating point (exponential notation)
%lf	Double (decimal notation)
%s	String
%%	print the percent sign itself
%p	print address of pointer

Specification of the base

- When printing integers it is possible to specify which base should be used for their representation

Specification	System
%o	Octal
%d	Decimal
%x	Hexadecimal

Formatting Integer Numbers

- It is possible to specify how many digits should be used while printing an integer

```
1     int a = 145;
2     printf("The value is %6d\n", a);
```

- This will print three spaces and then 145 (i.e., 6 places for a three digits number)
- If the number of digits is too small, it will be ignored

```
1       int a = 145;
2       printf("The value is %2d\n", a);
```

- This will print 145 over 3 places

The Precision Modifier

- The precision modifier is written `.number`
- For floating point numbers it controls the number of digits printed after the decimal point

  ```
  printf("%.3f", 1.2);
  ```

  will print 1.200
- If the number provided has more precision than is given, it will rounded

  ```
  printf("%.3f", 1.2348);
  ```

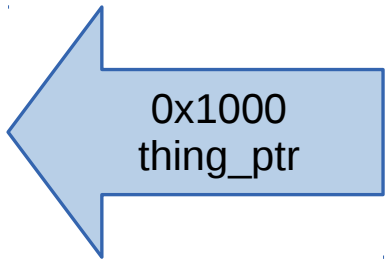
  will display as 1.235

Pointers: Introduction (1)

- Every data is stored in memory
- The memory is organized as a sequence of increasingly numbered cells
  - The number of a cell is its address
  - The address is determined when the variable is declared

0x1000

A thing



A pointer

Pointer Comparison

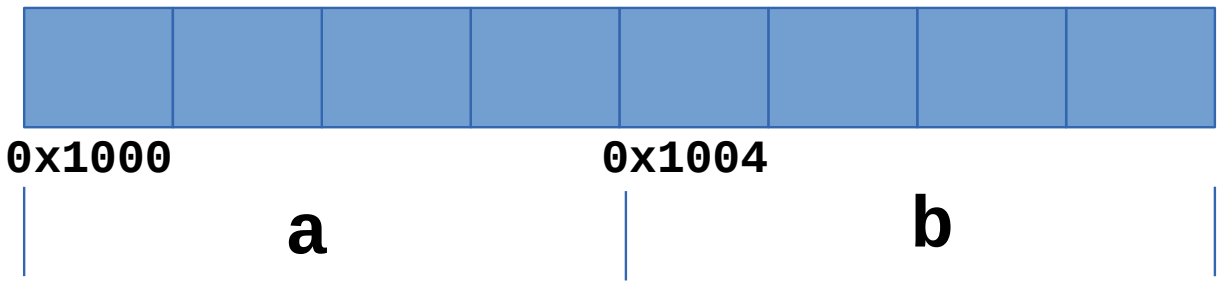
- You can compare pointers and data to street addresses and houses
- Houses may vary in size, while the address which points to the house is approximately of the same size and small
- More than one pointer may point to the same thing

Pointers: Introduction (2)

- Consider the following declaration

```
int a = 145, b = 98;
```

- Supposing that an `int` occupies 4 bytes the following situation could arise



The address of `a` is `0x1000` and the address of `b` is `0x1004`.

Pointers

- A pointer is a variable which stores the address of another variable
- In the previous example, a pointer to `a` would store `0x1000`, as that is its address
- Pointers are variables: thus they can be read, written and so on
- Being variables, they are stored in memory and they have an address as well

Pointers: Declaration and Initialization

- To declare a pointer, it is necessary to specify the type of the pointed object

```
int *thing_ptr;
```

- In this case `thing_ptr` will hold the address of an `int`
- The operator `&` returns the address of a variable

```
thing_ptr = &a;
```

Do We Need Pointers?

- Pointers are a critical aspect of C
- Extremely powerful, allow to solve quickly, efficiently and shortly difficult problems
- Extremely powerful: if you misuse them you can compromise the integrity of your program
  - it is the general source of crashes of programs
- Students usually do not like them, because the different meanings of $*$ is confusing
- You cannot claim you know C if you do not master them

Managing a Variable via a Pointer

- The following pieces of code do the same

```
1    int a = 10;
2    a = a + 5;
```

```
1    int a = 10;
2    int *ptr;
3    ptr = &a;
4    *ptr = *ptr + 5;
```

- Note the different meanings of $*$:
  - Declaration: $*$ declares `ptr` as pointer to integer
  - Next line: $*$ dereference (or content-) operator

The Dereference Operator (The Content Operator)

- The $*$ operator allows you to access the content of the pointed object (content-operator)
- In the previous example we do not increase the pointer, but the content of the pointed object: `a`
- Note that

```
1    int *ptr;
2    ptr = &a;
3    ptr = ptr + 5;
```

  is accepted by the compiler but has a completely different meaning!
  (the pointer itself will be increased by the size of 5 integers meaning $5 * 4 = 20$ bytes)

An Example

```
1   #include <stdio.h>
2   int main() {
3      int a = 7;
4      int *ptr;
5      ptr = &a;
6
7      printf("Address of a: %p\n", ptr);
8      printf("Value of a: %d\n", *ptr);
9      return 0;
10  }
```

