

Úvod do procedurálneho programovania

Základy procedurálneho programovania 1, 2021

Ing. Marek Galinski, PhD.

Algorithmus

Algoritmus

- Postupnosť presne definovaných krokov určujúcich poradie vykonávania konečného počtu elementárnych operácií, ktoré zabezpečujú vyriešenie úloh toho istého typu ^[1]
- Algoritmy tu boli skôr ako informatika
 - Muhammad ibn **Músá Al-Chwárizmí** –perský matematik a astronóm (9. storočie)
 - Autor traktátu z r. 825 preloženého v 1145 do latinčiny Algorithmi de numero indorum – európska veda sa zoznámila s indickou pozičnou sústavou a číslom 0.
 - Meno Al-Chwárizmí bolo latinizované na **Al-Gorizmí**, neskôr **Algoritmí** – základ slova algoritmus ^[2]

[1] <http://kifri.fri.uniza.sk/~bene/vyuka/zi/predn/algoritmus.html>

[2] Gabriela Grmanová, Základy procedurálneho programovania 1, úvodná prednáška

Požadované vlastnosti algoritmů

Vlastnosti algoritmov

- **Jednoznačnosť** – každý krok musí byť presne definovaný
- **Konečnosť** – výpočet skončí po konečnom počte krokov
- **Rezultatívnosť** – po konečnom počte krokov musíme dostať výsledok
- **Správnosť** – výsledok algoritmu je vždy korektný

- **Efektívnosť** – výpočtový čas a priestor by mali byť čo najmenšie
 - Toto však často ide jedno proti druhému, treba si určiť priority

- **Všeobecnosť** – algoritmus nerieši jeden konkrétny problém (napr. $5+5$), rieši všeobecnú triedu problémov (napr. $a+b$)

algorithmus \neq program

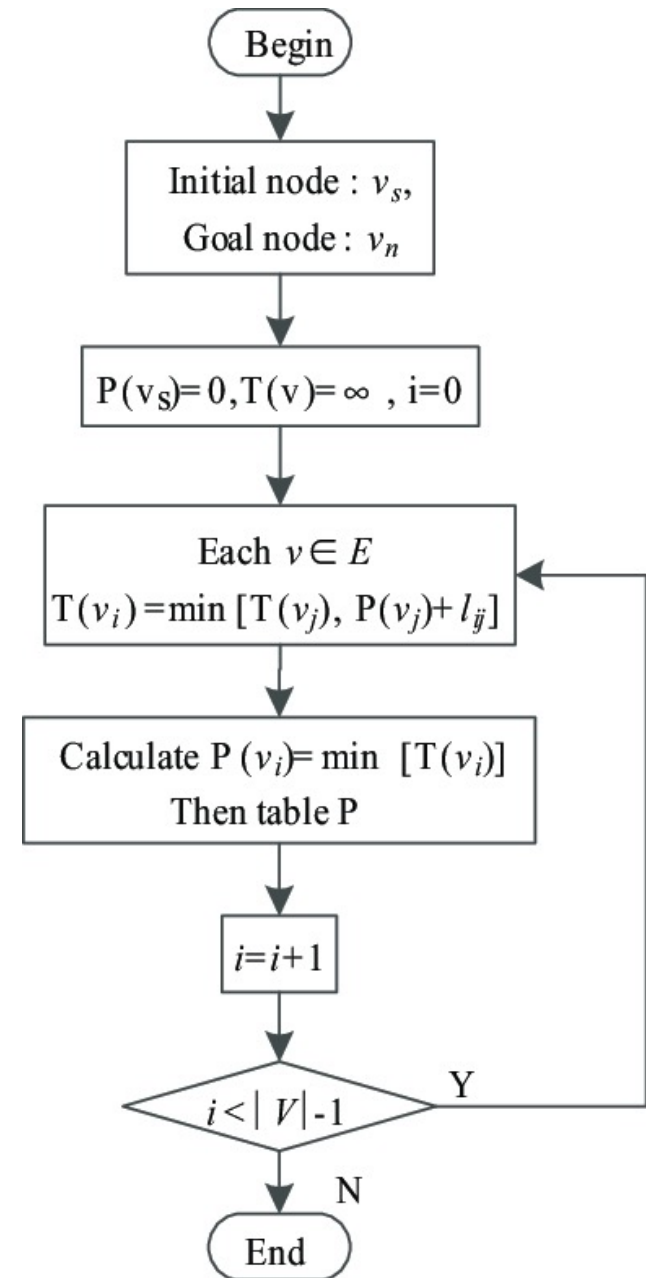
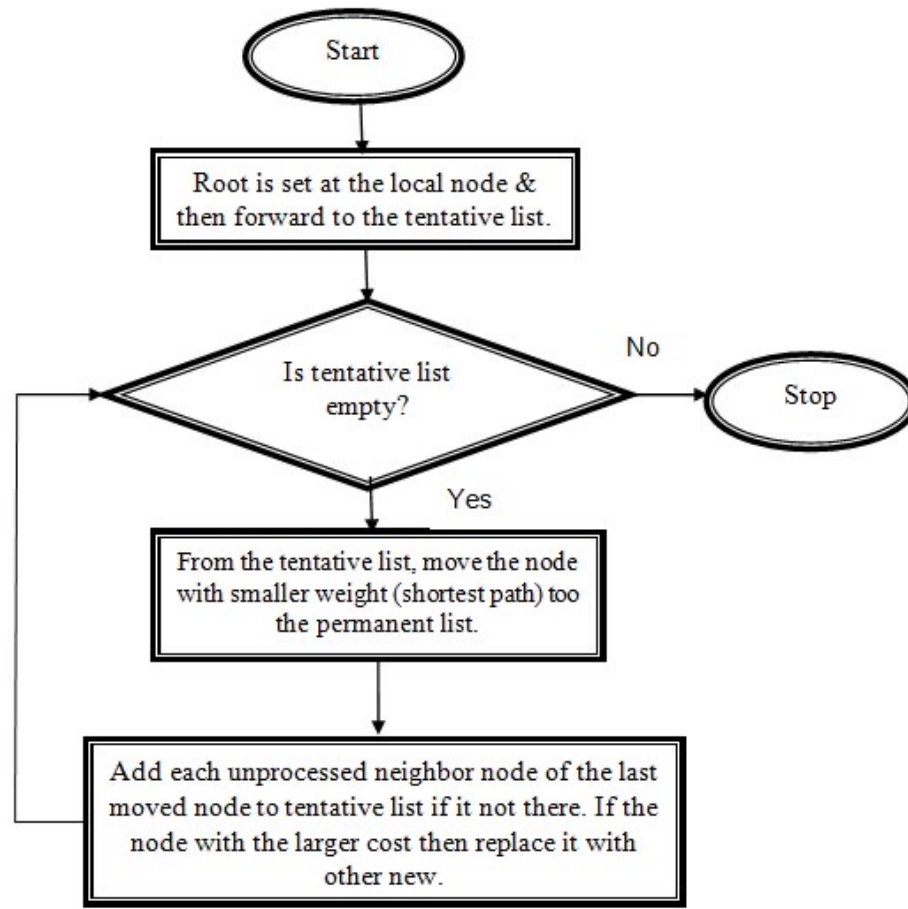
Zápis algoritmu

