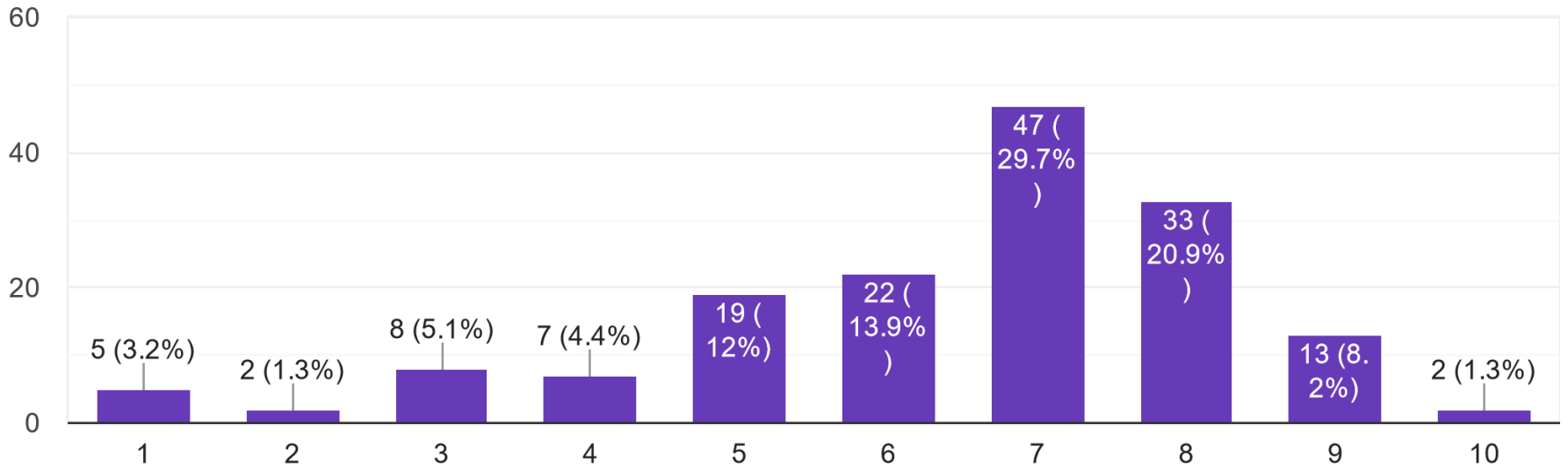
C PROGRAMMING

1st semester 2022-2023

Survey results

How good do you estimate your programming skills?

158 responses



What is the result of the following code?

```
#include <stdio.h>

int main(){
    int sir[5]={1,2,3,4,5}, i;
    int *p;
    p=&sir[2];
    *p=10;
    for(i=0;i<4;i++)
        printf("%d, ", sir[i]);
        printf("%d\n",sir[4]);
    return 0;}
```

History

- C
  - Evolved from BCPL and B, developed by Ritchie
  - Used to implement UNIX
  - Used for other operating systems
- 1970's
  - Evolved to traditional C
- 1980's
  - Standard version occurred (ANSI)

