Programování 3

Objektově orientované programování

Martin Pergel

28. září 2020

Pragmatické informace

- Volitelný předmět (většinou),

Pragmatické informace

- Volitelný předmět (většinou),
- zápočet: zápočtový program (s dokumentací), aktivní účast na cvičení (body v CodExu), praktický test,

Pragmatické informace

- Volitelný předmět (většinou),
- zápočet: zápočtový program (s dokumentací), aktivní účast na cvičení (body v CodExu), praktický test,
- zkouška: zkoušková písemka na objektový návrh a následná ústní část (o tom, co se probere).

Cíle předmětu

- Programování v C a C++,

Cíle předmětu

- Programování v C a C++,
- objektové programování,

Cíle předmětu

- Programování v C a C++,
- objektové programování,
- moderní vývojové prostředky.

Souvislosti

- Prvácké Programování 2 s kódem NPRG je výrazně odlišné od Programování 2 s kódem NMIN,

Souvislosti

- Prvácké Programování 2 s kódem NPRG je výrazně odlišné od Programování 2 s kódem NMIN,
- proto matematici nejsou schopni navštěvovat prakticky žádné navazující přednášky o programování.

Souvislosti

- Prvácké Programování 2 s kódem NPRG je výrazně odlišné od Programování 2 s kódem NMIN,
- proto matematici nejsou schopni navštěvovat prakticky žádné navazující přednášky o programování.
- Kolegové Matematici si vyžádali C a C++.

Souvislosti

- Prvácké Programování 2 s kódem NPRG je výrazně odlišné od Programování 2 s kódem NMIN,
- proto matematici nejsou schopni navštěvovat prakticky žádné navazující přednášky o programování.
- Kolegové Matematici si vyžádali C a C++.
- Budeme stavět na Pythonu.

Prostředky

- Budeme používat ReCodEx,

Prostředky

- Budeme používat ReCodEx,
- využívat budeme buď to Microsoft Visual Studio (v dostupné verzi),

Prostředky

- Budeme používat ReCodEx,
- využívat budeme buďto Microsoft Visual Studio (v dostupné verzi),
- nebo prostředí MONO (obojí je k dispozici legálně zdarma aspoň v omezené verzi).

Prostředky

- Budeme používat ReCodEx,
- využívat budeme buďto Microsoft Visual Studio (v dostupné verzi),
- nebo prostředí MONO (obojí je k dispozici legálně zdarma aspoň v omezené verzi).
- Tvořit budeme převážně konzolové aplikace.

Historie

- Navržen v Bell Labs v rámci vývoje UNIXu,
- podle K. Thompsona dostali B. Kernighan a D. Ritchie uloženo vytvořit obfuscovaný Pascal,
- navrhli jazyk A, se kterým nebyli spokojeni,
- proto navrhli jayzk B, ...,
- spokojeni byli až s jazykem C,
- jazyk se ovšem začal masívně využívat.

Všeobecné vlasntosti jazyka C

- Syntax vychází z Pascalu,
- plno věcí je ale navrženo odlišně,
- jde o typový jazyk, tedy proměnné mají stanovený datový typ a proměnné je třeba definovat.
- Jazyk je výrazně zjednodušen (není string a boolean).
- Existuje několik norem (K & R, ANSI C86, C99).
- Plíživě dochází k omezování možností, C++ možnosti omezuje ještě více.

C VS C++

- C je jazyk podobný Pascalu a výrazně odlišný od Pythonu,
- C++ je jazyk, ve kterém je k vidění téměř vše,
- mají velký průnik syntaxe společný, C++ podporuje objekty, C podporuje kouzlení.
- Obvykle uvidíte překladače C++, ale nikdy nevíte, kdy budete potřebovat C, proto si řekneme o obou jazycích.
- Překladače: GCC, G++, Visual Studio .NET,...

Datové typy a proměnné

- Ve všech jazycích pracujeme s proměnnými,
- každá proměnná má stanoven datový typ,
- `char`, `byte`, `int`, `float`, `double`
- modifikátory (intu): `long`, `short`, `unsigned`.
  Například: `unsigned long int`.
- Ovšem stačí jen: `short` nebo `unsigned long`.
- Definice proměnných: `int a,b,c;`
- Nechybějí ve výčtu některé důležité datové typy?

