Getting Started with C

Basic information

The C Language

- C developed in the late 1960’s
- ANSI C – American National Standard Institute.
  - Established in 1989.
  - Allowed for portable code that can be transferred from one computer platform to another and still work.

Hello World program

```
/*-------------------------------------------------------------------*/
/* Ruthann Biel         */
/* Lab 1                */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
          printf("\nLab 1 \n\n");
          printf("Hi, Ruthann Biel \n\n");
          return(EXIT_SUCCESS);
}
/*-------------------------------------------------------------------*/
```

The run will look like this:

```
Lab 1

Hi, Ruthann Biel
```

```
/*---------------------------*/
/* Ruthann Biel              */
/* Lab 1                     */
```

==

Examples of comments which can extend over several lines.

- Can be at end of line of code also
  e.g.   printf(“\n”); /* print newline */

- Alternative form:
  printf(“/n”); //print newline

Preprocessor Directives – give the compiler the information it needs to run the program.

==================================

#include <stdio.h>

Stands for "STandarD Input Output"

Needed because we used:

printf

#include <stdlib.h>

Stands for "STandarD LIBrary"

Needed because we used:

EXIT_SUCCESS

int main(void)

must be in program

==

The first module of every C program is called “main”.

Some sources use “void main(void)” with no return.
It does not work with EXIT_SUCCESS, so
we will NOT use this style.

{

…   braces surround BODY of the function

}

==================================

Later we will find additional uses for braces.

```
printf(“\nLab 1 \n\n”);
printf(“Hi, Ruthann Biel \n\n”);
```

==

Examples of the printf function.

Each declaration and statement MUST end with semicolon.

The format string or control string must be enclosed by double quotes.

return EXIT_SUCCESS;

==================================

Shows a successful end of program

It is optional in ANSI C, but it is **not** optional in this class.

```
/*-----------------------------------------------------*/
/* Your name here                                      */
/* Simple computation program                          */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main (void)
{   double x1=1, y1=5, x2=4, y2=7;
    double side_1, side_2, distance;

    printf("\nJane Smith\n\n");
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);
   printf("The distance between the two points "
                “is %5.2f \n\n", distance);
    return EXIT_SUCCESS;
}
/*-----------------------------------------------*/
```

The RUN will look like this:

```
Jane Smith

The distance between the two points is  3.61
```

The nitty-gritty details!

Variable & Identifier Name Rules:

- Must begin with an alphabetic character (a-z, A-Z) or underscore (_).

- Digits are OK but not as first character.

- Can be any length, BUT first 31 characters must be unique.

- C is case sensitive.
  sum, Sum, SuM, and SUM are all different variables.

- A C Reserved Word or Keyword cannot be used as an identifier.

ANSI C Reserved Words:

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

Examples of:

Valid Names	**Invalid Names**
density	2sum
sum3	x&y
x_y	x-y
x2_2	x2.2
Volume	1Volume

TYPES

Each variable must have a “type” which tells us the size, precision, and accuracy the variable will be allowed.
The word “type” is an important buzz word in computers.

NOTE: Min & Max values *VARY* from system to system.

NUMERIC

- Integers
  - short
  - int
  - long
  - unsigned
- Floating point
  - float
  - double
  - long double

CHARACTER

- char
- string

Limits on **athena**

SIGNED INTEGERS:

short minimum: -32768
short maximum: 32767

int minimum: -2147483648
int maximum: 2147483647

long minimum: -2147483648
long maximum: 2147483647

UNSIGNED INTEGERS:

The lower limit for all unsigned integer is zero.
unsigned short maximum: 65535
unsigned int   maximum: 4294967295
unsigned long maximum: 4294967295

More limits on **athena**

```
FLOAT PRECISION:
 float precision digits:  6
 float maximum exponent:  38
 float maximum:  3.402823e+038

 double precision digits: 15
 double maximum exponent: 308
 double maximum: 1.797693e+308

 long double precision: 18
 long double maximum exponent: 4932
 long double maximum: 1.189731e+4932
```

DECLARING VARIABLES:

All variables MUST be declared.