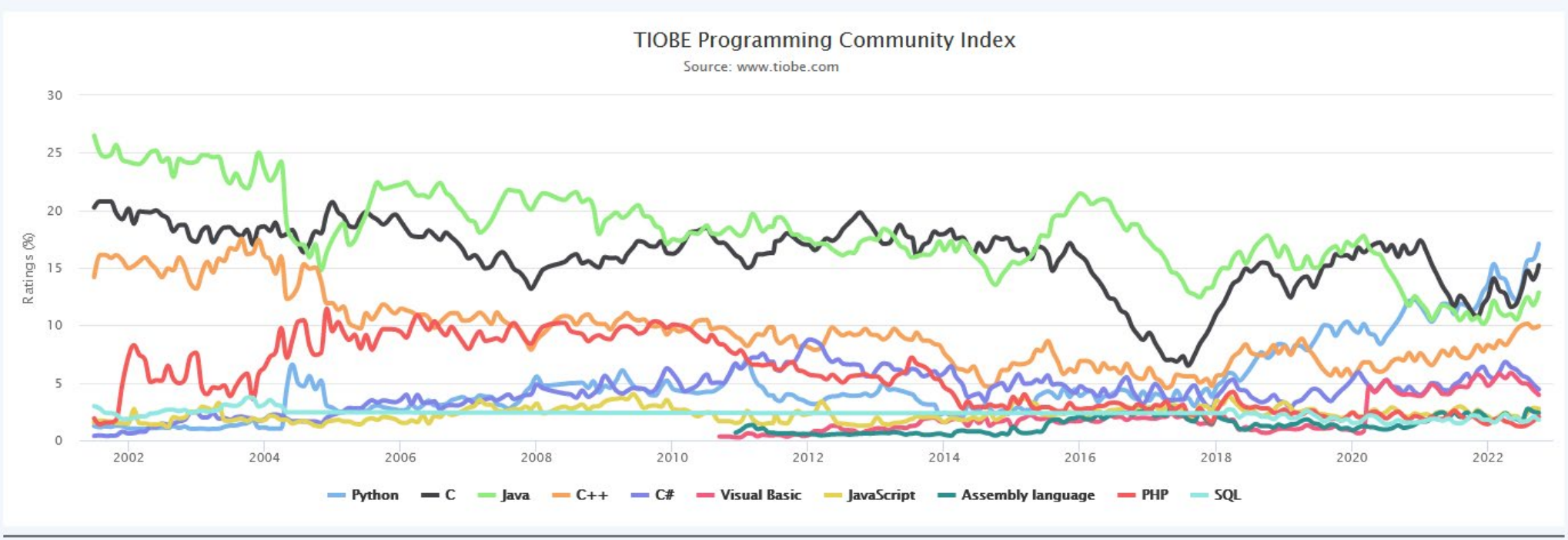
Language Types

1. Machine language
   - Strings of numbers representing instructions
2. Assembly language
   - Human-like abbreviations representing elementary computer operations
3. High-Level languages
   - Code similar to spoken language
   - Use mathematical notations

