

Informatika 3

Kto je kto

Prednášajúci (a aj cvičiaci)

Viliam TAVAČ, Žilina
(viliam.tavac@fri.uniza.sk)
FRI A120

Miroslav Gábor, Prievídza, Žilina
(miroslav.gabor@fri.uniza.sk)
FRI A117

Len cvičiaci

Renáta Tóthová, Žilina
(renata.tothova@fri.uniza.sk)
FRI A316

Pomôcky pri štúdiu



Moodle

- Prezentácie
- Texty prednášok
- Riešené príklady z cvičení

Knihy

- Thinking in C++, 2nd Edition, Bruce Eckel,
<http://www.msedv.at/books/eckel/>
- C++ Primer Plus, Stephen Prata, 6th Edition, Addison-Wesley


Visual .net vývojové prostredie, help

Požiadavky na skúšku - Moodle

1. Získanie minimálne **10** bodov počas semestra.
2. Získať minimálne **17** bodov zo skúškového testu (z **30**).



Počet prepočítaných bodov = $(DU+KT+AC-10)/2$

- **DU** - domáce úlohy 
- **KT** - kontrolné testy
- **AC** - aktivita na cvičeniach

3. Ústna skúška

Pozícia C a C++ v praxi

Tiobe index

Sep 2014	Programming Language	Ratings
1	C	16.721%
2	Java	14.140%
3	Objective-C	9.935%
4	C++	4.674%
5	C#	4.352%
6	Basic	3.547%
7	PHP	3.121%
8	Python	2.782%
9	JavaScript	2.448%



C jazyk

- „vedľajší produkt“ vývoja operačného systému Unix 1969-1973
 - 1989 prijatý štandard ANSI-C, 1990 prijatý aj organizáciou ISO
 - C99 – prijatý ISO v 1999 a ANSI v marci 2000
 - C11 – december 2011
- **Stručný**
 - **Kompaktný výkonný cieľový kód**
 - **Možnosť ovládať hardvér**
 - **Strojovo-nezávislý jazyk**
 - **Prenositeľnosť na rôzne hardvérové konfigurácie**



Dennis Ritchie

Filozofia programovania v C

Program = Dáta + Algoritmy

Dáta – informácia, spracovávaná programom

Algoritmy – metódy programu na spracovanie dát

Procedurálny jazyk



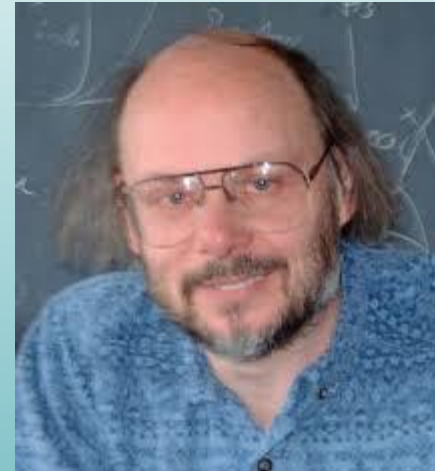
- Dôraz na algoritmickú stránku programovania
- Zhora nadol
- Funkcie (procedúry)

C++

- uľahčiť písanie dobrých programov



- 1983 – C++
- 1998 štandard ISO/IEC - ANSI C++ alebo C++98
- 2003 – aktuálna verzia C++03
- 2011 – August, C++11 ISO štandard