- Text – postupnosť krokov

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.^[14]
3. For the current node, consider all of its unvisited neighbours and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbour *B* has length 2, then the distance to *B* through *A* will be $6 + 2 = 8$. If *B* was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Zápis algoritmu

- Vývojový diagram



Zápis algoritmu

- **Pseudokód**

k = number of paths to find, n = paths found so far,
 s = source node, t = destination node,
 $G[i,j]$ = network connectivity matrix,
 $C[i,j]$ = network capacity matrix,
 $H[u]$ = cost of a node,
 Inf = a constant larger than the greatest possible path length

1. Initialize $G[i,j]$ and $C[i,j]$ with network values
2. Remove the span between s and t , to emulate a failure
3. $G[s,t] = inf$, $C[s,t] = 0$
4. Call k -Dijkstra (k, n, s, t, G, C)
5. k -Dijkstra (k, n, s, t, G, C) {
6. while ($n < k$) {
7. Dijkstra (s, t, G, C)
8. Record the path by tracing the predecessor vector
9. Subtract every entry in $G[i,j]$ that appears in the path
10. $n++$
11. }
12. }

Zápis algoritmu

- Kód v konkrétnom jazyku

```
from heapq import heappush, heappop
# based on recipe 119466
def dijkstra_shortest_path(graph, source):
    distances = {}
    predecessors = {}
    seen = {source: 0}
    priority_queue = [(0, source)]

    while priority_queue:
        v_dist, v = heappop(priority_queue)
        distances[v] = v_dist

        for w in graph[v]:
            vw_dist = distances[v] + 1
            if w not in seen or vw_dist < seen[w]:
                seen[w] = vw_dist
                heappush(priority_queue, (vw_dist, w))
                predecessors[w] = v

    return distances, predecessors
```

