Úvod do procedurálneho programovania

Základy procedurálneho programovania 1, 2022

Ing. Marek Galinski, PhD.

Algoritmus

2

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

3

Požadované vlastnosti algoritmov

4

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 problem (napr. 5+5), rieši všeobecnú triedu problémov (napr. a+b)

5

algoritmus ≠ program

6

Zápis algoritmu

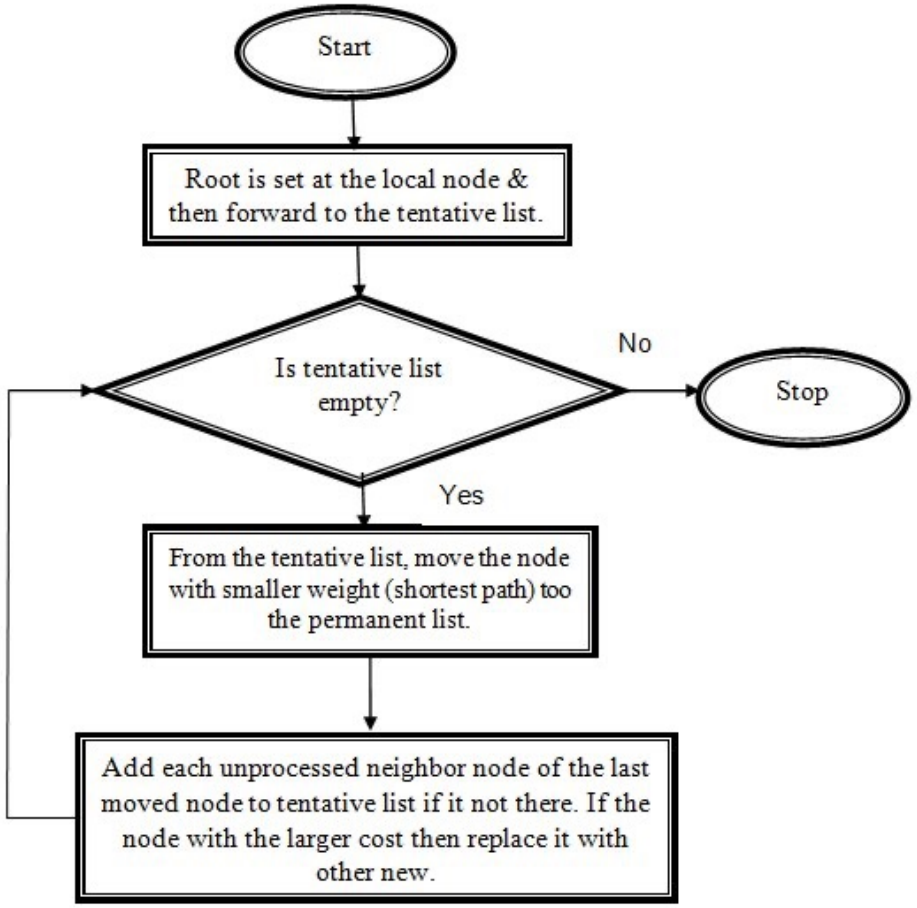
- **Text – postupnosť krokov**

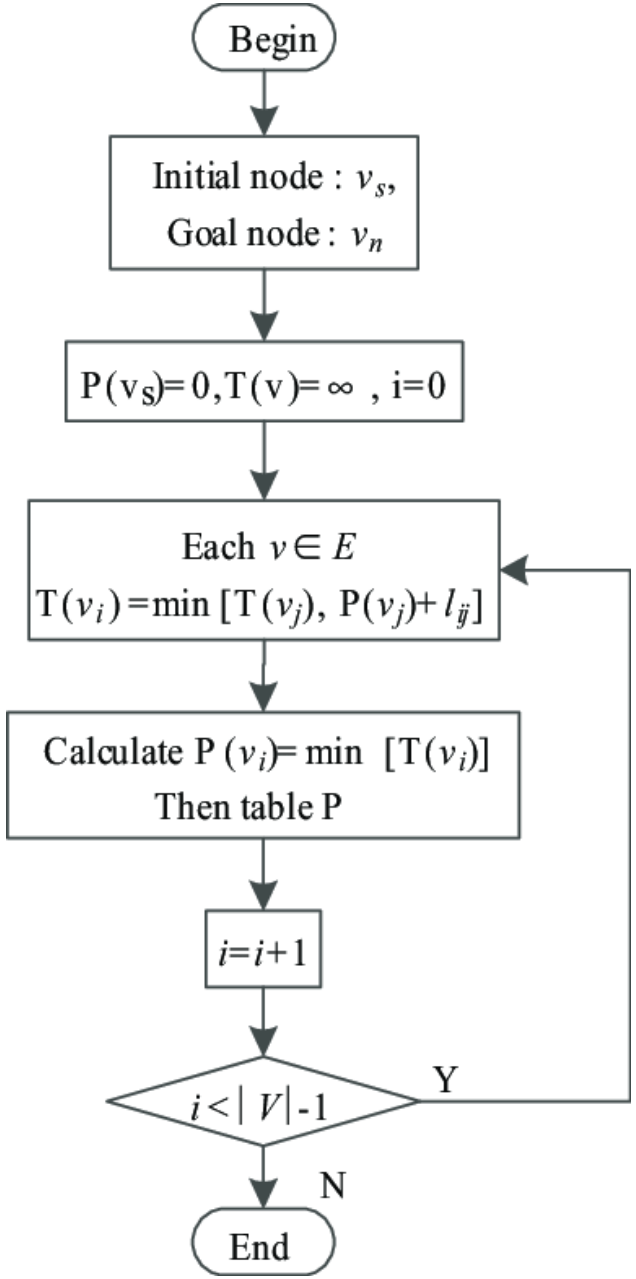
1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.[14]
3. For the current node, consider all of its 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.

7

Zápis algoritmu

- **Vývojový diagram**





https://www.researchgate.net/publication/327982778/figure/fig1/AS:676900049543171@1538396956393/Flowchart-of-the-Dijkstra-algorithm.png
https://www.researchgate.net/profile/Amandeep_Singh69/publication/303741658/figure/fig2/AS:370736045805573@1465401769518/Flow-diagram-of-Dijkstra-algorithm_Q640.jpg

8

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. }

https://www.researchgate.net/profile/T_Sanguankotchakorn/publication/232636583/figure/fig2/AS:667635717009428@1536188167917/The-Pseudo-Code-for-the-k-Shortest-Path-Dijkstra-Algorithm.png