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,
- nula je `false`, cokoliv jiného znamená `true`.

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,
- nula je `false`, cokoliv jiného znamená `true`.
- Místo `stringu` použijeme pointer na `char`,

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,
- nula je `false`, cokoliv jiného znamená `true`.
- Místo `stringu` použijeme pointer na `char`,
- jednotlivé prvky stringu budeme ukládat na adresy po sobě jdoucí, pointer ukazuje na začátek.

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,
- nula je `false`, cokoliv jiného znamená `true`.
- Místo `stringu` použijeme pointer na `char`,
- jednotlivé prvky stringu budeme ukládat na adresy po sobě jdoucí, pointer ukazuje na začátek.
- V C++ tyto typy již jsou, nicméně procvičení pointerů není nikdy od věci.

Chybějící datové typy

a jak se s nimi vypořádáme:

- `boolean` nahrazen `intem`,
- nula je `false`, cokoliv jiného znamená `true`.
- Místo `stringu` použijeme pointer na `char`,
- jednotlivé prvky stringu budeme ukládat na adresy po sobě jdoucí, pointer ukazuje na začátek.
- V C++ tyto typy již jsou, nicméně procvičení pointerů není nikdy od věci.
- Než se dostaneme k práci s pointery, není moc co cvičit.

Pointery

alias ukazatele

- V Pythonu nejsou, tam jsou jen reference,
- reference je automatizovaný pointer, pointer (čili ukazatel) ukazuje na nějakou adresu.
- U pointeru my rozhodneme, kdy se dereferencuje (tedy kdy koukneme pod něj).
- Paměť je stále lineárně organizovaná (dělena do jednotek zvaných adresy, každá adresa má číslo).
- Můžeme udělat typ ukazatel (na něco), do kterého zapíšeme adresu, na které se nacházejí data (dotyčného typu).
- Něco podobného jste dělali při práci se spojovými seznamy, jenže to byly reference na objekty, my budeme dělat pointery i na primitivní typy.

Od Pythonu k C

v Pythonu umíme především:

- Tělo hlavního programu,
- používat bloky (značené indentací),
- komentovat zdrojové texty,
- definovat funkce a procedury,
- používat (nikoliv definovat) proměnné,
- vracet návratovou hodnotu,
- pracovat s operátory,
- používat základní řídící struktury.

Case sensitivita

je stejná skoro u všech jazyků z familie C

- C, C++, C#, Java, Javascript... case-sensitive,
- tedy záleží na velikosti písmen.
- Ovšem pozor, i funkce (pro vstup a výstup) se v různých jazycích jmenují různě!
- Modernější vývojová prostředí našeptávají, je ale dobré nestat se na těchto prostředích naprosto závislým.

Kompilované a interpretované jazyky

a rozdíly mezi nimi

- Python je interpretovaný, C a C++ jsou kompilované.
- U kompilovaných jazyků je třeba před spuštěním (na zdrojové kódy) poslat kompilátor (a. k. a. překladač), který vytvoří binární soubor (která pak spustíme).
- Ruční volání: `gcc zdrojak.c`
- Implicitní vyvolání (v IDE MS VS): `Ctrl+Shift+B`

Hlavní program

- V Pythonu je hlavní program volně sypaný ve zdrojáku.

Hlavní program

- V Pythonu je hlavní program volně sypaný ve zdrojáku.
- Jazyk C je metodičtější: funkce `main`,

Hlavní program

- V Pythonu je hlavní program volně sypaný ve zdrojáku.
- Jazyk C je metodičtější: funkce `main`,
- tato funkce bere dva argumenty: Počet argumentů (int) a dotyčné argumenty (pole stringů, tedy char-pointerů).

Hlavní program

- V Pythonu je hlavní program volně sypaný ve zdrojáku.
- Jazyk C je metodičtější: funkce `main`,
- tato funkce bere dva argumenty: Počet argumentů (int) a dotyčné argumenty (pole stringů, tedy char-pointerů).
- Vrací int (čímž může říct, zda se program nedostal do potíží).