algorithmus \neq program

**Počítačový program je už konkrétnou
reprezentáciou algoritmu v zvolenom
programovacom jazyku**

Procedurálne programovanie

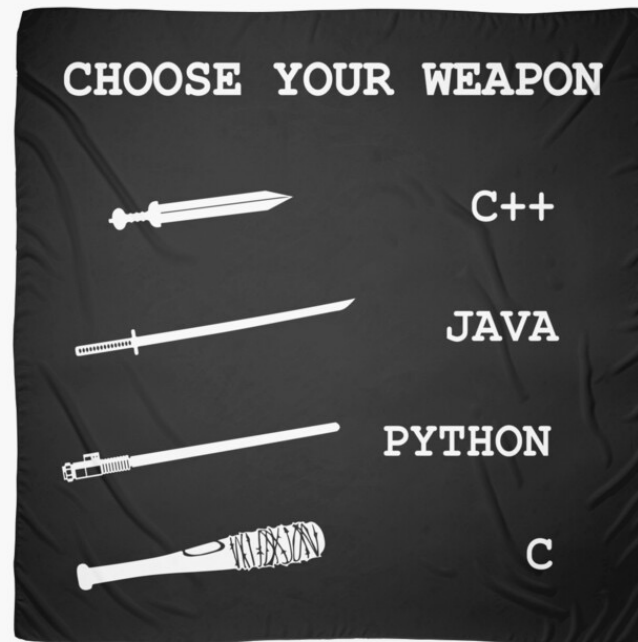
Procedurálne programovanie

- **Základom je volanie “procedúr” – funkcii, subrutín, ...**
 - Funkcia obsahuje nejakú postupnosť krokov, je volaná počas behu programu inými funkciami alebo aj sama sebou
- Používajú sa rôzne riadiace štruktúry – cykly, vetvenia (podmienky)
- Samotný program je koncipovaný ako postupnosť príkazov, ktoré sa vykonávajú v poradí, v akom sú zapísané (ak nejaka riadiaca procedúra neurčí inak)
- **Jazyky: C, Pascal, Cobol**
 - Nie C++, nie C#, nie Objective-C

Procedurálne programovanie

- **Operácie sa vykonávajú nad dátami**
- Dáta sú uložené v pamäťových miestach v podobe premenných
 - Dáta môžu byť načítavané zo vstupných zariadení, zo súborov, atď ...
 - Počas behu program sa tieto data spravidla menia

Úvod do jazyka C



O vzniku C

- Prvý standard – 1978 – “Kernighan a Ritchie: The C Programming Language” v Bell Laboratories



Dennis MacAlistair Ritchie

1941 – 2011

Autor jazyka C

Autor OS Unix

PhD @ Harvard University 1968

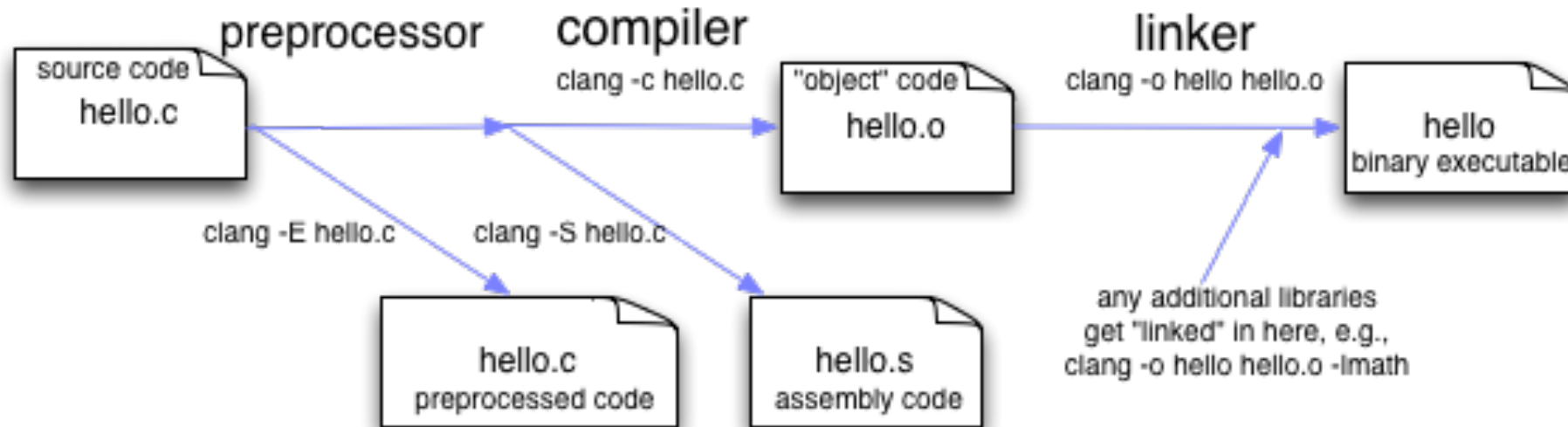


NOKIA Bell Labs

O vzniku C

- **C je univerzálny jazyk nízkej úrovne**
- Pre veľa úloh je dodnes v princípe najefektívnejším jazykom
- Jednoduché stavebné prvky
- Navrhnutý a implementovaný pod OS Unix
- Pracuje iba so základnými dátovými typmi (znak, celé číslo, reálne číslo, ...)
 - Tieto dátové typy sa následne dajú rozširovať
- Prenosný strojový jazyk, 100% prenositeľnosť (ANSI standard)

Spracovanie program v jazyku C



Spracovanie program v jazyku C

- Pôvodný kód

```

1 //
2 //  main.c
3 //  testc2
4 //
5 //  Created by Marek Galinski on 19/09/2020.
6 //  Copyright © 2020 Marek Galinski. All rights reserved.
7 //
8
9 #include <stdio.h>
10
11 int main(int argc, const char * argv[]) {
12     printf("Zjavne sa mi to kompiluje spravne – funguje to!\n");
13     return 0;
14 }
```

Spracovanie program v jazyku C

- Predspracovaný kód

```

1 # 1 "/Users/marek/projects/testc2/testc2/main.c"
2 # 1 "<built-in>" 1
3 # 1 "<built-in>" 3
4 # 362 "<built-in>" 3
5 # 1 "<command line>" 1
6 # 1 "<built-in>" 2
7 # 1 "/Users/marek/projects/testc2/testc2/main.c" 2
8
9
10
11
12
13
14
15
16 #pragma clang module import Darwin.C.stdio /* clang -E: implicit
17
18 int main(int argc, const char * argv[]) {
19     printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");
20     return 0;
21 }