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
```

https://image.slidesharecdn.com/csnap-120714113205-phpapp01/95/complex-and-social-network-analysis-in-python-36-728.jpg?cb=1342270917

10

algoritmus ≠ program

Počítačový program je už konkrétnou reprezentáciou algoritmu v zvolenom programovacom jazyku

11

Procedurálne programovanie

12

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 štrukturý – 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

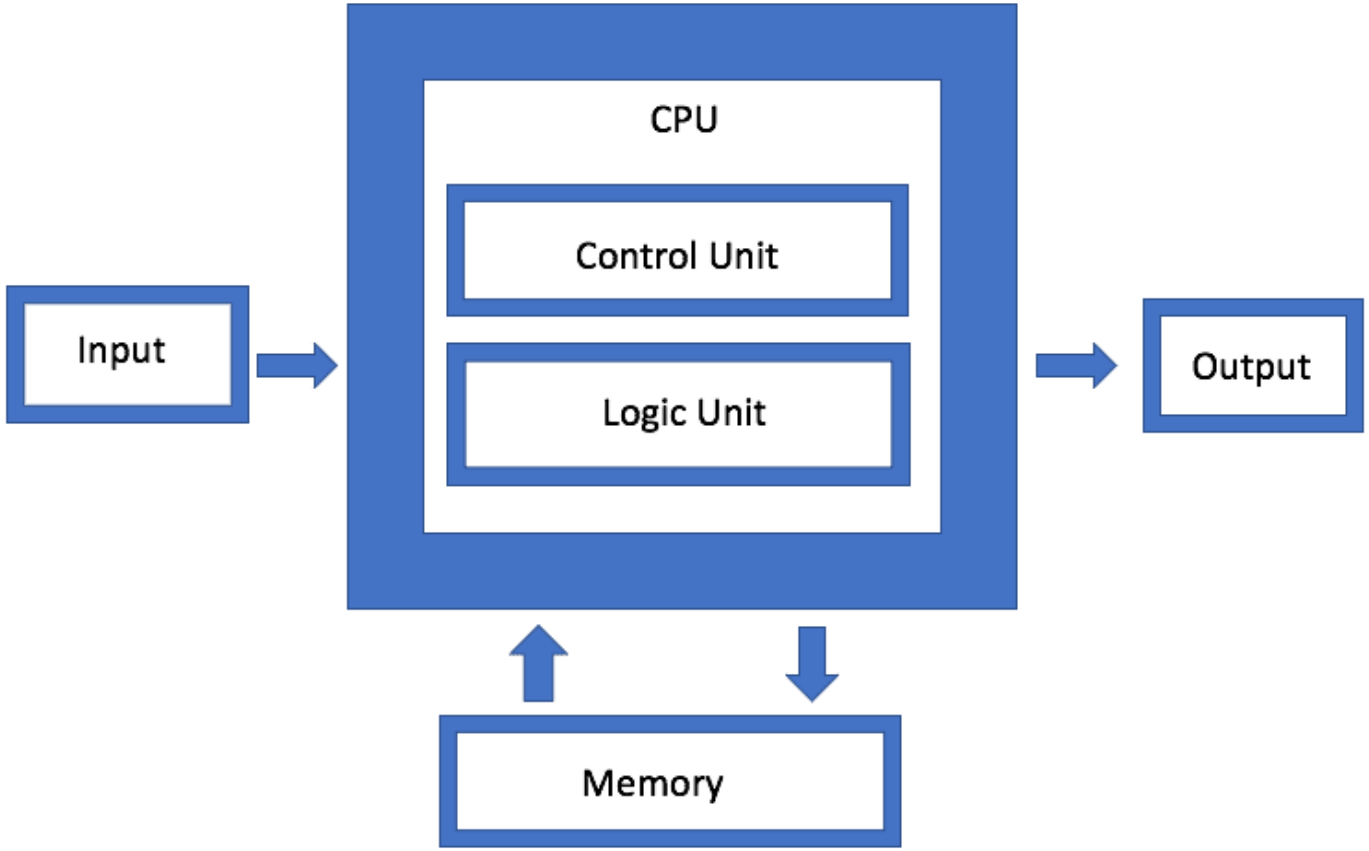
13

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

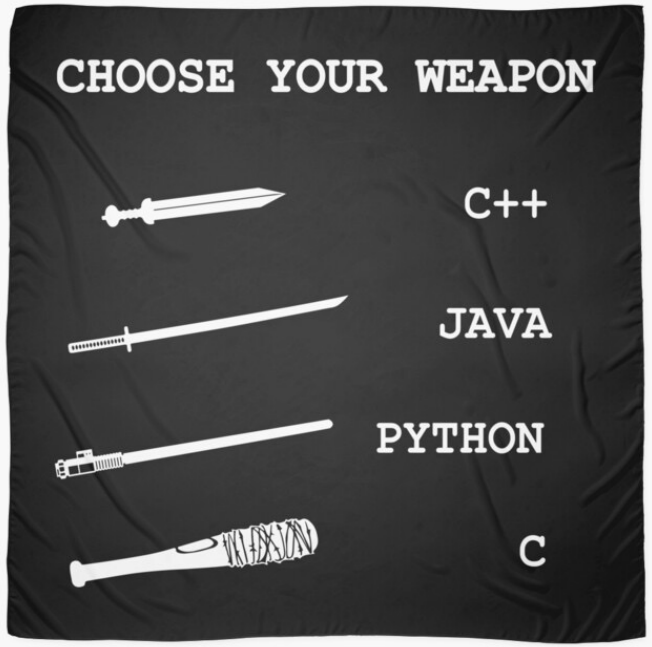
Základný princíp



https://res.cloudinary.com/practicaldev/image/fetch/s--fz7N-TNb--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https://i.imgur.com/Ms9ohSR.png

15