Hlavní program

- V Pythonu je hlavní program volně sypaný ve zdrojáku.
- Jazyk C je metodičtější: funkce `main`,
- tato funkce bere dva argumenty: Počet argumentů (int) a dotyčné argumenty (pole stringů, tedy char-pointerů).
- Vrací int (čímž může říct, zda se program nedostal do potíží).
- Občas se vstupní bod programu jmenuje jinak (`_tmain`) nebo (`wmain`).

Hlavní program

- V Pythonu je hlavní program volně sypaný ve zdrojáku.
- Jazyk C je metodičtější: funkce `main`,
- tato funkce bere dva argumenty: Počet argumentů (int) a dotyčné argumenty (pole stringů, tedy char-pointerů).
- Vrací int (čímž může říct, zda se program nedostal do potíží).
- Občas se vstupní bod programu jmenuje jinak (`_tmain`) nebo (`wmain`).
- Tyto funkce mohou umožnit pohodlnější práci například s parametry (pokud mají smysl).

Typové jazyky

a jejich specifika

- Proměnné musíme před použitím definovat (říct jak se budou jmenovat a jakého budou typu).

Typové jazyky

a jejich specifika

- Proměnné musíme před použitím definovat (říct jak se budou jmenovat a jakého budou typu).
- Je třeba stanovit území platnosti: Globální a lokální proměnné - odlišíme podle místa, kde je proměnná definována.

Typové jazyky

a jejich specifika

- Proměnné musíme před použitím definovat (říct jak se budou jmenovat a jakého budou typu).
- Je třeba stanovit území platnosti: Globální a lokální proměnné - odlišíme podle místa, kde je proměnná definována.
- Jazyk C umožňuje při definici proměnné tuto rovnou inicializovat.

Typové jazyky

a jejich specifika

- Proměnné musíme před použitím definovat (říct jak se budou jmenovat a jakého budou typu).
- Je třeba stanovit území platnosti: Globální a lokální proměnné - odlišíme podle místa, kde je proměnná definována.
- Jazyk C umožňuje při definici proměnné tuto rovnou inicializovat.
- Proměnnou definujeme (v území platnosti) jednou, pak už ji používáme.

Typové jazyky

a jejich specifika

- Proměnné musíme před použitím definovat (říct jak se budou jmenovat a jakého budou typu).
- Je třeba stanovit území platnosti: Globální a lokální proměnné - odlišíme podle místa, kde je proměnná definována.
- Jazyk C umožňuje při definici proměnné tuto rovnou inicializovat.
- Proměnnou definujeme (v území platnosti) jednou, pak už ji používáme.
- Proměnná v území platnosti **nemění** datový typ!

Území platnosti proměnných

v Pythonu je obestřeno mlhou, ve skutečnosti jsou proměnné vždy lokální nebo globální

- Proměnné mohou být globální (ty jsou k dispozici stále po celou dobu běhu programu, sedí staticky v paměti),

Území platnosti proměnných

v Pythonu je obestřeno mlhou, ve skutečnosti jsou proměnné vždy lokální nebo globální

- Proměnné mohou být globální (ty jsou k dispozici stále po celou dobu běhu programu, sedí staticky v paměti),
- anebo lokální (ty existují jen při zavolání dotyčné funkce, ve které jsou lokální).

Území platnosti proměnných

v Pythonu je obestřeno mlhou, ve skutečnosti jsou proměnné vždy lokální nebo globální

- Proměnné mohou být globální (ty jsou k dispozici stále po celou dobu běhu programu, sedí staticky v paměti),
- anebo lokální (ty existují jen při zavolání dotyčné funkce, ve které jsou lokální).
- Lokální proměnné definujeme v příslušné funkci (nejlépe hned na začátku),

Území platnosti proměnných

v Pythonu je obestřeno mlhou, ve skutečnosti jsou proměnné vždy lokální nebo globální

- Proměnné mohou být globální (ty jsou k dispozici stále po celou dobu běhu programu, sedí staticky v paměti),
- anebo lokální (ty existují jen při zavolání dotyčné funkce, ve které jsou lokální).
- Lokální proměnné definujeme v příslušné funkci (nejlépe hned na začátku),
- globální proměnné definujeme vně funkcí.