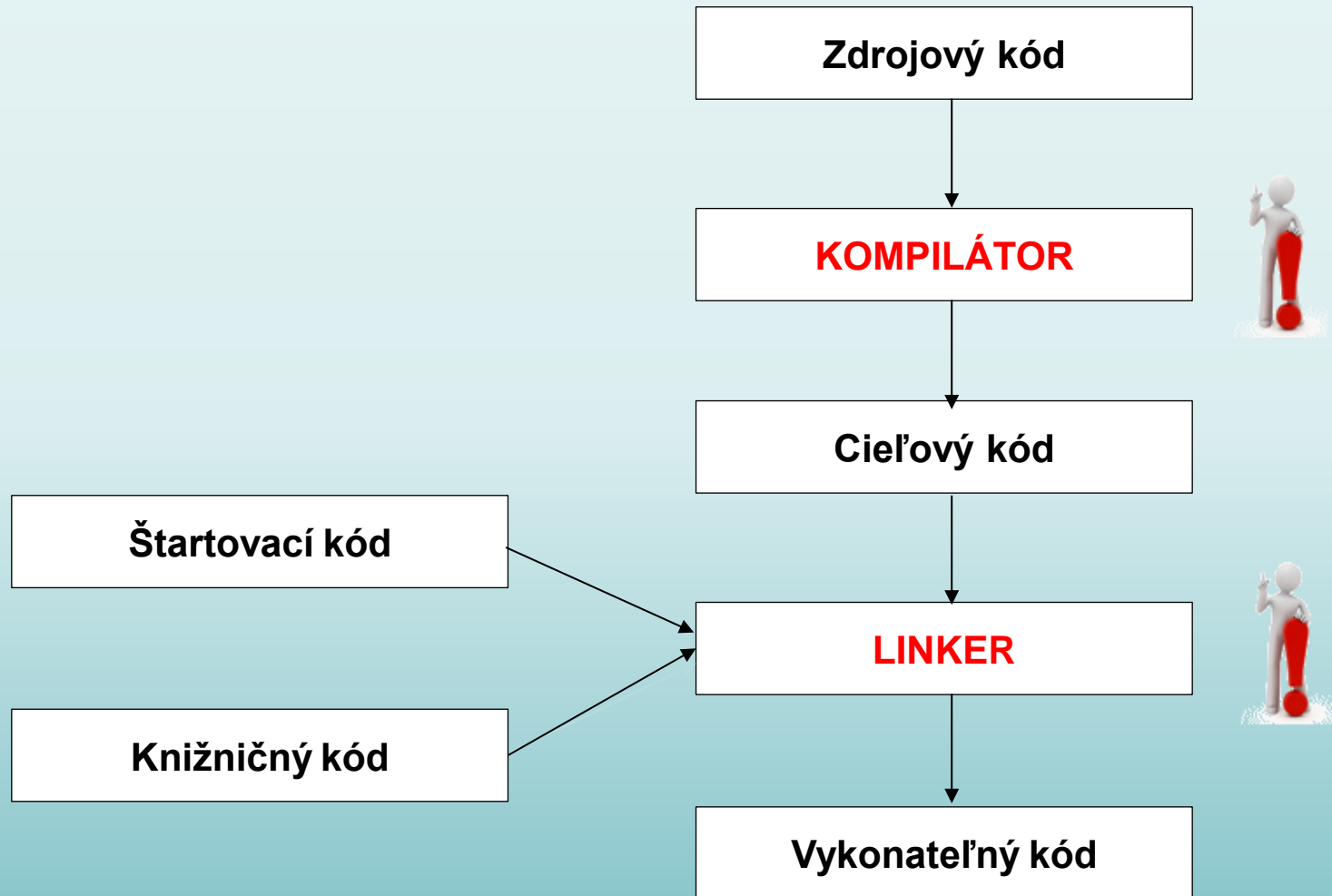
Bjarne Stroustrup

Filozofia programovania v C++

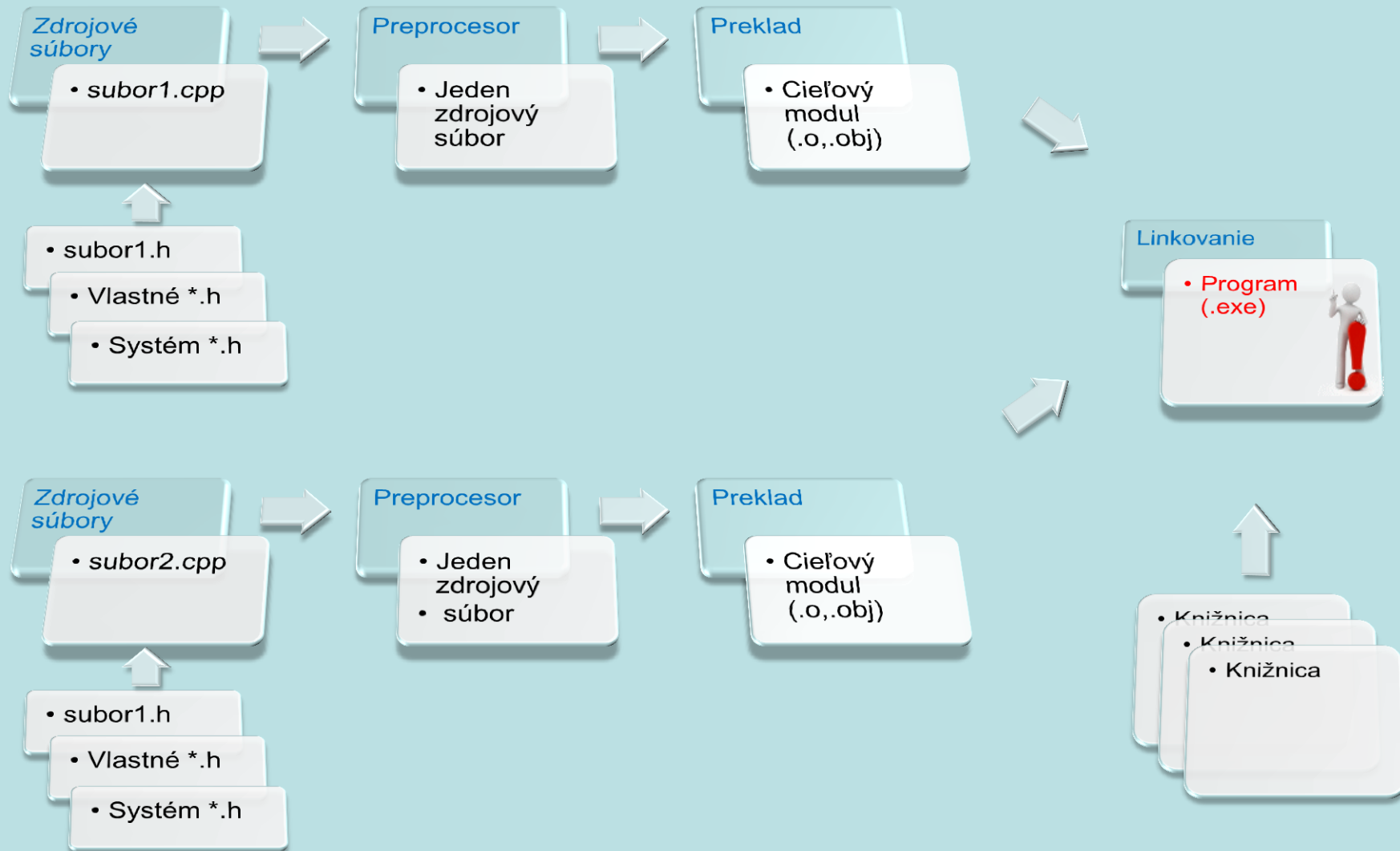
- OOP zdôrazňuje dáta
- **Prispôbiť jazyk problému**
- Návrh dátových formulárov, ktoré budú zodpovedať podstatným charakteristikám problému
 - class – objekt
- Programovanie zdola nahor
- Generické programovanie - šablóny