Examples:

```
int  day;
int nickels, dimes;

float x;
float y1, y2;

double degrees;
double a, b, c;
```

ARITHMETIC OPERATORS & USE Intrinsic to C

Addition	**+**	a = b + c;
Subtraction	**–**	c = a – b;
Multiplication	*****	d = x * y;
Division	**/**	x = d / y;
Modulus (or Mod)	**%**	z = f %g;

29 % 5 → 4 which is the *remainder* from the division of 29 & 5.

Shortened Operator and Arithmetic Precedence

Precedence	Operator	Associativity
1	( )	inner-most first
2	unary + -	right to left
3	binary * / %	left to right
4	binary + -	left to right
5	assignment operator =	right to left

Unary involves only one number with the operator.

Ex. -8 -x

Binary involves two numbers with the operator in between.

Ex. 9 + 8 or 9 * -8 or x / y

GETTING VALUES INTO VARIABLES:

```
    int day = 21;                       } declarations
    double y1 = 5.0, y2 = 10.0;         }
```

OR

```
    day = 6;          }
    y1 = 7.2;         }    assignments
    y2 = 7.0;         }
```

GENERAL FORM of Assignment:

```
    variable_name = value;
```

Assignment Statements

Use the equal sign (=) to move a value from the right side to the left side. (Same as in Java)

```
int i = 5;
```

Conceptually it acts like:

```
int i ← 5;
```

Order of Precedence for Numeric Conversion:

Highest precedence:	long double
	double
	float
	long integer
	integer
Lowest precedence:	short integer

Constants

```
int a, b;

a = b + 6:              /* the 6 is a constant integer */


double c, d;

c = d * 2.3;            /* 2.3 is, by default, a double */
```

Mixing Numeric Types

int a = 7, b = 3, c;	
c = a / b;	int / int
now c will have 2	since 7 / 3 is 2 r 1

INTEGER DIVISION TRUNCATES!!!

int a = 7, b = 3;	int / int
float c;	
c = a / b;	so 7 / 3 yields 2 (as an int)
but	then
	converts it to a float, so …
	final value of c is 2.0

```
float a = 7, b = 3, c;
c = a / b;
```

float / float
so 7.0 / 3.0 → 2.333333
final value of c is 2.333333

```
float c;
c = 7 / 3.0;
```

int / double
double takes precedence
acts like 7.0 / 3.0 → 2.333333
final value of c is 2.333333

```
float a = 7.0, b = 3.0;
int c;
c =      a / b;              so 7.0/ 3.0 → 2.333
      float/float            final value of c is 2
```

To force the action we want, use **CASTING**.

```
int a = 7, b = 3;
float c;
c = (float) a / (float) b;
```

Note that the **( )** go around the **type**, not the variable.

General Form for CASTING:

```
        (type) expression
```

Use of Precedence

If we do 2 + 4 * 6 – 3, there are THREE ways it could be done.

2 + 4 * 6 - 3	2 + 4 * 6 – 3	2 + 4 * 6 – 3
2 + 24 - 3	6 * 3	6 * 6 - 3
26 - 3	18	36 - 3
23		33

Using Precedence and no parentheses, C would give you the **first** answer of 23.

To get the second answer of 18, do:

(2 + 4) * (6 -3)

To get the third answer of 33, do:

(((2 + 4) * 6) – 3) **or** (2 + 4) * 6 - 3

Beginning Precedence

Precedence Level	Symbol	Comments	Example
1	( )	Done inner-most first. Then left to right	A + (C * D) + E
2	+ -	Positive & Negative Both unary. Done right to left	-A or +A
3	* / %	Done left to right	A + C * D+ E
4	+ -	Add & Subtract Both binary Done left to right	A + C – D + E

Level 1 is highest. Level 4 is lowest.