Bloky a komentáře

- V Pythonu se bloky označovaly indentací (což bylo legrační, když textový editor začal zaměňovat tabelátory a mezery).

Bloky a komentáře

- V Pythonu se bloky označovaly indentací (což bylo legrační, když textový editor začal zaměňovat tabelátory a mezery).
- V jazyku C se bloky zahajují resp. ukončují složenými závorkami.

Bloky a komentáře

- V Pythonu se bloky označovaly indentací (což bylo legrační, když textový editor začal zaměňovat tabelátory a mezery).
- V jazyku C se bloky zahajují resp. ukončují složenými závorkami.
- I komentáře se v C syntakticky liší. Jsou zde dvě možnosti:

Bloky a komentáře

- V Pythonu se bloky označovaly indentací (což bylo legrační, když textový editor začal zaměňovat tabelátory a mezery).
- V jazyku C se bloky zahajují resp. ukončují složenými závorkami.
- I komentáře se v C syntakticky liší. Jsou zde dvě možnosti:
- Komentář obecný: `/* Zde je komentar... a bude az do ukonceni komentare. */`

Bloky a komentáře

- V Pythonu se bloky označovaly indentací (což bylo legrační, když textový editor začal zaměňovat tabelátory a mezery).
- V jazyku C se bloky zahajují resp. ukončují složenými závorkami.
- I komentáře se v C syntakticky liší. Jsou zde dvě možnosti:
- Komentář obecný: `/* Zde je komentar... a bude az do ukonceni komentare. */`
- Komentář jednořádkový (pozor, proti ANSI C86):
  `// Komentar do konce radku.`

Procedury a funkce

aneb proč jsem už v prváku nerad slyšel o procedurách

- Procedura je součást programu, která umí načíst a zpracovat zadané parametry.

Procedury a funkce

aneb proč jsem už v prváku nerad slyšel o procedurách

- Procedura je součást programu, která umí načíst a zpracovat zadané parametry.
- Funkce je součást programu, která umí načíst a zpracovat zadané parametry a vrátit výsledek.

Procedury a funkce

aneb proč jsem už v prváku nerad slyšel o procedurách

- Procedura je součást programu, která umí načíst a zpracovat zadané parametry.
- Funkce je součást programu, která umí načíst a zpracovat zadané parametry a vrátit výsledek.
- Vidíme, že procedura je jen funkce, která nevrací hodnotu, proto v C procedury nejsou.

Procedury a funkce

aneb proč jsem už v prváku nerad slyšel o procedurách

- Procedura je součást programu, která umí načíst a zpracovat zadané parametry.
- Funkce je součást programu, která umí načíst a zpracovat zadané parametry a vrátit výsledek.
- Vidíme, že procedura je jen funkce, která nevrací hodnotu, proto v C procedury nejsou.
- Procedura je funkce vracející nic. Syntakticky tedy zavedeme prázdný datový typ `void`.

Procedury a funkce

aneb proč jsem už v prváku nerad slyšel o procedurách

- Procedura je součást programu, která umí načíst a zpracovat zadané parametry.
- Funkce je součást programu, která umí načíst a zpracovat zadané parametry a vrátit výsledek.
- Vidíme, že procedura je jen funkce, která nevrací hodnotu, proto v C procedury nejsou.
- Procedura je funkce vracející nic. Syntakticky tedy zavedeme prázdný datový typ `void`.
- V typových jazycích i funkce má definovaný datový typ, který má vrátit.

Procedury a funkce

příklad

- V Pythonu: `def nazev(arg1, arg2,...):`

Procedury a funkce

příklad

- V Pythonu: `def nazev(arg1, arg2,...):`
- V C: `navr_typ nazev(typ1 arg1, typ2 arg2,...)`

Procedury a funkce

příklad

- V Pythonu: `def nazev(arg1, arg2,...):`
- V C: `navr_typ nazev(typ1 arg1, typ2 arg2,...)`
- Poznámky: I parametry funkcí mají předepsaný typ. Namísto slova `function` klademe návratový typ dotyčné funkce.