```

Spracovanie program v jazyku C

- Kód v jazyku symbolických inštrukcií

(strojový kód, assembler)

```

1  .section    __TEXT,__text,regular,pure_instructions
2  .build_version macos, 10, 15    sdk_version 10, 14
3  .file      1 "/Users/marek/projects/testc2" "/Users/marek/project
4  .globl     _main                  ## -- Begin function main
5  .p2align   4, 0x90
6  _main:                                         ## @main
7  Lfunc_begin0:
8  .loc       1 11 0                      ## /Users/marek/projects/testc
9  .cfi_startproc
10 ## %bb.0:
11     pushq   %rbp
12     .cfi_def_cfa_offset 16
13     .cfi_offset %rbp, -16
14     movq    %rsp, %rbp
15     .cfi_def_cfa_register %rbp
16     subq    $32, %rsp
17     movl    $0, -4(%rbp)
18     movl    %edi, -8(%rbp)
19     movq    %rsi, -16(%rbp)
20  Ltmp0:
21     .loc     1 12 5 prologue_end          ## /Users/marek/projects/testc
22     leaq    L_.str(%rip), %rdi

```

Spracovanie program v jazyku C

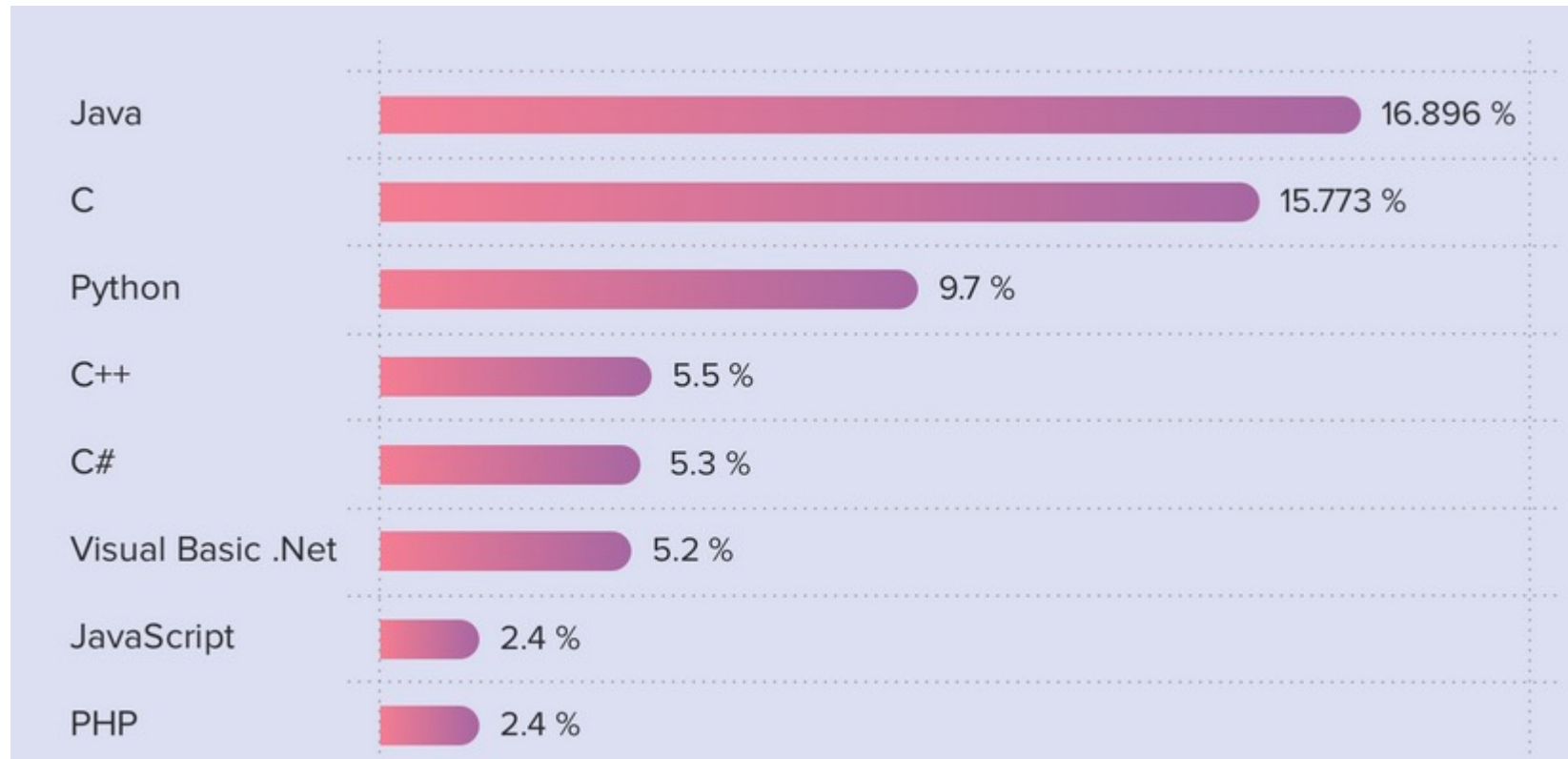
- **Compiler vs. Linker**
- **Po preklade** - Program v relatívnom tvare je prekladaný so štartovacou adresou 0, nemá nastavené žiadne absolútne adresy, **ešte nemá doriešené adresovania na premenné a volania funkcií** náväzných modulov programu.
- **Po linkovaní** - spája všetky relatívne moduly **do jedného uceleného programu** z knižničných súborov (.lib), ktoré obsahujú ďalšie relatívne moduly, pripája požadované moduly ... Týmto vzťahom vzájomnej komunikácie modulov hovoríme krížové referencie. Výsledkom činnosti linkera je **spustiteľný program** rovnakého názvu s príponou (.exe)