Mechanizmus tvorby programu v C a C++



Proces prekladu



Súbor - zdrojový kód

- Ľubovoľný textový editor
- Sofistikované vývojové prostredie

- **Visual Studio 2013**

(<http://www.fri.uniza.sk/stranka/softver-a-internet>)



„...zaradenie do licenčného programu Microsoft DreamSpark (predtým MSDN AA), kde si študenti FRI bezplatne môžu sťahovať a inštalovať softvér Microsoft a to operačné systémy, vývojové prostredie a aplikácie - podľa návodu [msdn](#) - čítajte návod - od 1.1. 2013 nový elektronický systém registrácie...”



Prípona zdrojového súboru

C++ implementácia	Prípona zdrojového súboru
Unix	C, cc, cxx, c
GNU C++	C, cc, cxx, cpp, c++, c
Microsoft Visual C++	cpp, cxx, cc, c



C a C++ program

```
// prvy.c - zobrazí oznam
```

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

```
int main()
{
    printf("Ahoj C++.");
    printf("\n");
    printf("Zaciname!\n");
    return 0;
}
```

```
// prvy.cpp - zobrazí oznam
```

```
#include <iostream>
```

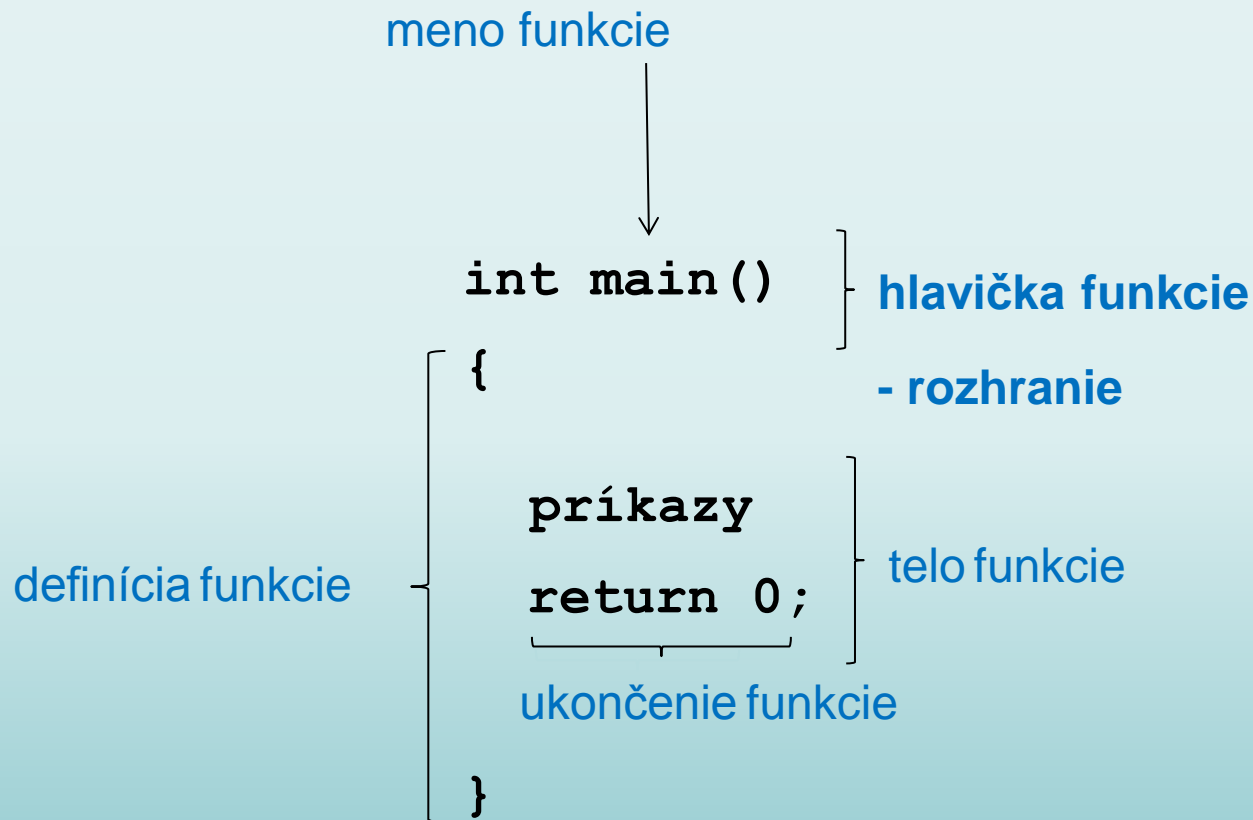
```
using namespace std;
```

```
int main()
{
    cout << "Ahoj C++. ";
    cout << endl;
    cout << "Zaciname!" << endl;
    return 0;
}
```

```
Ahoj C++.
Zaciname!
Press any key to continue . . .
```