Simple Use of Pointers

```c
#include <stdio.h>
int main() {
   int  thing_var; /* def. var. for a thing */
   int *thing_ptr; /* def. pointer to a thing */
   thing_var = 2; /* assign a value to a thing */
   printf("Thing %d\n", thing_var);
   thing_ptr = &thing_var; /* make the pointer
    point to thing_var */
   *thing_ptr = 3; /* thing_ptr points to
    thing_var,so thing_var changes to 3 */
   printf("Thing %d\n", thing_var);
   printf("Thing %d\n", *thing_ptr);
   /* another way */
   return 0;
}
```

Comments

- It is obvious that it is necessary to specify the type of the pointed variable
  - Previous example: how many bytes for the integer: 2 of 4? Knowing the type the answer can be given
- The following does not work

```
1    int a = 45;
2    int *int_ptr = &a;
3    char *char_ptr;
4    char_ptr = int_ptr; /* WRONG */
```

Misuse of Pointers

- `*thing_var`
  - illegal: Get the object pointed to by the variable `thing_var`, `thing_var` is not a pointer, so operation is invalid
- `&thing_ptr`
  - legal, but strange. `thing_ptr` is a pointer: Get address of pointer to a object, result is a pointer to a pointer

Arrays (1)

- Arrays (or vectors) can be defined in different ways:
  - An indexed variable
  - A contiguous memory area holding elements of the same type
- Arrays are useful if you have to deal with many variables of the same type
  - Logically related to each other

The Need for Arrays

- Write a program that reads all the students grades and prints the maximum and minimum, the mean and the variance
  - Assume at most 200 students
- Should we declare 200 float variables?
- From a logical point of view, all the grades belong to the same set of data

Arrays (2)

- An array has a common name and a shared type
- An array has a dimension
- Elements are numbered sequentially
- Elements can be accessed (read/write) in an array by using their position (also called index or subscript)

Arrays in C

- In C you declare an array by specifying the dimension between square brackets

```
int data[4];
```

- The former is an array of 4 elements
- The first one is at position 0, the last one is at position 3

```
1    data[0] = 5;
2    data[1] = 7;
3    data[2] = 3;
4    data[3] = 8;
```

Strings

- Strings are sequences of characters
- Many languages have strings as a built-in type, but C does not
- C has character arrays, with the special character '\0' to indicate the end of the string

```
1  #include <stdio.h>
2  int main() {
3    char name[30] = "Hello World";
4    printf("%s\n", name);
5    name[5] = '\0';
6    printf("%s\n", name);
7    return 0;
8  }
```

Reading Strings from the Keyboard

Standard function `fgets` may be used to read a string from the keyboard.

```
fgets(name, sizeof(name), stdin);
```

- `name`
  - the name of the character array
- `sizeof(name)`
  - maximum number of characters to read
- `stdin`
  - is the file/stream to read from
    `stdin` means standard input (keyboard)

Example Program

```
1   #include <string.h>
2   #include <stdio.h>
3   int main() {
4     char line[100];
5     printf("Enter a line: ");
6     fgets(line, sizeof(line), stdin);
7     printf("You entered: %s\n", line");
8     printf("The length of the line is: %s\n",
9       strlen(line));
10    return 0;
11  }
```

This example contains three errors, one syntax error, and two logical errors. One logical error might crash your program, the other will just count the number of characters you have put wrong.

Reading Numbers with `fgets` and `sscanf`

Use `fgets` to read a line of input and `sscanf` to convert the text into numbers

```
1   #include <stdio.h>
2   int main() {
3       char line[100];
4       int value;
5       printf("Enter a value: ");
6       fgets(line, sizeof(line), stdin);
7       sscanf(line, "%d", &value);
8       printf("You entered: %d\n", value);
9       return 0;
10  }
```

Reading Numbers with scanf

```
1  #include <stdio.h>
2  int main() {
3    int value;
4    printf("Enter a value: ");
5    scanf("%d", &value);
6    printf("You entered: %d\n", value);
7    return 0;
8  }
```

Problems with scanf (1)

```
1   #include <stdio.h>
2   int main() {
3      int nr;
4      char ch;
5      scanf("%d", &nr);
6      // getchar();
7      scanf("%c", &ch);
8      printf("nr=%d, ch=%c\n", nr, ch);
9      return 0;
10  }
```

Assuming that you want to input 12 for `nr` and `'a'` for `ch` then you will experience `nr=12` and `c='\n'` which is not correct, therefore using `scanf` is not advisable in this context

One solution to solve this is to remove the comments in line 6

Problems with `scanf` (2)

- "The function `scanf` works like `printf`, except that `scanf` reads numbers instead of writing them. `scanf` provides a simple and easy way of reading numbers that almost never works. The function `scanf` is notorious for its poor end-of-line handling, which makes `scanf` useless for all but an expert."
- "However we have found a simple way to get around the defiencies of `scanf`, we do not use it."

From: Steve Oualline, Practical C Programming, O'Reilly, 3rd edition

Where to Study?

- Chapter 2 of Kernighan & Ritchie (some part will be covered next week)
- Paragraphs 5.1, 7.2 (and 7.4, but better use `sscanf`)
- If you have time, you can also read chapter one (general overview of the language)