**Chcem sa v roku 2020 učiť 50
rokov starý jazyk C?**

Popularita C

- **TIOBE Index, 2020**

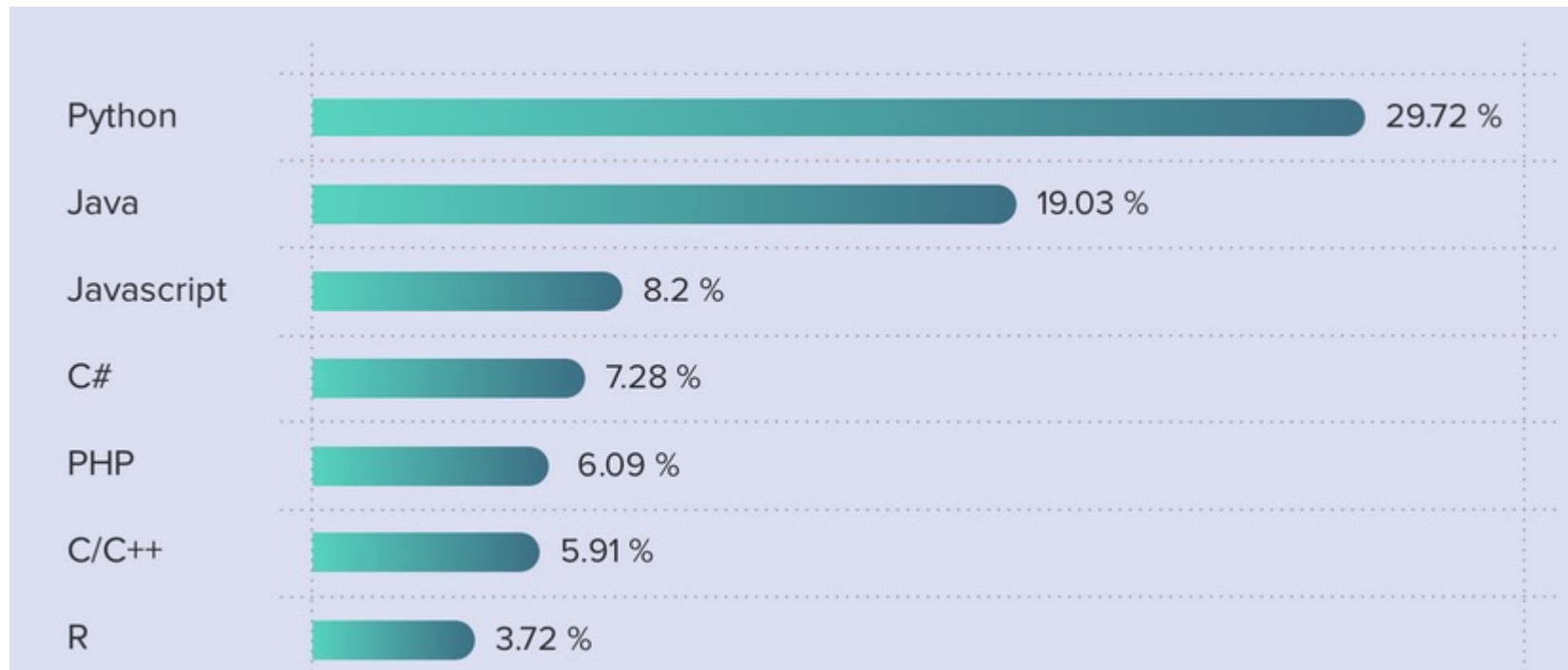
- TIOBE measures the sheer quantity of 25 search engine hits – including popular Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube, and Baidu.



Popularita C

- **PYPL Index, 2020**

- PYPL – how often language tutorials are googled by exploring Google Trends.



A čo iné jazyky?