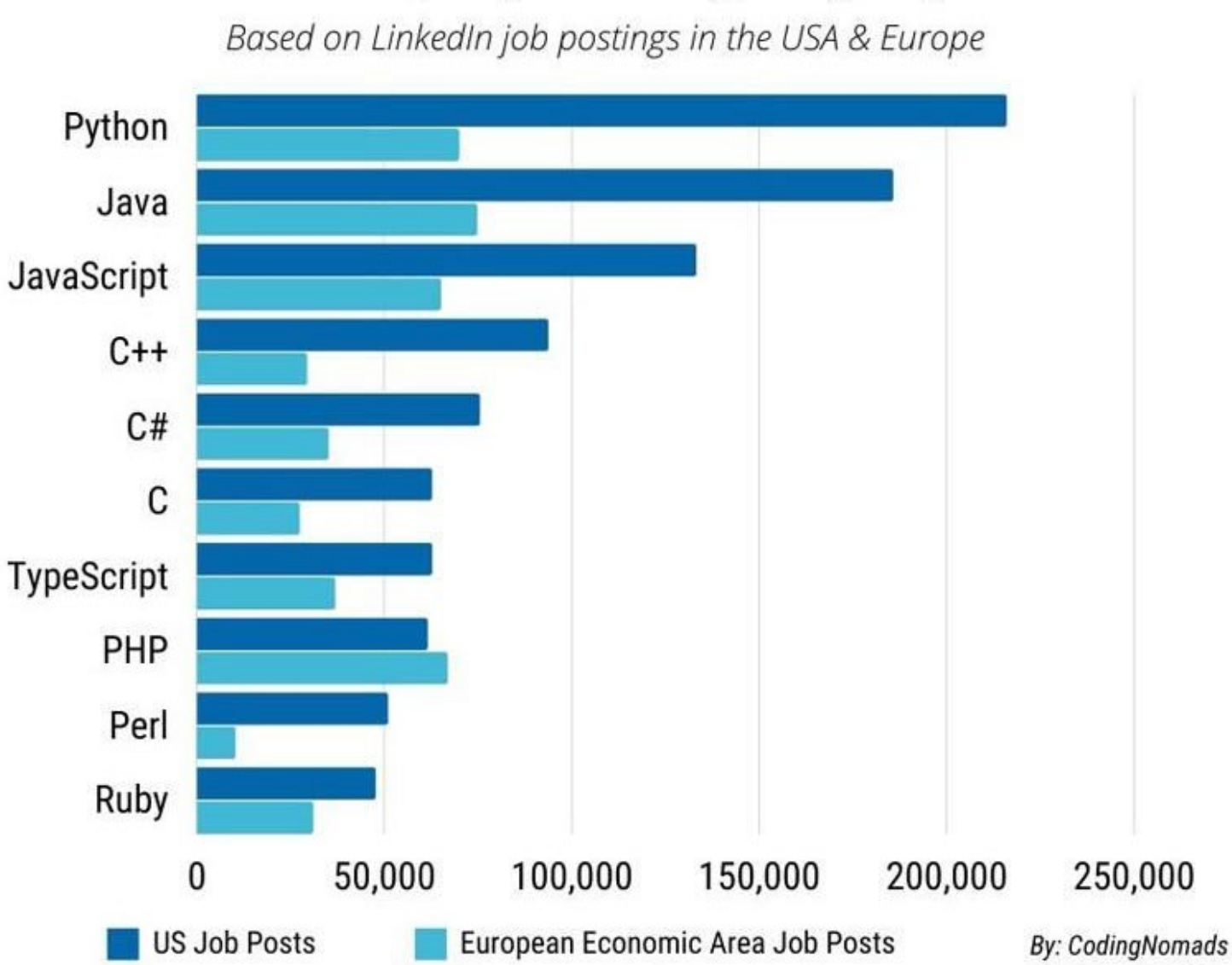
High-level Languages

- “high-level” is a relative term
- C can be seen as a relatively “low-level” high-level language
- Pascal, Fortran, COBOL are typical high-level languages
- Java, Python, Perl, VB are examples of “high-level” high-level languages
- Application specific languages (Matlab, Javascript, VBScript) are even higher-level.