Procedury a funkce

příklad

- V Pythonu: `def nazev(arg1, arg2,...):`
- V C: `navr_typ nazev(typ1 arg1, typ2 arg2,...)`
- Poznámky: I parametry funkcí mají předepsaný typ. Namísto slova `function` klademe návratový typ dotyčné funkce.
- Povšimněte si chybějícího středníku na konci hlavičky. Uvedeme-li středník v C, jde o deklaraci (která v Pythonu neměla smysl, ale v typových jazycích platí pravidlo napřed definovat, potom použít).

Procedury a funkce

příklad

- V Pythonu: `def nazev(arg1, arg2,...):`
- V C: `navr_typ nazev(typ1 arg1, typ2 arg2,...)`
- Poznámky: I parametry funkcí mají předepsaný typ. Namísto slova `function` klademe návratový typ dotyčné funkce.
- Povšimněte si chybějícího středníku na konci hlavičky. Uvedeme-li středník v C, jde o deklaraci (která v Pythonu neměla smysl, ale v typových jazycích platí pravidlo napřed definovat, potom použít).
- Funkce se volají jako v Pythonu, vždy je třeba použít operátor zavolání (kulaté závorky s argumenty – byť prázdnými)!

Procedury a funkce

příklad

- V Pythonu: `def nazev(arg1, arg2,...):`
- V C: `navr_typ nazev(typ1 arg1, typ2 arg2,...)`
- Poznámky: I parametry funkcí mají předepsaný typ. Namísto slova `function` klademe návratový typ dotyčné funkce.
- Povšimněte si chybějícího středníku na konci hlavičky. Uvedeme-li středník v C, jde o deklaraci (která v Pythonu neměla smysl, ale v typových jazycích platí pravidlo napřed definovat, potom použít).
- Funkce se volají jako v Pythonu, vždy je třeba použít operátor zavolání (kulaté závorky s argumenty – byť prázdnými)!
- Tři tečky lze takto v C opravdu použít (ale o tom později).

Definice proměnných

a inicializace

- `int a,b,c;`

Definice proměnných

a inicializace

- `int a,b,c;`
- Definujeme proměnné `a, b, c` typu `int`.

Definice proměnných

a inicializace

- `int a,b,c;`
- Definujeme proměnné `a, b, c` typu `int`.
- Definice s inicializací: `int a=0,b=1,c=2;`

Definice proměnných

a inicializace

- `int a,b,c;`
- Definujeme proměnné `a, b, c` typu `int`.
- Definice s inicializací: `int a=0,b=1,c=2;`
- Podle C86 bylo třeba lokální proměnné definovat na začátku funkce, podle C99 lze prakticky kdekoliv (ale není vhodné toho zneužívat).

Definice proměnných

a inicializace

- `int a,b,c;`
- Definujeme proměnné `a, b, c` typu `int`.
- Definice s inicializací: `int a=0,b=1,c=2;`
- Podle C86 bylo třeba lokální proměnné definovat na začátku funkce, podle C99 lze prakticky kdekoliv (ale není vhodné toho zneužívat).
- Definice globálních proměnných je volně sypaná ve zdrojáku (zdroják sestává především z definic proměnných a funkcí).

Návratová hodnota

funguje jako v Pythonu

- Jazyk C též vychází z toho, že cílem funkce je vrátit návratovou hodnotu, tedy

Návratová hodnota

funguje jako v Pythonu

- Jazyk C též vychází z toho, že cílem funkce je vrátit návratovou hodnotu, tedy
- jak je návratová hodnota jasná, funkce končí.

Návratová hodnota

funguje jako v Pythonu

- Jazyk C též vychází z toho, že cílem funkce je vrátit návratovou hodnotu, tedy
- jak je návratová hodnota jasná, funkce končí.
- Návratovou hodnotu vrací `return...`

Návratová hodnota

funguje jako v Pythonu

- Jazyk C též vychází z toho, že cílem funkce je vrátit návratovou hodnotu, tedy
- jak je návratová hodnota jasná, funkce končí.
- Návratovou hodnotu vrací `return...`
- Příklady: `return;`, `return hodnota;`

Příklad

využívající operátory, o kterých si něco řekneme za chvíli...