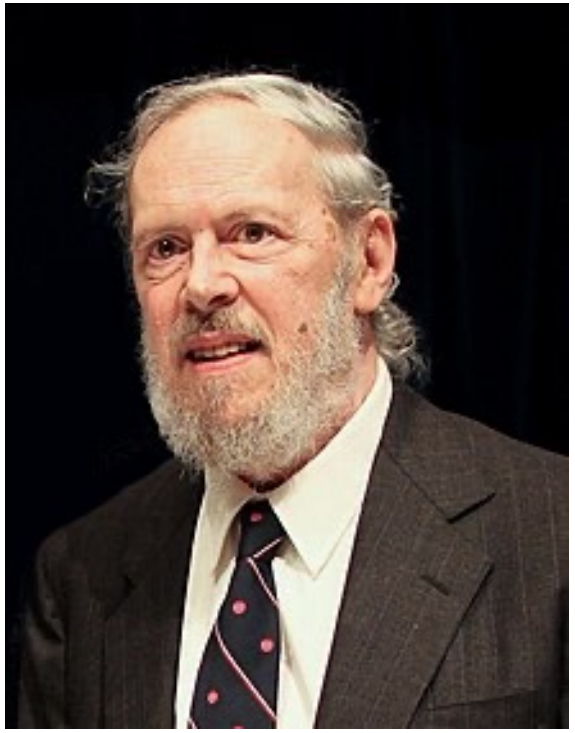
Úvod do jazyka C



16

O vzniku C

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



Dennis MacAlistair Ritchie
1941 – 2011

Autor jazyka C
Autor OS Unix

PhD @ Harvard University 1968

17

O vzniku C

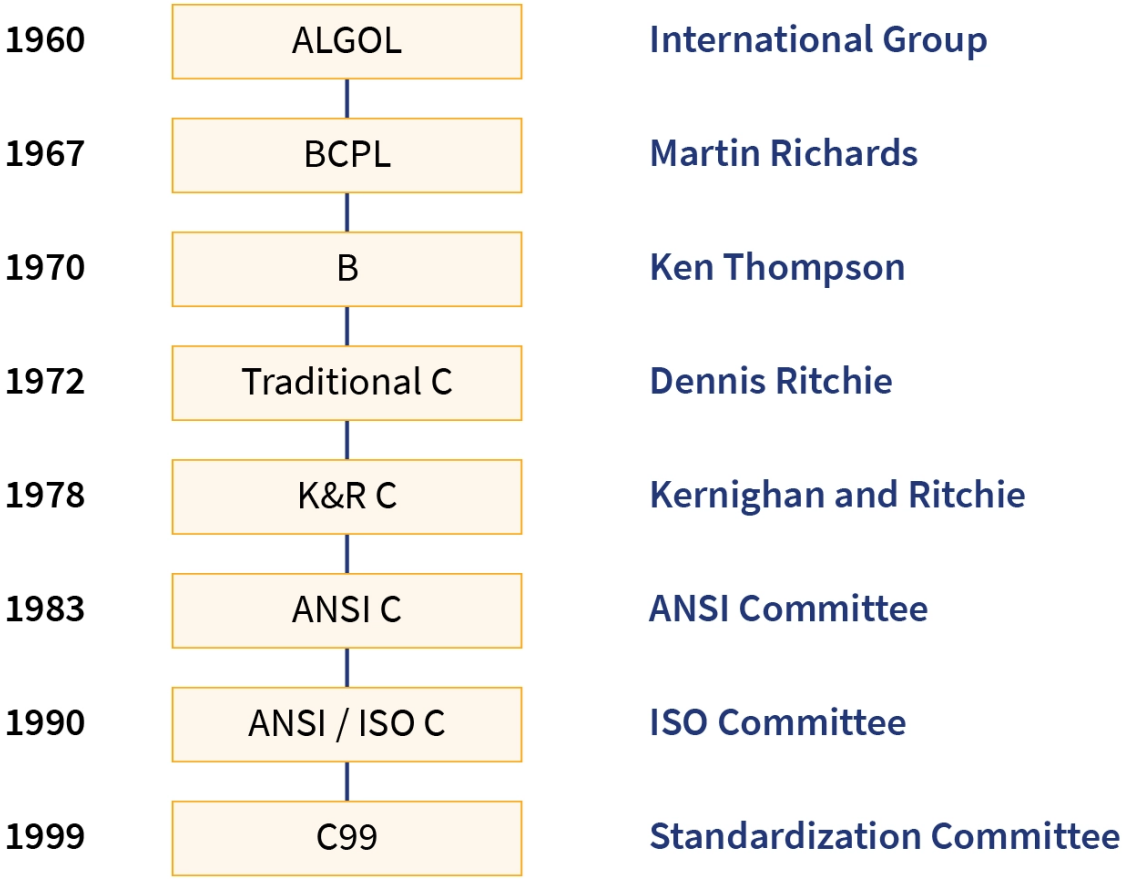
- **C je univerzálny jazyk nízkej úrovne**

- Pre vela 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 následnej dajú rozširovať