Programming Languages

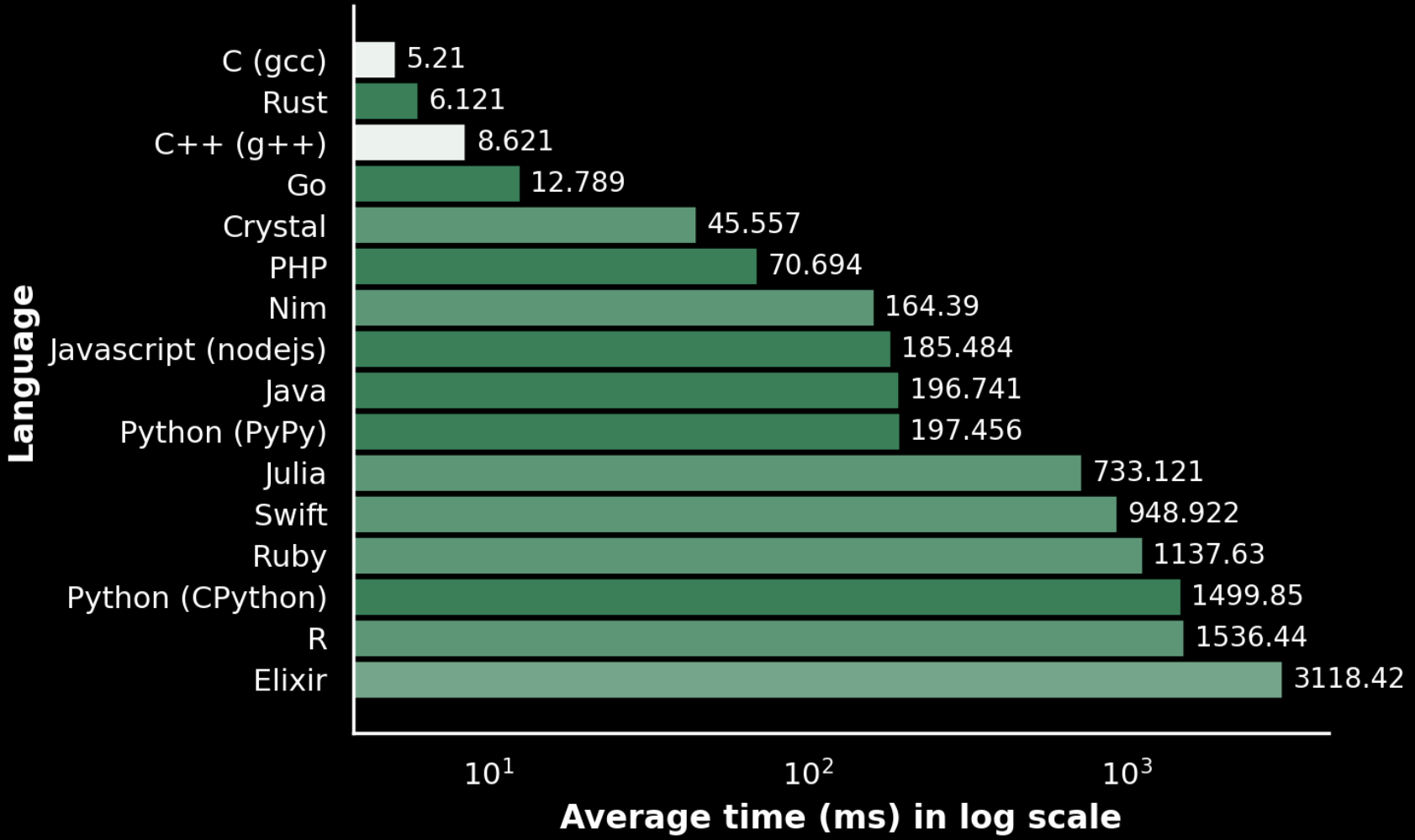


In-demand programming languages of 2022



Speed comparison

Method: calculating π through the Leibniz formula 1000000 times



Generated: 2022-10-02 22:18

https://github.com/niklas-heer/speed-comparison

Programming Paradigms

- Imperative programming
  - statements specified as commands
  - the order of statements execution is important
- Declarative programming
  - describe **what** to do, not **how** to do
  - functional programming
  - logical programming
- Object-oriented programming
  - organized around classes and objects

Programming Language Implementation

- Compilers
  - Scans the entire program and translates it as a whole into machine code
  - Generates intermediate object code which further requires linking
- Interpreters
  - Translates program one statement at a time
  - No intermediate object code is generated
  - Continues translating the program until the first error is met, in which case it stops

C Programming Language

- C is a structured, portable programming language
- Many software applications are written in C
- Useful to make user understand fundamentals of programming
- Simple/short (32 reserved words, 4 basic data types)