V Pythonu:

```
function secti(a,b)
     pom=a+b #Secteme
     return pom #Vratime hodnotu
```

Kdežto v C: `long globpna; //Globalni promenna`

```
int secti(int a, int b)
{    int pom=a+b; //Secteme pri inicializaci
     return pom; //Slo by i return a+b;
}
```

Příklad

Jazyk C

hello.c

```
#include <stdio.h>
int main()
{     printf("Hallo, world!\n");
}
```

Další příklad

Jazyk C

k_nicemu.c

```
#include <stdio.h>
void f()
{     printf("Hallo, world!\n");
}
int main()
{     f();
}
```

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)
- Pozor, přiřazovací operátory vracejí hodnotu. Návratová hodnota je přiřazovaná hodnota a jsou asociativní zprava.

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)
- Pozor, přiřazovací operátory vracejí hodnotu. Návratová hodnota je přiřazovaná hodnota a jsou asociativní zprava.
- Důsledek: `a=b=c=1;` `a=(b=(c=1)+1);`

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)
- Pozor, přiřazovací operátory vracejí hodnotu. Návratová hodnota je přiřazovaná hodnota a jsou asociativní zprava.
- Důsledek: `a=b=c=1;` `a=(b=(c=1)+1);`
- Bitové: $\&, |, \hat{}, <<, >>$ (bitové and, or, xor, shift to left, shift to right např. `(16>>2) == 4`)

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)
- Pozor, přiřazovací operátory vracejí hodnotu. Návratová hodnota je přiřazovaná hodnota a jsou asociativní zprava.
- Důsledek: `a=b=c=1;` `a=(b=(c=1)+1);`
- Bitové: $\&, |, \hat{}, <<, >>$ (bitové and, or, xor, shift to left, shift to right např. `(16>>2) == 4`)
- Inkrementace, dekrementace: $++, --$.

Základní operátory

nejsou uvedeny podle priorit!

- aritmetické: $+, -, *, /, \%$ (součet, rozdíl, součin, podíl, zbytek po dělení),
- logické: $\&\&, ||, !$ (konjunkce, disjunkce, negace)
- relační: $<, >, >=, <=, ==, != $ (menší než, větší než, větší nebo rovno, menší nebo rovno, rovno, nerovno)
- přiřazovací: $=, +=, -=, |= \ldots$ (přiřaď, přičti, odečti, vyoruj...)
- Pozor, přiřazovací operátory vracejí hodnotu. Návratová hodnota je přiřazovaná hodnota a jsou asociativní zprava.
- Důsledek: `a=b=c=1;` `a=(b=(c=1)+1);`
- Bitové: $\&, |, \hat{}, <<, >>$ (bitové and, or, xor, shift to left, shift to right např. `(16>>2) == 4`)
- Inkrementace, dekrementace: $++, --$.
- Pozor na priority!

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`
- `if(podm) prikaz_nebo_blok else prikaz_nebo_blok`

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`
- `if(podm) prikaz_nebo_blok else prikaz_nebo_blok`
- Příklad:

```
if(teplota>25)
    printf("Jdu do hostince\n");
```

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`
- `if(podm) prikaz_nebo_blok else prikaz_nebo_blok`
- Příklad:
```
if(teplota>25)
    printf("Jdu do hostince\n");
```
- `if(teplota>25) printf("Do hostince\n"); else printf("Nikam!");`

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`
- `if(podm) prikaz_nebo_blok else prikaz_nebo_blok`
- Příklad:
```
if(teplota>25)
                printf("Jdu do hostince\n");
```
- `if(teplota>25) printf("Do hostince\n"); else printf("Nikam!");`
- `if(teplota>25) { teplota=teplota-25; printf("Putyka vola!\n");} else printf("Nic nebude!\n");`