Printing in Scientific Notation

X = 157.8926;

Specifer	Value Printer
%.3E	1.579E+002
%.3e	1.579e+002
%.2e	1.58e+002
%g	157.893

General Form:

[sign]d.ddd*e*[sign]ddd

Trigonometric Functions

System defined in math.h

#include <math.h>

sin(x)	x in radians
cos(x)	x in radians
tan(x)	x in radians

asin(x)	arcsine, -1 <= x <= +1
acos(x)	arccosine, -1 <= x <= +1
atan(x)	arctangent

atan2(y, x)	arctangent of y / x

Trigonometric Functions

to convert to radians to degrees, or degrees to radians:

```
#define PI        3.14159
          …
angle_degree = angle_radian * (180/PI);
angle_radian = angle_degree * (PI/180);
```

Math Functions

Intrinsic arithmetic operators (+ - * / %) are part of the **core** of C.

All the extra math functions are stored in:

#include <math.h>

All math functions are type double

fabs(x)	absolute value
sqrt(x)	square root, x >= 0
pow(x,y)	exponentiations, x^y error: if x = 0 & y <= 0 if x <= 0 & y not an integer
ceil(x)	rounds up to next integer
floor(x)	rounds down to previous integer

More Math Functions

All the extra math functions are stored in:

#include <math.h>

All math functions are type double ←

exp(x)	ex (2.718282)
log(x)	ln x, x > 0
log10(x)	log10x, x > 0
abs(x)	absolute value of integer x in **<stdlib.h>**

GETTING THE VARIABLE ON THE SCREEN:

General Form:

```
printf(format_string, argument_list);
```

The format_string has 3 parts:

- characters
- conversion specifiers
- escape sequences

GETTING THE VARIABLE ON THE SCREEN:

Example:

What I want to appear on the screen:

Daniel’s age is 23.

To get it:

```
int age = 23;
printf(“Daniel\’s age is %d. \n”, age);
```

or *same result*

```
printf(“Daniel\’s age is %i. \n”, age);
```

where

%d = conversion specifier

age = list of variables (in this case, list has only one variable in it)

```
printf(“Height is %6.2f \nLength is %6.2f \n”, height, length);
```

on screen:

```
Height is 123.45
Length is   6.27
```

%6.2 f 6 refers to **width** total
2 refers to precision

123.45 = 6 characters printed on the screen

If we changed it to %8.2 f
8 refers to width total
2 refers to precision

bb123.45 = 8 characters printed on the screen

```
int group = 3;
float money = 78.25;
printf(“Group %1i raised $%6.2f.\n”, group, money);
```

Output:

```
Group 3 raised $ 78.25.
```

Print Conversion Specifiers

int, short	%d, %i
short	%hd, %hi
long	%ld, %li
unsigned int	%u
unsigned short	%hu
unsigned long	%lu

More Print Conversion Specifiers

float, double	%f		floating pt.
	%e	%E	scientific
	%g	%G	%e or %f whichever is shorter
long double	%lf		
	%le	%lE	
	%lg	%lG	
character	%c		
string	%s		

Examples of Conversion Specifiers for printf:

(b stands for Blank)

```
int i = 1, j = 29;
float      x = 333.12345678901234567890;
double y = 333.12345678901234567890;
```

format	exp	how printed	why
%d	-j	“-29”	field length 3 by default
%010d	i	“0000000001”	padded with zeros
%-12d	j	“29bbbbbbbbbb“	left adjusted
%12o	j	“bbbbbbbbbb35”	octal/right adjusted
%-12x	j	“1dbbbbbbbbbb“	hex/left adjusted

Examples of Conversion Specifiers for printf :

(b stands for Blank)

```
int i = 1, j = 29;
float       x = 333.12345678901234567890;
double y = 333.12345678901234567890;
```

format	exp	how printed	why
%f	x	“333.123444”	precision 6 by default
%.1f	x	“333.1”	precision 1
%20.3f	x	“bbbbbbbbbbbbb333.123”	right adjusted
%.9f	y	“333.123456789”	precision 9
%-20.3e	y	“3.331e+02bbbbbbbbbbb “	left adjusted