C Programming Language

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

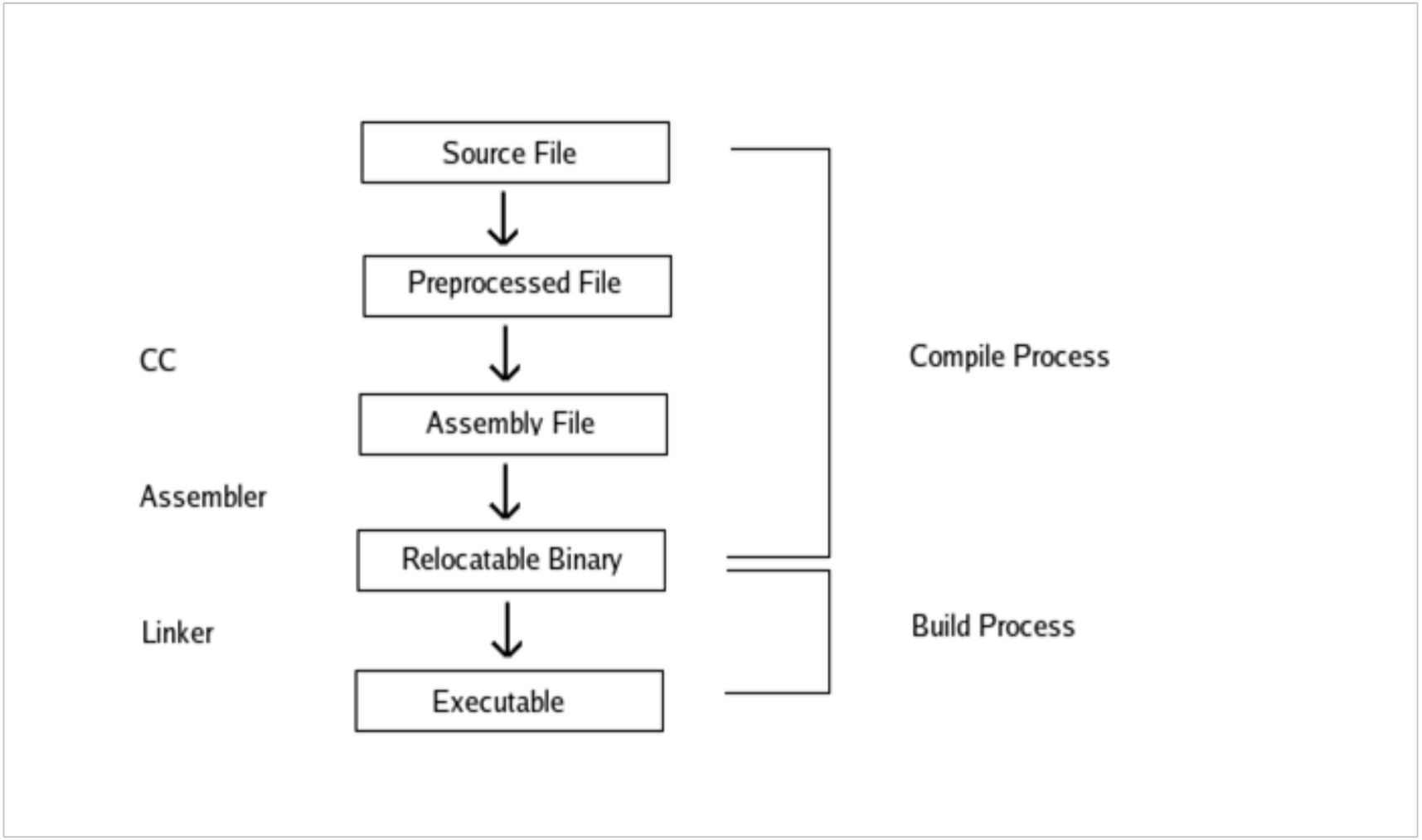
High-level programming

- Source (text) file
  - Instructions that form the program
  - “the look” depends on the programming language
  - File is valid if syntax rules are followed
  - File is called “source code”

High-level programming

- Compile
  - The source code is converted (compiled) into machine language – this is stored in a new file
- Run
  - Execute the machine code file

Compilation Process



C - Introduction

- You need (for this course):
  - Computer with OS (Linux access recommended)
  - Text editor (joe, nano/pico, vi recommended)
  - Compiler (gcc, if using Linux)
- Useful to become acquainted with C in conjunction with Linux and text editors for next semester (OS course)
- If it seems too complicated, you can use Code::blocks or Bloodshed Dev-C++, or something else that uses C compiler

First Program

```
main()
{


}
```

First Program

```
int main()
```

- C programs contain one or more functions, **exactly** one must be called **`main`**
- A function is indicated by parenthesis
- Braces `{ }` indicate a block
- The bodies of functions must be enclosed in braces

First Program

- After writting the source code within an editor, it has to be compiled. Good practice to:
  - Give a meaningful name
  - Add .c “extension” to the file name
- Compile using (on Linux platforms) gcc
  - gcc first.c (will result an output file a.out)
- Run the binary
  - ./a.out (or full path to the binary)