- **Python** – interpretovaný do C
- **PHP** – interpretované do C
- **JavaScript** – trochu zložitejšie, ale na konci C++

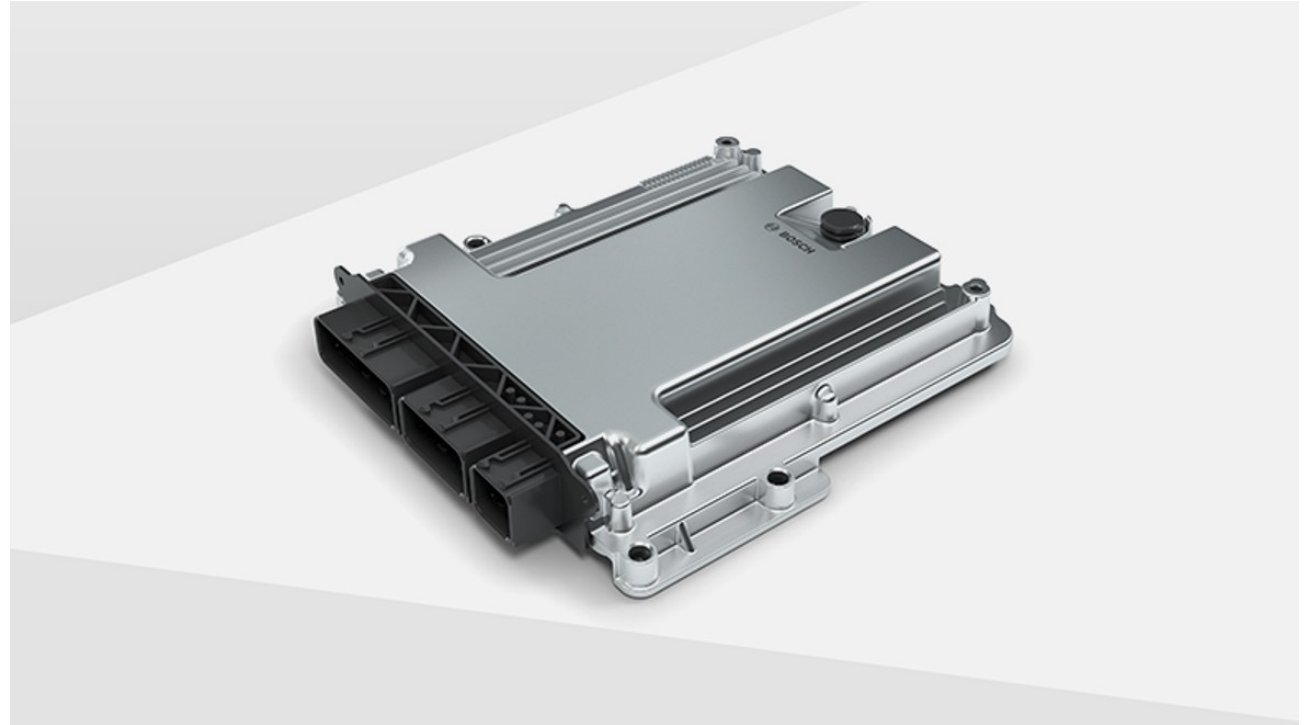
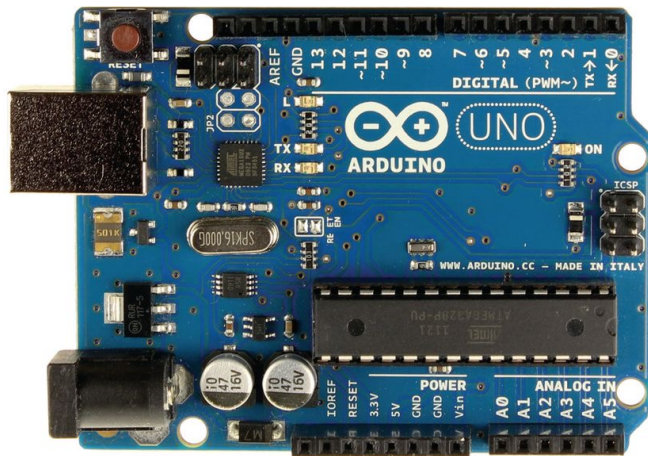
Využitie C

- Prostredia s extrémnym dôrazom na spoľahlivosť kódu



Využitie C

- Microcontrollery, IoT, High-performance



Prečo C?

- **Niekoľko code-safety štandardov, spravidla určených pre C**
 - **Certifikácia, homologizácia**
- MISRA C
- AUTOSAR – cituje MISRA C
- ISO26262 – cituje MISRA C
- ASIL – cituje ISO26262
- NASA Jet Propulsion Laboratory – cituje MISRA C

A napriek tomu ...

- Ariane 5's first test flight on 4 June 1996 **failed**, with the rocket self-destructing **37 seconds after launch** because of a malfunction in the control software.
- A data conversion from **64-bit floating point value to 16-bit signed integer value** to be stored in a variable representing horizontal bias caused a processor trap (operand error) **because the floating point value was too large to be represented by a 16-bit signed integer**.



A napriek tomu ...

- Ariane 5's first test flight on 4 June 1996 **failed**, with the rocket self-destructing **67 seconds after launch** because of a malfunction in the control software.
- A data conversion from 32-bit floating point value to 16-bit signed integer value to be stored in a variable representing horizontal time caused a processor trap (operand error) **because the floating point value was too large to be represented by a 16-bit signed integer.**



Základné prvky jazyka C

Hello world!