Summary: Conversion Specifiers for printf:

octal	%o
hexadecimal	%x
left adjusted	%-
right adjusted	%+
zero filled	%0

<u>Examples</u>:

<u>Left Adjusted</u>	<u>Right Adjusted</u>
123	123
4	4
67.8	67.8
5678	5678

<u>Zero Filled</u>

int x = 65;

Use a %05d & get: → 00065

Escape Sequences for printf:

\n	Line feed or New Line
\a	Alert. Beep. Bell.
\b	Backspace.
\r	Carriage return. Moves to start of line.
\	Concatenate lines.
\”	Print double quotes.
\f	Formfeed (ejects printer page)
\t	Horizontal tab.
\v	Vertical tab.
\\	Print backslash.
\’	Print single quote.
\?	Print question mark.
%%	Print percent character.

scanf function
(reads values from keyboard)

```
int count;
scanf(“%i”, &count);
```

%i - control string
& - address operator REQUIRED for scanf
count – identifier of variable

scanf function

With two variables:

```
float height, length;
scanf(“%f%f “, &height, &length);
```

scanf function

printf and scanf often appear in pairs:

```
int age;
printf(“\nEnter your age: “);
scanf(“%i”, &age);
```

NOTE;
scanf does **not** like “\n” in the control string.
“\n” is an instruction aimed at the output,
not at input from the keyboard.

scanf is very picky!

1) You MUST use the “&” symbol.

2) You MUST be sure your conversion specifiers AGREE with your variables.

double	**NEEDS %lf**	**(that’s a lower case L)**
int	**NEEDS %i or %d**	
float	**NEEDS %f**	

CONSTANTS

Values that will not change during program.

Constants can be set up in a program using preprocessor directives.

Examples:

```
#define PI  3.14159
#define MONTHS_IN_YEAR  12
```

General Form:

```
#define SYMBOLIC_NAME  replacement
```

***WRONG WAY:

```
#define PI = 3.1415;

#define PI = 3.1415;    // Everything after the name
                        // gets substituted
```

What will happen?

```
        x =  2;
        y = x * PI;
```

It will fill in as:

```
    y = 2 *  =  3.1415;  ;
```

the whole phrase gets substituted!
and will NOT work as written.

ANOTHER WAY to do PI:

Use   #include <math.h>

You may use these constants in my class,
but know that they are NOT ANSI standard!

```
/* Traditional/XOPEN math constants (double precison) */
#ifndef __STRICT_ANSI__
#define M_E             2.7182818284590452354
#define M_LOG2E         1.4426950408889634074
#define M_LOG10E        0.43429448190325182765
#define M_LN2           0.69314718055994530942
#define M_LN10          2.30258509299404568402
#define M_PI            3.14159265358979323846
#define M_PI_2          1.57079632679489661923
#define M_PI_4          0.78539816339744830962
#define M_1_PI          0.31830988618379067154
#define M_2_PI          0.63661977236758134308
#define M_2_SQRTPI      1.12837916709551257390
#define M_SQRT2         1.41421356237309504880
#define M_SQRT1_2       0.70710678118654752440
#endif
```

More Operators: += -= *= /= %=

examples:

```
x = x + 5;        x += 5;

y = y – 7;        y -= 7;

z = z * 9;        z *= 9;

a = a / 13;       a /= 13;

b = b % 15;       b %= 15;
```

Operator and Arithmetic Precedence

Precedence	Operator	Associativity
1	( )	inner-most first
2	+ - Unary	right to left
3	+ - Unary Postfix	left to right
4	+ - Unary Prefix	right to left
5	* / % Binary	left to right
6	+ - Binary	left to right
7	= Assignment Operator	right to left

Operator and Arithmetic Precedence

Unary involves only one number with the operator.

Ex. -8 -x y++

Binary involves two numbers with the operator in between.

Ex. 9 + 8 or 9 * -8

More on PRINTF Statements:

```
int a = 5, b= 9;
printf (“%i%i\n\n”, a, b);
```

OUTPUT:

```
59
```

COMMENTS:

Oops, the numbers are bumped right against each other.