- Prenosný strojový jazyk, 100% prenositeľnosť (ANSI standard)

18

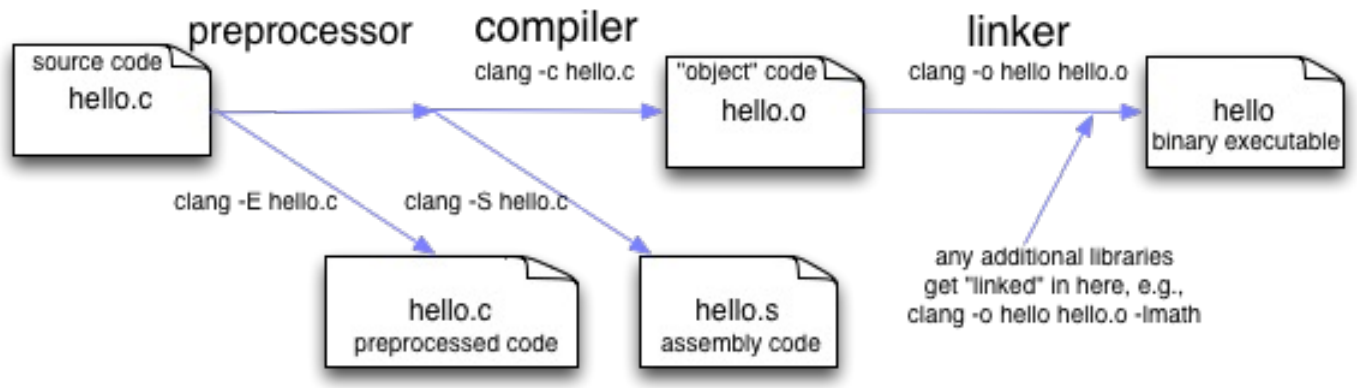
O vzniku C

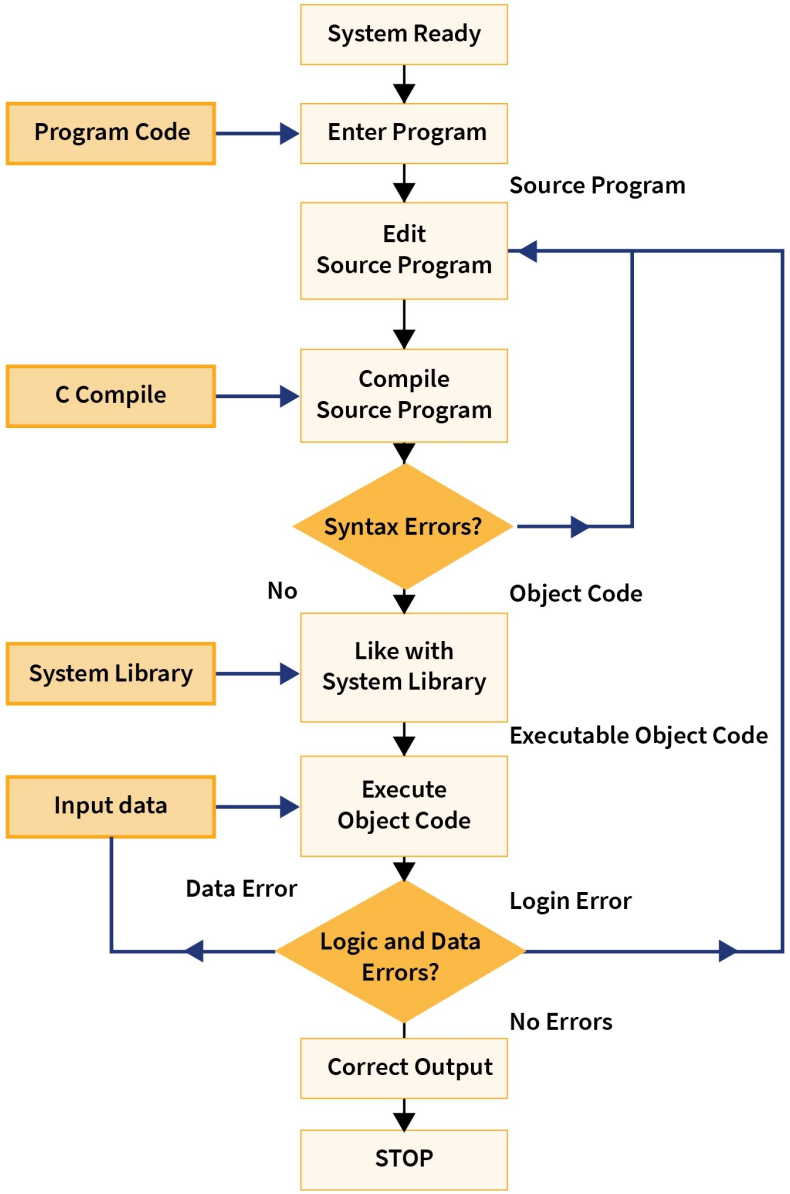


https://www.scaler.com/topics/c/history-of-c-language/

19

Spracovanie programu v jazyku C





SCALER Topics

https://www.scaler.com/topics/c/history-of-c-language/

Spracovanie programu v jazyku C

- **Pôvodný kód**