Funkcia main



```
int main(int argc, char *argv[])
```

Komentáre



```
#include <stdio.h>    /* a C komentár */
```

```
#include <iostream> // a C++ komentár (od C99 i v C)
```

Preprocesor



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

- program, ktorý spracováva zdrojový súbor pred vykonaním samotnej kompilácie
- direktívy preprocesora začínajú znakom #

Hlavičkové (včleňované) súbory

`stdio.h, iostream`

- včleňujú sa na začiatku do iných súborov
- príkaz preprocesora **#include**

`#include <standardny>`



alebo

`#include "vlastny.h"`



Hlavičkové súbory - prípony



`stdio.h, iostream`

Druh hlavičky	Konvencia	Príklad	Použiteľné v programoch
C++ starý štýl	Končí príponou .h	iostream.h	C++
C starý štýl	Končí príponou .h	math.h	C a C++
C++ nový štýl	Bez prípony	iostream	C++, používa namespace std
Konvertované C	C prefix, bez prípony	cmath	C++, môžu používať nie-C vlastnosti

Namespace - menopriestor

- Písanie rozsiahlych programov, ktoré kombinujú existujúci kód od rôznych tvorcov
- Štandardné knižnice v meno priestore **std**
- Direktíva **using**
 - `using namespace std;`
 - `std::cout`



```
// prvy.cpp - zobrazí oznam
```

```
#include <iostream>
```

```
// using namespace std;
```

```
int main()
```

```
{
```

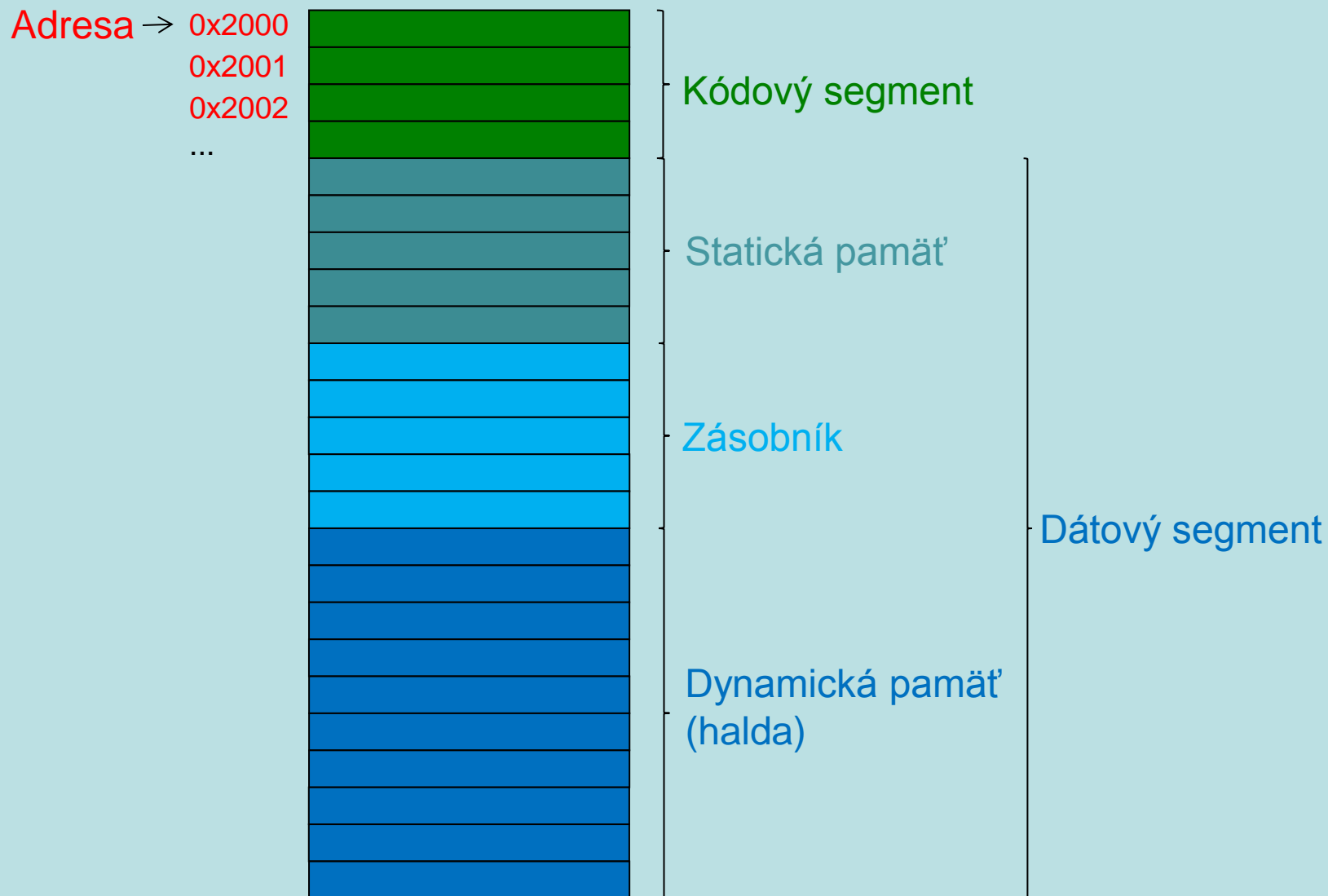
```
    std::cout << "Ahoj C++.";
```

```
    std::cout << std::endl;
```

```
    return 0;
```

```
}
```