Základní řídící struktury `if`

- Podmínky se reprezentují podobně jako v Pythonu, tedy klíčovými slovy `if` a `else`. Syntax se trochu liší.
- `if(podm) prikaz_nebo_blok`
- `if(podm) prikaz_nebo_blok else prikaz_nebo_blok`
- Příklad:
```
if(teplota>25)
                printf("Jdu do hostince\n");
```
- `if(teplota>25) printf("Do hostince\n"); else printf("Nikam!");`
- `if(teplota>25) { teplota=teplota-25; printf("Putyka vola!\n");} else printf("Nic nebude!\n");`
- Pozn: String uzavíráme do uvozovek, znak (typ `char`) do apostrofů.

Pozor!

Podmínky v jazyce C jsou záludné.

- V jayzku C není typ `boolean`. Podmínka se reprezentuje `intem`.

Pozor!

Podmínky v jazyce C jsou záludné.

- V jayzku C není typ `boolean`. Podmínka se reprezentuje `intem`.
- `if(0)...` vždy nesplněná podmínka

Pozor!

Podmínky v jazyce C jsou záludné.

- V jayzku C není typ `boolean`. Podmínka se reprezentuje `intem`.
- `if(0)....` vždy nesplněná podmínka
- `if(11)....` vždy splněná podmínka

Pozor!

Podmínky v jazyce C jsou záludné.

- V jayzku C není typ `boolean`. Podmínka se reprezentuje `intem`.
- `if(0)....` vždy nesplněná podmínka
- `if(11)....` vždy splněná podmínka
- `if(11 && 1)...` vždy splněná podmínka

Pozor!

Podmínky v jazyce C jsou záludné.

- V jayzku C není typ `boolean`. Podmínka se reprezentuje `intem`.
- `if(0)....` vždy nesplněná podmínka
- `if(11)....` vždy splněná podmínka
- `if(11 && 1)...` vždy splněná podmínka
- `if(a=1)...` vždy splněná podmínka

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)
- na `if(a=1)` obvykle přijde warning (že možná nevíme, co děláme).

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)
- na `if(a=1)` obvykle přijde warning (že možná nevíme, co děláme).
- warningu nás obvykle zbaví `if((a=1))`

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)
- na `if(a=1)` obvykle přijde warning (že možná nevíme, co děláme).
- warningu nás obvykle zbaví `if((a=1))`
- Errory jsou předepsány normou, warningy může překladač vydávat dle uvážení.

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)
- na `if(a=1)` obvykle přijde warning (že možná nevíme, co děláme).
- warningu nás obvykle zbaví `if((a=1))`
- Errory jsou předepsány normou, warningy může překladač vydávat dle uvážení.
- Oproti Pythonu (kdy program stále padal po chvíli běhu) v C se na většinu chyb přijde již při překladu.

Warningy a errory

- Errory známe z Pythonu (interpret nám jimi za běhu dává najevo, že jsme něco pokazili),
- jazyk C stanoví, že překladač může varovat (například před jazykovou spodobou)
- na `if(a=1)` obvykle přijde warning (že možná nevíme, co děláme).
- warningu nás obvykle zbaví `if((a=1))`
- Errory jsou předepsány normou, warningy může překladač vydávat dle uvážení.
- Oproti Pythonu (kdy program stále padal po chvíli běhu) v C se na většinu chyb přijde již při překladu.
- Je vhodné kód ladit, dokud není bez warningů.

while

základní řídící struktury 2

- Ovládá se analogicky, tedy
- `while(podm) prikaz_nebo_blok`
- Příklad: `while(teplota-->25)printf("Pivo!\n");`
- Pozor, pokud cyklus neproběhne, proměnná teplota se sníží o jedna (přestože tělo neproběhne)!
- `while` nemá `else` větev.

Cyklus s podmínkou na konci

je v podobě konstrukce `do ... while`

- do ... while(podm);
- Opakujeme, dokud je podmínka splněna, prvně se podmínka vyhodnotí po průchodu cyklem.