Better First Program

```
/* A first program with no warnings*/
#include <stdlib.h>


int main()
{
        return 0;
}
```

Better First Program

- First we should specify all the preprocessor directives
- Comments are enclosed between /* and */ , no nesting allowed
- **`int`** means that **`main`** "returns" an integer value
- **`return 0;`**
- When exiting a function you should return something
- **`return 0`**, in this case, means that the program terminated successfully

Better First Program

- Compile and execute
  - gcc -Wall first.c -o first
  - ./first
- -Wall turns on all warnings
- -o gives a name to the output binary file instead of default “a.out”

Basic C Data Types

- 4 basic data types:
  - char: holds 1 character in local char set in 1 byte
  - int: holds an integer number (size is platform dependent)
  - float: single precision floating point value
  - double: double precision floating point value
- In addition:
  - void: placeholder o represent incomplete type, or “no data”

char Type

- 1 byte of storage
- Signed or unsigned
- Stored internally as a number (using ASCII char set)
- Remember: Analyze difference when printing with format %d, %c or computing ++

int Type

- Signed integer usually stored on 4/8 bytes (platform 32/64 bits)
- C allows to use also octal and hexadecimal values as int
  - Octal begins with a leading zero (0)
  - Hexadecimal begins with 0x or 0X

  decimal 59, octal 073, hexa 0x3b

float and double Types

- Store 32/64 bit floating point (real) numbers
- Differences are represented by the number of digits for precision (twice number of digits for double than for float)
- float constants can have an f/F suffix
- Can be written using usual notation, scientific notation, exponential notation (e-notation)
- double: $322.56 = 3.2256 \times 10^2 = 3.2256e2$
- float: $322.56f = 3.2256 \times 10^2 f = 3.2256e2F$

Additional types

- In addition to these 4 type there can be constructed using modifiers:
  - unsigned, signed, short (int), long (int, double)
- C99
  - long long
  - int8_t, int16_t, int32_t, int64_t, intmax_t
  - _Bool, _Complex, _Imaginary

Numerical Types Lengths

Type	Macintosh Metrowerks CW (Default)	Linux on a PC	IBM PC Windows XP and Windows NT	ANSI C Minimum
char	8	8	8	8
int	32	32	32	16
short	16	16	16	16
long	32	32	32	32
long long	64	64	64	64

Type	Macintosh Metrowerks CW (Default)	Linux on a PC	IBM PC Windows XP and Windows NT	ANSI C Minimum
float	6 digits	6 digits	6 digits	6 digits
	-37 to 38	-37 to 38	-37 to 38	-37 to 37
double	18 digits	15 digits	15 digits	10 digits
	-4931 to 4932	-307 to 308	-307 to 308	-37 to 37
long double	18 digits	18 digits	18 digits	10 digits
	-4931 to 4932	-4931 to 4932	-4931 to 4932	-37 to 37

Numerical Type Lengths

- Can be found in library files limits.h and float.h
  - CHAR_MAX, INT_MAX, INT_MIN, UINT_MAX
- Can be used a special operator *sizeof*

```
/*  name: types_size.c
*  purpose: find number of bytes ocupied by certain types*/
#include <stdio.h>
#include <limits.h>
#include <float.h>
int main(void) {
   printf("An int occupies: %d bytes \n", sizeof(int));
   printf("Smallest int is: %d\n", INT_MIN);
   printf("Greatest int is: %d\n\n", INT_MAX);
   printf("A double occupies: %d bytes \n", sizeof(double));
   printf("Smallest double is: %e\n", DBL_MIN);
   printf("Greatest double is: %e\n", DBL_MAX);
   return 0;}
```

Variables in C

- A program (with some logic) has at least:
  - Variables declarations
  - Variables initialization
  - Main body
- Variable
  - Storage unit in the program
  - Data type of variable determines storage size

Variables in C

- To place a value in a variable use *assignment*
- Variables must be *declared* before used (assigned)
- A variable is declared when given a name
- C is Case Sensitive; names must begin with a letter and can be followed by any non space character (max length 31 chars to be portable)