Pamät'



Deklarácia

Oznamuje kompilátoru:

„*Táto funkcia alebo premenná niekde existuje a bude vypadat' takto.*“



```
int  func(int x, int y);    // deklarácie funkcie
extern int  a;              // deklarácia premennej
```

- Môže byť viacnásobná (ak je zhodná)
- Väčšinou sa umiestňuje do hlavičkového (včleňovaného) súboru
- Deklarácie funkcie = **prototyp funkcie**
 - Prototyp funkcie musíme umiestniť pred prvé použitie funkcie
 - Prototyp funkcie napíšeme explicitne do zdrojového kódu sami alebo
 - Včleníme do zdrojového kódu hlavičkový súbor, ktorý tento prototyp obsahuje.

Definícia

Prikazuje kompilátoru:

„*Vytvor túto premennú na tomto mieste*“

alebo

„*Vytvor túto funkciu na tomto mieste*“.



```
int a;                // definícia premennej
int func1(int,int)    // definícia funkcie
{
    // ... telo funkcie
}
```

- Alokuje pamäť pre identifikátor.
- Môže byť iba jedna (ODR – one definition rule)
- Umiestňuje sa do zdrojového súboru

- Flexibilné deklarácie (C++, C99)



Hlavička funkcie ako rozhranie

- rozhranie medzi **volanou** funkciou a **volajúcou** funkciou

návratový typ meno funkcie

`int` `fun` (`long` `pocet`)

zoznam parametrov (argumentov)



- **návratový typ** - popis informácie, ktorá ide z volanej funkcie do volajúcej funkcie
- **zoznam parametrov (argumentov)** - popis informácie, ktorá ide z volajúcej do funkcie volanej funkcie

Prototyp funkcie

Deklarácie funkcie = prototyp funkcie



- Prototyp funkcie musíme umiestniť pred prvé použitie funkcie
- Prototyp funkcie napíšeme explicitne do zdrojového kódu sami alebo
- Včleníme do zdrojového kódu hlavičkový súbor, ktorý tento prototyp obsahuje

Funkcie

S návratovou hodnotou - vytvára a vracia hodnotu

- môžeme ju priradiť do premennej alebo
- použiť v inom výraze

```
double sqrt(double) ;  
x = sqrt(6.25) ;
```

Bez návratovej hodnoty - niečo vykonáva, nevracia hodnotu

- procedúry alebo podprogramy

```
void fun(double) ;  
fun(10.3) ;
```

Knižničné funkcie

- uložené v knižničných súboroch
- automatické prehľadávanie knižníc a pripájanie knižnice
- explicitné prehľadávanie knižníc (-lm)
- štandardná knižnica C jazyka - viac ako 140 preddefinovaných funkcií

**Ak použitie štandardnej funkcie postačuje,
treba ju použiť a nevytvárať vlastnú.**



Funkcie definované programátorom

- prototyp funkcie – v hlavičkovom súbore
- definícia tela funkcie – v zdrojovom súbore

```
// mojafun.h - hlavičkový súbor
```

```
void mojaFun(int pocet);
```

```
// mojafun.c - zdrojový súbor
```

```
#include "mojaFun.h"
```



```
void mojaFun(int pocet)
{
    // ... kód
}
```


```
// hlavny.c- vstupný bod programu
```

```
#include "mojaFun.h"
```



```
int main(void)
{
    mojaFun(10);
}
```

Kľúčové slová

- slovník počítačového jazyka
- **nepoužívať na iné účely (ako identifikátory)** 

Konvencie pomenovania

- závisia od riešiteľského tímu
- mali by byť jednotné 

Premenné

Nástroj na identifikáciu dát



Program musí sledovať 3 vlastnosti:



- Kde je informácia uložená
- Akú hodnotu uchováva
- O aký druh informácie sa jedná

Názvy premenných

Názov musí byť zmysluplný !