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");
    return 0;
}
```

Hello world!

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

```
int main(int argc, const char * argv[]) {  
    printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");  
    return 0;  
}
```

Odkaz na hlavičkový súbor

Hello world!

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");
    return 0;
}
```

Hlavná funkcia – main()

Hello world!

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");
    return 0;
}
```

Parametre funkcie

Hello world!

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");
    return 0;
}
```

Samotný aplikačný kód funkcie

Hello world!

```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    printf("Zjavne sa mi to kompiluje spravne - funguje to!\n");
    return 0;
}
```

Návratová hodnota funkcie

Hello world!

- **Funkcia `main()`**
 - Vždy musí byť v programe, je volaná ako prvá pri spustení program
 - Má spravidla celočíselnú návratovú hodnotu (ale nie je to nutné)
 - Nemusí mať argumenty, ale môže mať
 - Tu nám vyplýva, že program v C pozostávajú z funkcií (min. 1)

Hello world!

- **Komentáre v jazyku C**

- Komentáre sú dôležité – sprehl'adňujú a popisujú kód, aby sa v ňom vyznal niekto iný, ale aj programátor sám.

```
12 printf("Zjavne sa mi to kompiluje s
13 //toto je jednoriadkovy komentar
14 return 0;
```

```
12 printf("Zjavne sa mi to kompil
13 /*toto funguje tiez */
14 return 0;
```

```
12 printf("Zjavne sa mi to kom
13 /*toto je
14 komentar na viac
15 riadkov */
16 return 0;
```

- **Takto nie**

```
12 printf("Zjavne sa mi to kompiluje
13 /*toto /* je uplna */ blbost */
14 return 0;
```


Premenné a dátové typy

- **Premenné**
 - **Pomenované pamäťové miesto kde je možné uložiť hodnotu**
 - Hodnoty môžu byť rôznych typov – číslo, znak, reťazec ...
 - C je typový jazyk, je nutné určiť typ premennej vopred

Premenné a dátové typy

- **Dátové typy**

- `int` – celé číslo
- `long int (long)` – veľké celé číslo
- `short int (short)` – malé celé číslo
- `char` – znak, ASCII hodnota 0-255
- `float` – reálne číslo
- `double` – väčšie / presnejšie reálne číslo (20 desatinných miest)
- `long double` – ešte väčšie reálne číslo

Premenné a dátové typy

- Veľkosť premennej

`sizeof(int)`

- Vyjadruje sa v jednotke Byte (B)
- 1 Byte (B) = 8 bitov (b)
- 1 Byte = 256 možných hodnôt, lebo 2^8

Premenné a dátové typy

- V jazyku C platí, že:
 - `sizeof(char) = 1 Byte`
 - `sizeof(short int) <= sizeof(int) <= sizeof(long int)`
 - `sizeof(unsigned int) = sizeof(signed int)`
 - `sizeof(float) <= sizeof(double) <= sizeof(long double)`

Premenné a dátové typy

- V jazyku C platí, že:
 - C neposkytuje typ `boolean` - booleove hodnoty sa reprezentujú pomocou typu `int`:
 - **FALSE**: 0
 - **TRUE**: nenulová hodnota (najčastejšie 1)

Premenné a dátové typy

- V jazyku C platí, že:
 - C je **case-sensitive** – premenna, Premenna, PREMENNA sú tri rôzne identifikátory premenných
 - Kľúčové slová sa píšú vždy malými písmenami – if, for, ...
- Používanie podčiarkovníka _
 - `_pom` – systémový identifikátor, nepoužívať
 - `Pom_x` – môže sa používať
 - `pom_` - nepoužíva sa, ľahko sa prehliadne

Premenné a dátové typy

- V jazyku C platí, že:
 - **Definícia premennej** – príkaz, ktorý priradí premennej určitého typu meno a pamäť.
 - Definície premenných
 - `int i;` – premenná `i` typu `int`
 - `char c, ch;` – premenné `c` a `ch` typu `char`
 - `double f,g;` - premenné `f` a `g` typu `double`

Premenné a dátové typy

- **Priradenie premennej**

- **l-hodnota** – predstavuje adresu, kam je možné priradiť hodnotu

- **Príklad**

- premenná **i** je l-hodnotou
 - konštanta **123** nie je l-hodnotou
 - výraz **x+3** nie je l-hodnotou

- Výraz má svoju hodnotu
- Priradenie hodnoty do premennej
- Príkaz ukončený bodkočiarkou

`i * 2 + 3`

`j = i * 2 + 3`

`j = i * 2 + 3;`

Premenné a dátové typy

- Priradenie premennej

- **l-hodnota** – predstavuje adresu, kam je možné priradiť hodnotu

- Príklad

- premenná **i** je l-hodnotou
 - konštanta **123** nie je l-hodnotou
 - výraz **x+3** nie je l-hodnotou

- Výraz má svoju hodnotu
- Priradenie hodnoty do premennej
- Príkaz ukončený bodkočiarkou

```

i * 2 + 3
j = i * 2 + 3
j = i * 2 + 3;