More on PRINTF Statements:

```
int a = 5, b= 9;
printf (“%ibb%i\n\n”, a, b);
```

OUTPUT:

```
5bb9
```

Try it again:

b - represents a blank

```
int a = 5, b= 9;
printf (“%2i%2i\n\n”, a, b);
```

OUTPUT:

```
 5 9
b5b9    /* this line showing where the blanks are */
```

COMMENTS:

Now the numbers have space between them.

Try it again:

b - represents a blank

```
int a = 5, b= 9;
printf ("%3i%3i\n\n", a, b);
```

OUTPUT:

```
  5  9
bb5bb9          /* this line showing where the
                   blanks are */
```

COMMENTS:

Now the numbers have more space between them.

Another Problem:

```
int a = 5, b = 9;
int c = 223, d = 123;

printf ("%2i%2i\n\n", a, b);
printf ("%2i%2i\n\n", c, d);
```

OUTPUT:

```
 5 9

223123
```

COMMENTS:

Not enough room for the three-digit numbers.
Also the numbers do not line up under each other.

Try it again:

b - represents a blank

```
int a = 5, b = 9;
int c = 223, d = 123;

printf ("%4i%4i\n\n", a, b);
printf ("%4i%4i\n\n", c, d);
```

OUTPUT:

```
bbb5bbb9
b223b123
```

COMMENTS:

Now the numbers print with space between them.
Also the numbers now line up under each other.

The C Language

- Currently, the most commonly-used language for embedded systems
- "High-level assembly"
- Very portable: compilers exist for

  virtually every processor
- Easy-to-understand compilation
- Produces efficient code
- Fairly concise

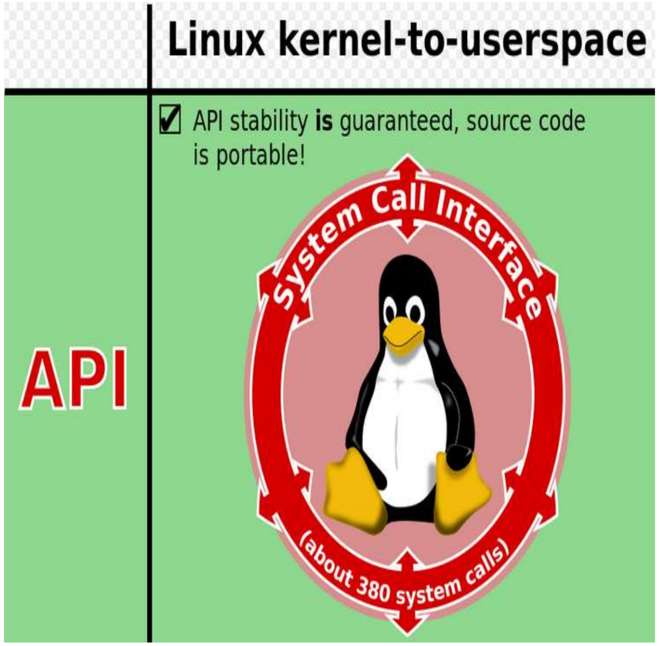
Source: **Embedded Systems Programming Languages**
(http://www.eetimes.com/author.asp?section_id=36&doc_id=1323907
9/12/2014)

What is API?

Application Program Interface (**API**) is a set of routines, protocols, and tools for building software applications.