Pravidlá tvorby:

1. V menách môžeme používať písmena abecedy, číslice a podtržítka (_).
2. Prvým znakom mena nesmie byť číslica. 
3. Malé a veľké písmena sa rozlišujú. 
4. Ako názov nemôžeme použiť kľúčové slovo jazyka C++.
5. Názvy, začínajúce podtržítkom alebo dvomi podtržítkami sú rezervované pre použitie kompilátorom a prostriedkami, ktoré používa.
6. C++ neohraničuje dĺžku názvu a všetky znaky mena sú významné.
(Avšak niektoré platformy môžu mať svoje vlastné limity. ANSI C99 garantuje len 63 znakov.)

Štandardné celočíselné typy

- **char** - má minimálnu šírku 8 bitov
- **short** - má minimálnu šírku 16 bitov
- **int** - je minimálne taký veľký ako short
- **long** - má minimálnu šírku 32 bitov a je minimálne taký ako int
- **long long** - má minimálnu šírku 64 bitov a je minimálne taký ako long
- **wchar_t** - široký znak – variabilná šírka

C++11

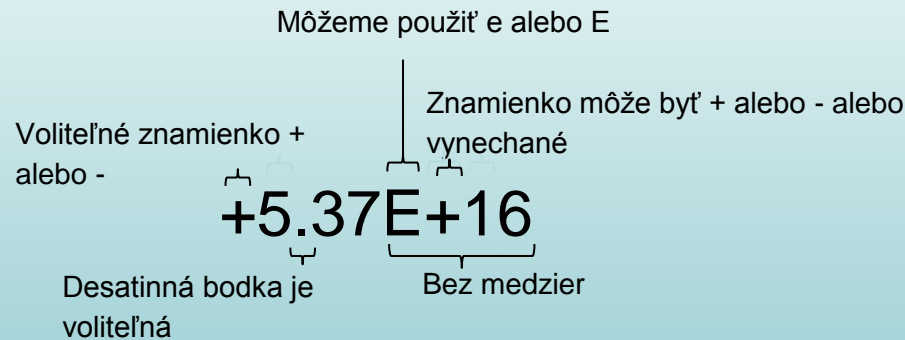
- **char16_t** - šírka 16 bitov
- **char32_t** - šírka 32 bitov

- Štandardne **signed**
- Modifikátor **unsigned**



Štandardné typy pohyblivej rádovej bodky

- **float** - 32 bitov
- **double** - 48 bitov a nie menej float
- **long double** - nie menej ako double (80, 96, 128)



- čísla medzi celými číslami
- omnoho väčší rozsah hodnôt
- strata presnosti



Číselné konštanty

Celočíselné


- celé číslo 1536
- celé čísla bez znamienka (54321u, 31U)
- hexadecimálne konštanty (**0x**31, **0X**1b2C)
- oktálové konštanty (**0**15, **0**324)
- Celé číslo typu long 1536**l** alebo 1536**L**



C++11

- Celé číslo typu long long 1536**ll** alebo 1536**LL**

Čísla s pohyblivou rádovou bodkou

- v desatinnom tvare (-35.245)
- v semilogaritmickej tvare (1e12, -22.56E-11, 1E+3)
- pridaním f, F k celému číslu 15**f**, -321**F** 

Znakové konštanty



Znakové konštanty - znak uzavretý v apostrofoch

- ľubovoľný ('a', '2')
- L'a' alebo l'a' - konštantu typu `wchar_t`
- u'a' – konštantu typu `char16_t`
- U'a' – konštantu typu `char32_t`
- špeciálne znaky:

Názov znaku	ASCII symbol	C++ kód	Desiatkový/Hexa kód
Nový riadok	NL (LF)	<code>\n</code>	10/0xA
Horizontálny tabelátor	HT	<code>\t</code>	9/0x9
Vertikálny tabelátor	VT	<code>\v</code>	11/0xB
Krok späť	BS	<code>\b</code>	8/0x8
Návrat vozíka	CR	<code>\r</code>	13/0xD
Výstraha	BEL	<code>\a</code>	7/0x7
Opačné lomítko	<code>\</code>	<code>\\</code>	92/0x5C
Otáznik	<code>?</code>	<code>\?</code>	63/0x3F
Jednoduchá úvodzovka	<code>'</code>	<code>\'</code>	39/0x27
Dvojitá úvodzovka	<code>"</code>	<code>\"</code>	34/0x22

Reťazcové konštanty – znaky uzavreté v úvodzovkách

"Ahoj.\nSom na prednaske z C-cka\n"

Definícia konštanty

Konštanty typu const

const typ premenná

- konštanta, ktorú môžeme používať ako premennú, ale nemôžeme do nej priamo zapisovať
- ak v deklarácii chýba typ, predpokladá sa „int“

const float pi = 3.1415926535;

const max = 10000;

Symbolické konštanty (literálové)

- sú konštanty definované „#define“
- nie je to ozajstná konštanta
- **nahrádza identifikátor textom uvedeným za ním**
- zvykom je písať ich veľkými písmenami