```

Premenné a dátové typy

- Priradenie premennej

- Do premennej naľavo sa priradí hodnota pravej strany
- Nie je to rovnica
- Nie je to porovnanie
- Priradiť hodnotu sa dá aj priamo v definícii
- Viacnásobné priradenie `k = j = i = 2;`

```
int i;

i = 5;

i = i + 1;

i = i * 5 - 20;
```

```
int i = 5;

i = i + 1;

i = i * 5 - 20;
```

Premenné a dátové typy

- Priradenie premennej
 - Ak pracujeme s neinicializovanou premennou (nepriradili sme do nej žiadnu hodnotu), program nevypíše chybu pri kompilovaní, nespadne – ale nemusí pracovať správne

Aritmetické výrazy

- **Aritmetický výraz (obyčajne s priradením) ukončený bodkočiarkou sa stáva príkazom**
 - `i=2` výraz s priradením
 - `i=2;` príkaz
- **Operátory:**
 - Unárne
 - Binárne
 - Špeciálne unárne
 - Prirad'ovacie

Aritmetické výrazy

- **Unárne operatory**

- Plus (+)
- Mínus (-)

```
...  
x = +5;  
y = -7;  
...
```

Aritmetické výrazy

• Binárne operátory

- Sčítanie (+)
- Odčítanie (-)
- Násobenie (*)
- Reálne delenie (/) – závisí od typov operandov
- Celočíselné delenie (/) – závisí od typov operandov
- Modulo (%)

int / int	→ celočíselné
int / float	→ reálne
float / int	→ reálne
float / float	→ reálne

```
s = a * b;
```

Aritmetické výrazy

- **Priorita vyhodnocovania aritmetických výrazov**

Operátor(y)	Operácia(e)	Priorita
()	zátvorky	Vyhodnotené ako prvé Vnorené zátvorky – najvnútornejšia najskôr Na rovnakej úrovni – zľava doprava
* / %	Násobenie, delenie, zvyšok po delení	Vyhodnotené ako druhé Na rovnakej úrovni – zľava doprava
+ -	Pripočítanie, odpočítanie	Vyhodnotené ako tretie Na rovnakej úrovni – zľava doprava
=	priradenie	Vyhodnotené ako posledné

Aritmetické výrazy

- Priorita vyhodnocovania aritmetických výrazov

C: $p = a + b + c + d / 4;$

Algebra: $p = a + b + c + \frac{d}{4}$

Algebra: $z = p \cdot r \% q + \frac{w}{x} - y$

C: $p = p * r \% q + w / x - y$

Poradie: 6 1 2 4 3 5

Aritmetické výrazy

- Priorita vyhodnocovania aritmetických výrazov

C: $p = a + b + c + d / 4;$

Algebra: $p = a + b + c + \frac{d}{4}$

Algebra: $z = p \cdot r \% q + \frac{w}{x} - y$

C: $p = p * r \% q + w / x - y$

Poradie: 6 1 2 4 3 5

Operátory s rovnakou prioritou sa vyhodnocujú zľava doprava

Terminálový vstup a výstup

- **Vstupno/výstupné operácie nie sú súčasťou jazyka C, rieši to štandardná knižnica**
 - Najviac strojovo závislých akcií je práve vstupno/výstupných, oddelujú sa takto nezávislé a strojovo závislé časti jazyka

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

Terminálový vstup a výstup

- **Formátovaný vstup a výstup**
 - Funkcie, ktoré načítajú na vstupe reťazec číslíc a uložia ich ako číslo
 - Funkcie, ktoré konvertujú číselnú hodnotu do reťazca číslíc

`scanf("...", ...)`

`printf("...", ...)`

Terminálový vstup a výstup

- **Formátovaný vstup a výstup**
 - **formátovací reťazec obsahuje:**
 - formátovacie špecifikácie - začínajú znakom % a určujú formát vstupu alebo výstupu
 - znakové postupnosti - nezačínajú % a vypíšu sa tak, ako sú napísané (používajú sa len v printf())
 - **počet parametrov musí súhlasiť s formátovacou špecifikáciou**
 - počet % = počtu ďalších parametrov
 - ak počty nesúhlasia, kompilátor nehlási chybu, ale program sa nesprávne správa

Terminálový vstup a výstup

- Formátovaný vstup a výstup

```
printf("Zadajte strany obdlznika: ");
scanf("%f %f", &a, &b);
printf("Stvorec so stranami: %f a %f ma: \n", a, b);
o = 2 * a + 2 * b;
s = a * b;
printf("- obvod %f\n- obsah %f\n", o, s);
```

Terminálový vstup a výstup

- **Formátovacie znaky**

c	znak
d	desiatkové číslo typu signed int
ld	desiatkové číslo typu signed long
u	desiatkové číslo typu unsigned int
lu	desiatkové číslo typu unsigned long
f	float (pre printf() tiež double)
Lf	long double
lf	double
x	hexadecimálne číslo malými písmenami
X	hexadecimálne číslo veľkými písmenami
o	osmičkové číslo
g	general
s	reťazec

Príklady

Príklady

- **Zdrojové kódy príkladov sú dostupné aj na dokumentovom serveri v AIS**
 - Priečink **Príklady k prednáškam**
 - `p01%_hodiny.c`
 - `p01%_obdlznik.c`
 - `p01%_priemer.c`

Q?