```c
//
//  main.c
//  testc2
//
//  Created by Marek Galinski on 19/09/2020.
//  Copyright © 2020 Marek Galinski. All rights reserved.
//

#include <stdio.h>

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


21

Spracovanie programu v jazyku C

- **Predspracovaný kód**

```
# 1 "/Users/marek/projects/testc2/testc2/main.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 362 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "/Users/marek/projects/testc2/testc2/main.c" 2









#pragma clang module import Darwin.C.stdio /* clang -E: implicit

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


22

Spracovanie programu v jazyku C

- **Kód v jazyku symbolických inštrukcií**

  (strojový kód, assembler)

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


23

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)

24

Chcem sa v roku 2022 učiť 50 rokov starý jazyk C?

25

Popularita C

- **TIOBE Index, 2022**
  - TIOBE measures the sheer quantity of 25 search engine hits – including popular Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube, and Baidu.

Programming Language	2022	2017	2012	2007
Python	1	5	8	7
C	2	2	1	2
Java	3	1	2	1
C++	4	3	3	3
C#	5	4	4	8
Visual Basic	6	14	-	-
JavaScript	7	8	10	9

26

Popularita C

- **PYPL Index, 2022**
  - PYPL – how often language tutorials are googled by exploring Google Trends.

Rank	Change	Language	Share	Trend
1		Python	28.29 %	-1.8 %
2		Java	17.31 %	-0.7 %
3		JavaScript	9.44 %	-0.1 %
4		C#	7.04 %	-0.1 %
5		C/C++	6.27 %	-0.4 %
6		PHP	5.34 %	-1.0 %

27

A čo iné jazyky?

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

28

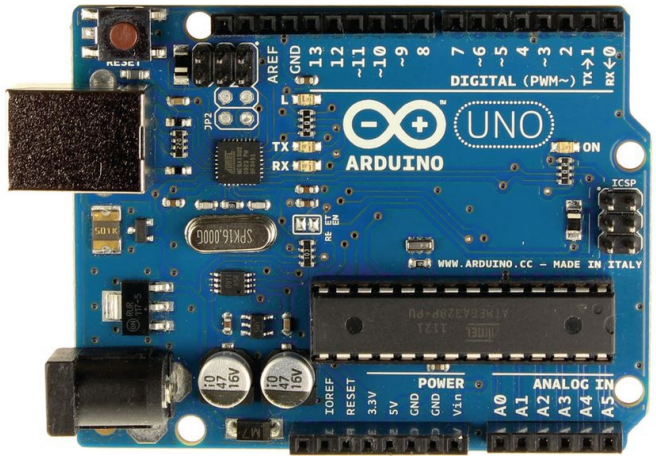
Využitie C

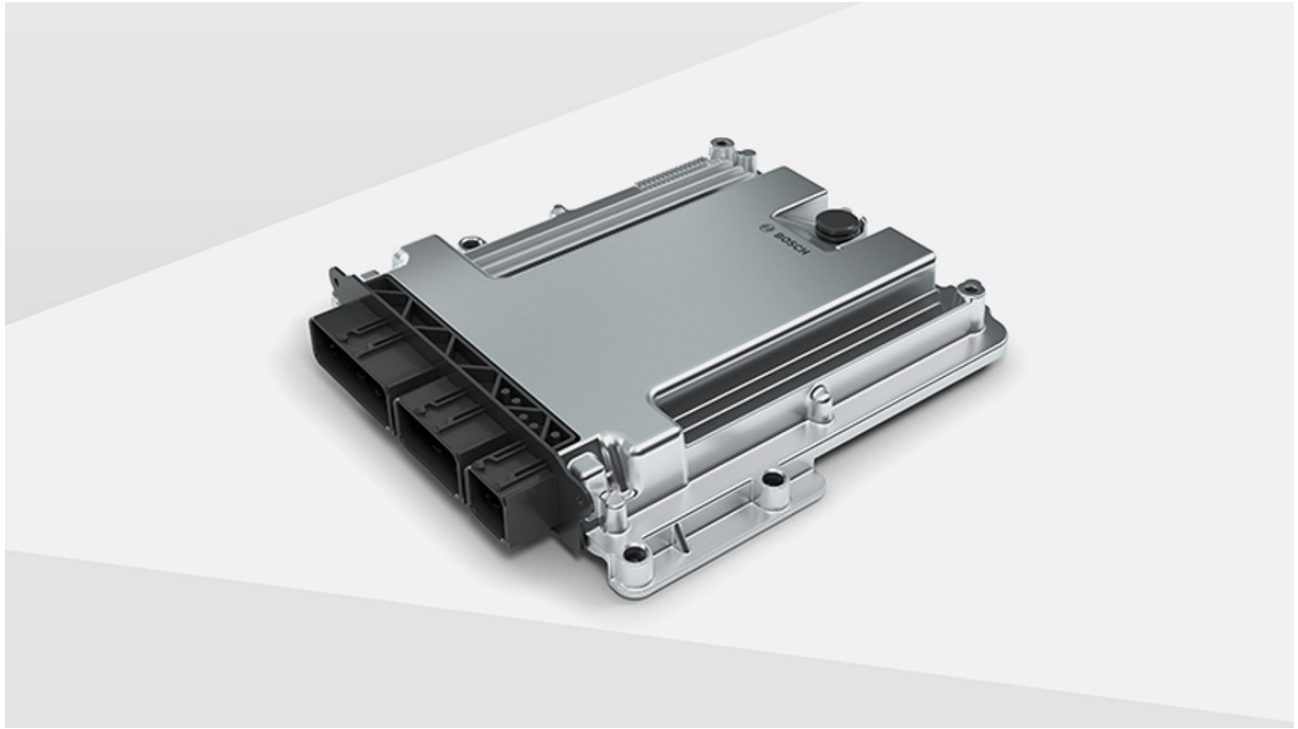
- **Prostredia s extrémnym dôrazom na spoľahlivosť kódu**

29

Využitie C

- **Microcontrollery, IoT, High-performance**





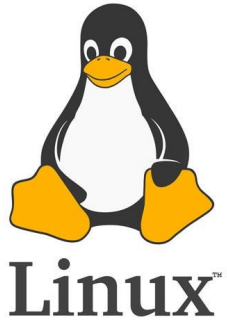
30

Využitie C


Windows


MySQL®


Ps
Photoshop
En
Encore
Ae
After Effects
Pr
Premiere Pro
Id
InDesign
Fl
Flash
Fb
Flash Builder
Ai
Illustrator
Br
Bridge
Fw
Fireworks
Dw
Dreamweaver
Au
Audition
Sg
SpeedGrade
Pl
Prelude
Lr
Lightroom


Linux™

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

32

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.

33

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.

34

Základné prvky jazyka C

35

Hello world!