#define PI 3.1415926535

#define OZNAM "Toto je oznam"

#define begin {

#define end ;}



Inicializácia premennej

- `int pocet = INT_MAX; // C, C++`
- `int pocet(INT_MAX); // C++`
- `int pocet = {INT_MAX}; // C++11`
- `int pocet {INT_MAX}; // C++11`
- `int pocet {};` `// C++11, inicializácia na 0`



Aritmetické operátory

- **+** pre sčítanie
- **-** pre odčítanie
- ***** pre násobenie
- **/** pre delenie (ak sú obidva operandy celočíselného typu, výsledkom je celočíselná časť podielu a zlomková časť sa zahodí)
- **%** pre výpočet modula, tj, zvyšku po delení (obidva operandy musia byť celočíselného typu)

```
int rozsah = 10;  
int i = 0;  
  
i = (i + 1) % rozsah;
```



Konverzie pri inicializácii a priradovaní



Typ konverzie	Potencionálny problém
Väčší typ pohyblivej rádovej bodky na menší typ pohyblivej rádovej bodky	Strata presnosti (platné číslice) <ul style="list-style-type: none">- hodnota môže byť mimo rozsah cieľového typu (výsledok nedefinovaný)
Typ pohyblivej rádovej bodky na celočíselný typ	<ul style="list-style-type: none">- Strata zlomkovej časti- Pôvodná hodnota môže byť mimo rozsah cieľovej hodnoty (výsledok nedefinovaný)
Väčší celočíselný typ do menšieho celočíselného typu	Pôvodná hodnota môže byť mimo rozsah cieľového typu (zvyčajne sa priradia iba spodné bity hodnoty)

Deklarácia polí

- Jednorozmerné polia
`<typ> pr1[poč1], pr2[poč2], ... prN[počN];`
- Viacrozmerné polia
`<typ_premennej> prem1[poč1][poč2][počN];`
- Indexovanie prvkov poľa:
`0 ... počet_prvkov-1`
- príklad:
`unsigned char buf[512], c, pole[1024]
int matica[4][3];
buf[0]='a'; pole[1023]='x';
matica[0][0]=10;
Matica[3][2]=100;`

Výrazy

- Podobne ako v matematike
- Skladá sa z operátorov a argumentov
- Výraz môže byť:
 - konštanta: 3.2, "retazec"
 - premenná: i, a[j], x.c[v]->d[3]
 - volanie funkcie: sin(2*pi), f()
 - zložený výraz: a+b, 3*(x+y), (c-d)
 - binárny operátor: V1 op V2
 - Unárny operátor: op V

Operátory

- Aritmetické

+ - * / %

- / - ak sú oba argumenty celočíselné, výsledok je celočíselný
- $3/2 \Rightarrow 1$
- $3.0/2 \Rightarrow 1.5$

- Relačné

> >= < <= == != || && !(unárny)

- Pravda != 0 Nepravda == 0
- Výraz vo vyhodnocovaní končí ak je známa hodnota
- POZOR: (a==3 && funkcia())

- Logické po bitoch

& súčin

| súčet

^ XOR (operátor nonequivalencie po bitoch)

<< posun vľavo ($v1 \ll v2 \Leftrightarrow v1 * 2^{v2}$)

>> posun vpravo ($v1 \gg v2 \Leftrightarrow v1 / 2^{v2}$)

~ jednotkový doplnok (negácia po bitoch) (unárny)

- `c = 0x7f & a; /* nulovanie najvyššieho 8-meho bitu */`
- Prísne rozlišovať logické operátory po bitoch a logické spojky
- `a = 1; b = 2;`
- `c = a & b; /* c = 0 */`
- `d = a && b; /* d = 1 Prečo ? */`

Inkrementovanie a dekrementovanie

- Operátor inkrementovania a dekrementovania

++prem

--prem

prefixové

prem++

prem--

postfixové

- Prefixový – vráti hodnotu po zmene
- Postfixový – vráti hodnotu pred zmenou
- príklad:

a=2;

b=2;

c=a--;

d=--b;

/* c=2, a=1 */

/* d=1, b=1 */

Prirad'ovacie operátory

- Prirad'ovací operátor „=”

prem = výraz

- Návratová hodnota prirad'ovacieho výrazu je hodnota prenášaná cez „=”
- $a=b=c=d=4$

- Prirad'ovací operátor „op=”

- $\text{prem op} = \text{výraz} \quad \Leftrightarrow \quad \text{prem} = \text{prem op} \text{ výraz}$
- Je to bližšie ľudskému chápaniu (pripočítame..., vynásobíme...)
- Výborné, ak máme na ľavej strane priradenia zložitý výraz

i = i + 10;

i += 10;

- POZOR:

a *= b + 1

\Leftrightarrow

a = a * (b + 1)

nie

a = a * b + 1

?: , sizeof

- Podmienené výrazy

podm ? výraz1 : výraz2

- Príkazom

if(a>b) z=a;

else z=b;