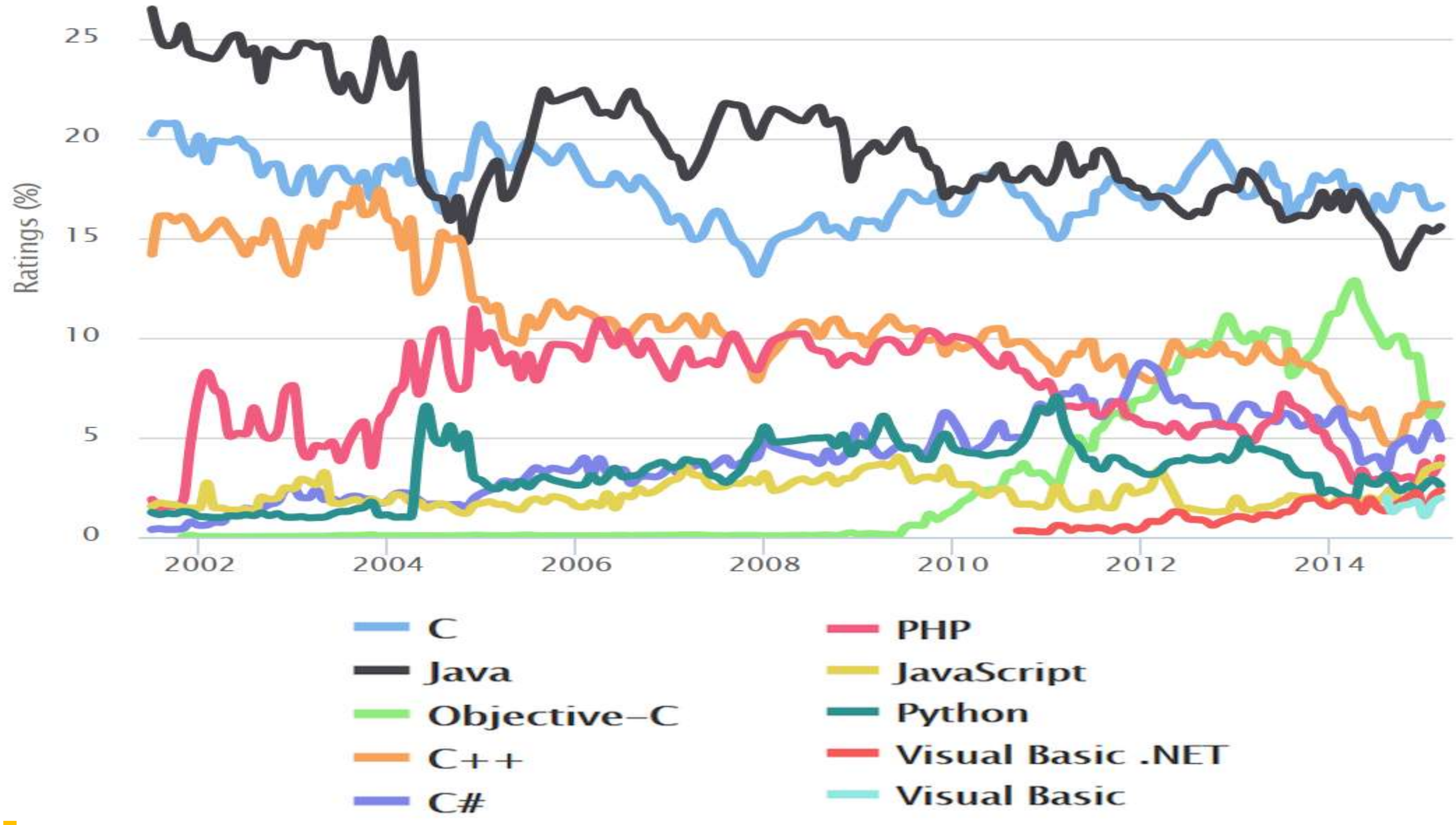
An **API** specifies how software components should interact and **APIs** are used when programming graphical user interface (GUI) components.

Why *C*?