```c
#include <stdio.h>

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


36

Hello world!

```c
#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

37

Hello world!

```c
#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()

38

Hello world!

```c
#include <stdio.h>

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

Parametre funkcie

39

Hello world!

```c
#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

40

Hello world!

```c
#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

41

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)

42

Hello world!

- **Komentáre v jazyku C**
  - **Komentáre sú dôležité – sprehľadňujú a popisujú kód, aby sa v ňom vyznal niekto iný, ale aj programátor sám.**

```
13      //toto je jednoriadkovy komentar
14      return 0;
```

```
13      /*toto funguje tiez */
14      return 0;
```

```
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;
```


43

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

44

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

45

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

46

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)

47

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)

48

Premenné a dátové typy

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

49

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`

50

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 ukočený bodkočiarkou

```
i * 2 + 3
```

```
j = i * 2 + 3
```

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


51

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 ukočený bodkočiarkou

```
i * 2 + 3
```

```
j = i * 2 + 3
```

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


52

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;
```


53

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**

54

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
  - Priraďovacie

55

Aritmetické výrazy

- **Unárne operatory**
  - Plus (+)
  - Mínus (-)

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


56

Aritmetické výrazy

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

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

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

57

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

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

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, oddeľujú 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 číslic a uložia ich ako číslo
  - Funkcie, ktoré konvertujú číselnú hodnotu do reťazca číslic

```
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 nespráva správne

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

66

Príklady

- **Zdrojové kódy príkladov sú dostupné aj na GitHube k predmetu**
  - Priečinok **Príklady k prednáškam**
    - **p01%_hodiny.c**
    - **p01%_obdlznik.c**
    - **p01%_priemer.c**

Q?

68