- C++ operátor

z = a>b ? a : b

- Operátor čiarka

výraz1, výraz2

- Vykonáva sa zľava doprava
- Návratová hodnota je hodnota posledného výrazu

a = (t=2), t+1; /* a = 3 */

- Operátor sizeof

sizeof(objekt)

- objekt môže byť identifikátor premennej alebo typ
- produkuje celé číslo = veľkosť špecifikovaného objektu (premennej alebo typu) v bajtoch

Konverzie typu – implicitná

- ak sa objavia operandy rôznych typov vo výrazoch
- vo všeobecnosti sa automaticky vykonávajú iba také konverzie, ktoré majú zmysel (reálna premenná v indexe je sabotáž)
- „nižší typ“ sa pred vykonaním operácie konvertuje na „vyšší typ“ a výsledok je vyššieho typu
- **Pravidlá:**
 1. **char, short** => **int**
 2. **float** => **double**
 3. **ak je 1. double** => **2. konvertuj na double** výsledok je **double**
 4. **inak ak je 1. long** => **2. konvertuj na long** výsledok je **long**
 5. **inak ak je 1. unsigned** => **2. konvertuj na unsigned** výsledok je **unsigned**
 6. **inak musia byť operandy int** a výsledok je **int**
- **Poznámka:**
 - Všetky aritmetické operácie v C++ sa vykonávajú v dvojnásobnej presnosti

Konverzie typu – operátor priradenia

- Operátor priradenia má ďalšie pravidlo: Hodnota pravej strany sa pred priradením konvertuje na typ ľavej strany a taký je aj výsledok. Pričom konverzie prebiehajú nasledovne:

int => char

horné bity sa ignorujú

long => int

vyššie rády sa ignorujú

float => int

odsekne sa zlomková časť

double => float

zaokrúhlenie

Konverzie typu – explicitná – operátor pretypovania (operátor roly)

(typ) výraz

- Príklady

```
int a,b;
```

```
float c;
```

```
a = 3; b=2;
```

```
c = a/b;          /* c = 1 */
```

```
c = a/(float)b;   /* c = 1.5 */
```

```
c = a/2           /* c = ? */
```

```
c = a/2f          /* c = ? */
```


Priorita operátorov

#	Category	Operator	What it is (or does)
1. Highest		() [] -> :: .	Function call Array subscript C++ indirect component selector C++ scope access/resolution C++ direct component selector
2. Unary		! ~ + - ++ -- & * sizeof new delete	Logical negation (NOT) Bitwise (1's) complement Unary plus Unary minus Preincrement or postincrement Predecrement or postdecrement Address Indirection (returns size of operand, in bytes) (dynamically allocates C++ storage) (dynamically deallocates C++ storage)
3. Member access		.* ->*	C++ dereference C++ dereference
4. Multiplicative		* / %	Multiply Divide Remainder (modulus)
5. Additive		+ -	Binary plus Binary minus
6. Shift		<< >>	Shift left Shift right

#	Category	Operator	What it is (or does)
7. Relational		< <= > >=	Less than Less than or equal to Greater than Greater than or equal to
8. Equality		== !=	Equal to Not equal to
9.		&	Bitwise AND
10.		^	Bitwise XOR
11.			Bitwise OR
12.		&&	Logical AND
13.			Logical OR
14. Conditional		?:	(a ? x : y)
15. Assignment		= *= /= %= += -= &= ^= = <<= >>=	Simple assignment Assign product Assign quotient Assign remainder (modulus) Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift
16. Comma		,	Evaluate

typedef

- umožní pomenovať typ syntakticky ekvivalentným identifikátorom
- syntax:

```
typedef definícia_typu identifikatory_typu;
```

```
typedef int cele, PoleCelych[30];
```

```
cele i, j, p[5];      // i,j – celočísl. prem.  
                     // p – pole celych čísiel
```

```
PoleCelych p2; // p2 pole celych čísiel – 30 prvkov
```

- rovnocenná definícia:

```
int i, j, k, p[5], p2[30];
```
- int a cele sú rovnocenné typy (môžu sa kombinovať)
- výhoda - sprehľadnenie zápisu programu
- Pozor:

```
int cele; // cele – premenná typu int  
typedef int cele; // cele – „vlastný typ“ typu int
```

enum

- Umožní - deklarovat' množinu konštánt typu int
 - pridelit' tejto množine meno
 - priradiť konštantám hodnoty
- Deklarácia

```
enum meno_mnoziny { identifikátor [=konštanta][,...] } ;
```
- Príklad:

```
enum konstanty { nula, prva, druha, siedma=7, osma };
```
- Ak nie je uvedená konštanta, každá nasledujúca konštanta má hodnotu o jedna väčšiu ako predchádzajúca, prvá má implicitnú hodnotu nula t.j.

```
nula=0, prva=1, druha=2, siedma=7, osma=8
```
- deklaráciu "enum" možno kombinovať s deklaráciou "typedef"

```
typedef enum { nula, jedna, dva, osem=8, devat } cisla;  
cisla c1,c2,c3;
```
- Každá konštanta, deklarovaná pomocou 'enum', môže byť použitá namiesto konštanty typu 'int'