Variables in C

- In ANSI C all variables must be declared before any executable statement
- Even if we might initialize variables when declared, in ANSI C they must be declared first and then be given values

```
int var=0;
```

vs

```
int var;
var=0;
```

Variables Declaration

- Variables (usually) must be declared before first executable instruction
- Declarations are constructions like:
  - var_type var_name
    - char c;
    - int value;
    - float first_number, second_number;
- When declared, memory is reserved but they have no meaning until they are assigned a value

Variables Assignment

- Values are assigned to variables using “=” operator
- Values assigned should have proper types, as the variable is declared
- Examples:

```
char c;
int value;
c='c';
value=10;
```

Constant Values

- Fixed values that can’t be modified during program execution
- The type of such a constant is given by the way it is written
- Integer constants

  - can be prefixed by a sign (+ or -)

  - modifiers can be used: U - unsigned, l - long

  - by default are represented as decimal (base 10)

  - to specify a different base, a prefix is needed: 0 (zero) for octal, 0x or 0X for hexadecimal

Constant Values

- Floating point constants
  - by default are of type double
  - can be restricted to float using suffix f, or
  - can be extended to long double using suffix Lf
- Character constants
  - a single character between single quotes or
  - ASCII code
- Character array constants
  - character array specified between double quotes

Structure of a program

```
/* description of program */

#include <stdio.h>
/* any other includes , macro definitions, other
   preprocessor directives go here */

int main(){
   /* program body */
   return 0;
}
```

Program Body

- Begins with variable declarations (usually)
- *V*ariable declarations/initializations are followed by executable statements
- *Executable statements* are represented by code that is not a declaration, like:
  - *“add 1 to the value of nr1 and store the result in nr2”*
  - *“divide nr1 by 2 and test whether it is greater than the value of nr3”*
  - *“print the value of nr1 and nr3”*

Expressions

- Formed by combining constants, variables and operators
- Some of the commonly used operators:
  - Arithmetic operators
  - Assignments
  - Relational operators
  - Logical operators
  - Increment, decrement

Operators - arithmetic

+ addition

- subtraction

* multiplication

/ division

% remainder (modulus)

- The usual precedence applies: *,/, % (same precedence) have greater precedence than +, - (same precedence)
- For the same precedence, are evaluated from left to right

Operators – assignment

= basic assignment operator

+=, -=, *=, /=, %=

- Multiple assignments possible:

  nr1 = nr2 = nr3 = 5

is equivalent to

  nr1 = (nr2 = (nr3 = 5))

- Var op= expr

is equivalent to

  var = var op expr

Operators - relational

== equal to

!= not equal to

> greater then

< less then

<= less then or equal to

>= greater then or equal to

- What about

nr1 == nr2 == nr3

Operators - logical

&& and

- expression formed with && evaluates to 1 (true) if all of its components are true

|| or

- expression formed with || evaluates to 1 (true) if any one of its components is true

! not

- Previous discussion:

  nr1 == nr2 && nr2 == nr3

- Precedence

  ! highest, && next then ||

- Discuss: nr1 && nr2 || nr3 and nr1&&(nr2||nr3)

Operators – increment and decrement

++ increment by 1

-- decrement by 1

- Discuss n++ and ++n

Statements

- Can be simple or compound
- All simple statements end with “;”
- Compound statements begin with { followed with any number of statements and end with }
- { may be followed by variables declarations, variables that are visible only within that compound statement

Declaration Statement

- A type and a name need to be specified
  int var1;
- All simple statements end with “;”
- If several variables have the same type, their declaration can be combined

  int var1, var2;
- C89 vs C99

Assignment Statement

- Uses = operator
- Before use the variable needs to be declared
- Declaration vs definition

```
int var1;
float var2, var3=3.14;
var1=1;
```

Variable Scope

- Determines the place where the variable has to be declared
  - global variables: accessible in the entire program; declared outside any function
  - local variables: accessible only in the block they were declared