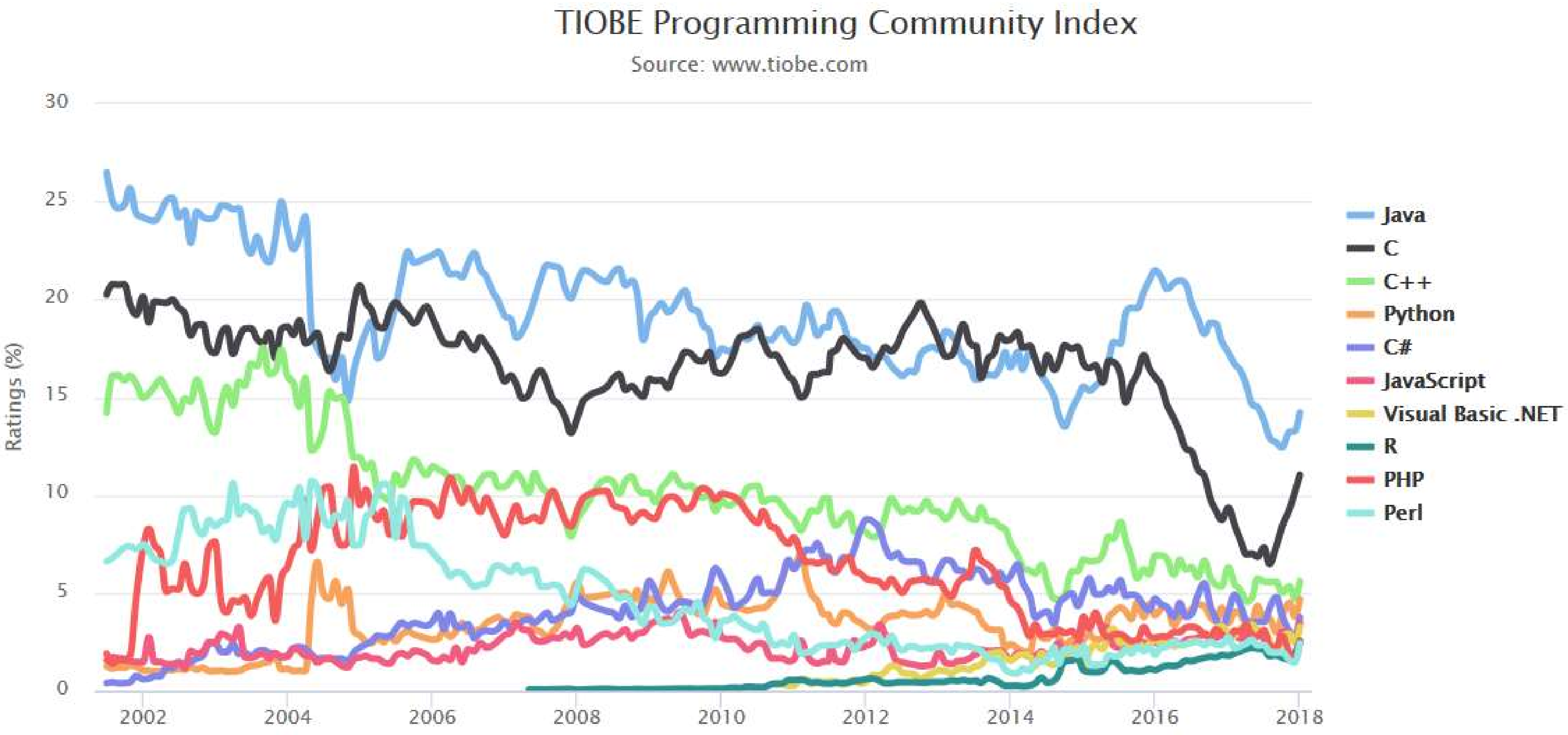

Linux kernel-to-userspace
☑ API stability **is** guaranteed, source code is portable!
API
System Call Interface
(about 380 system calls)

- Many situations where it is *only* language or system available
  - Small, embedded systems, instrumentation, etc.
- Many "low-level" situations that don't have support for "high-level" languages
  - Operating systems, real-time systems, drivers
- On Unix-like systems, that API is usually part of an implementation of the C library (libc), such as glibc, that provides wrapper functions for the system calls.

Languages Popularity



Source: https://en.wikipedia.org/wiki/TIOBE_index


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Java
C
C++
Python
C#
JavaScript
Visual Basic .NET
R
PHP
Perl

Getting Started

Basic information

THE END