`for`-cyklus

je v C výrazně výkonnější nežli v Pythonu

- Trocha historie: Pascal: `for i:=1 to 10 do...`

`for`-cyklus

je v C výrazně výkonnější nežli v Pythonu

- Trocha historie: Pascal: `for i:=1 to 10 do...`
- V Pythonu `for`-cyklus prohrabuje seznam, jinde běží odněkud někam.

`for`-cyklus

je v C výrazně výkonnější nežli v Pythonu

- Trocha historie: Pascal: `for i:=1 to 10 do...`
- V Pythonu `for`-cyklus prohrabuje seznam, jinde běží odněkud někam.
- `for(init;podm;increm)prikaz_nebo_blok`

`for`-cyklus

je v C výrazně výkonnější nežli v Pythonu

- Trocha historie: Pascal: `for i:=1 to 10 do...`
- V Pythonu `for`-cyklus prohrabuje seznam, jinde běží odněkud někam.
- `for(init;podm;increm)prikaz_nebo_blok`
- Příklady: Pascal: `for i:=1 to n do neco`
  C: `for(i=1;i<n;i++) neco`

`for`-cyklus

je v C výrazně výkonnější nežli v Pythonu

- Trocha historie: Pascal: `for i:=1 to 10 do...`
- V Pythonu `for`-cyklus prohrabuje seznam, jinde běží odněkud někam.
- `for(init;podm;increm)prikaz_nebo_blok`
- Příklady: Pascal: `for i:=1 to n do neco`
  C: `for(i=1;i<n;i++) neco`
- Cokoliv může být prázdné (init, increm, tělo i podmínka).

`for`-cyklus

je v C výrazně výkonnější nežli v Pythonu

- Trocha historie: Pascal: `for i:=1 to 10 do...`
- V Pythonu `for`-cyklus prohrabuje seznam, jinde běží odněkud někam.
- `for(init;podm;increm)prikaz_nebo_blok`
- Příklady: Pascal: `for i:=1 to n do neco`
  C: `for(i=1;i<n;i++) neco`
- Cokoliv může být prázdné (init, increm, tělo i podmínka).
- Prázdná podmínka vždy platí.

Faktoriál

už raději jen bez rekurze...

```
int fakt(int a)
{     int fakt;
      for(fakt=1;a>1;a--)
            fakt*=a;
}
```

Faktoriál podruhé

opět raději bez rekurze...

```
int fakt(int a)
{     int fakt=1;
      for(;a>1;fakt*=a--);
}
```

Faktoriál potřetí

opět bez rekurze a ještě zvrhleji...

```
int fakt(int a)
{     int fakt=1;
      for(a++;--a;)fakt*=a;
}
```

Prázdný `for`-cyklus

umí být nečekaně účinný...

```
for(;;);
```

switch

v Pythonu citelně chyběl

Chceme-li udělat mnoho ifů, kdy se ptáme po hodnotě nějakého výrazu:

```
switch(vyraz)
{     case prvni: kod;
      break;
      case druhy: dalsi kod;
      break;
      case treti: jeste;
            dalsi;
            kod;
      break;
      default: co udelat jinak;
}
```

Slovo break

není povinné a jaképak jsou z toho důsledky...

- Neuvedeme-li na konci bloku slovo `break`, kód pokračuje dalším blokem!
- Příklad:

```
switch(den_v_tydnu)
{    case 7: den_v_tydnu-=7;
     case 0: printf("Pondeli\n");
     break;
     case 1: printf("Utery\n");
     break;
     ...
}
```

Načítání vstupu

do doby, kdy pochopíme pointery jen po znacích

- V Pythonu `sys.stdin.read(1)`.
- V C: `int getchar(void)` vrátí ASCII-hodnotu dalšího znaku na vstup.
- Takto budeme načítat čísla. Ale jak?
- Přeci Hornerovým schématem!

Eukleidův algoritmus I

načtení vstupu

```
int main()
{   int a=0,b=0,c,pom;
    while((pom=getchar())>=48&&pom<=57)
        a=10*a+pom-48;
    while(pom<48||pom>57)
        pom=getchar();
    while(pom>=48&&pom<=57)
    {   b=10*b+pom-48;
        pom=getchar();
    }
//Povsimnete si zavorek v podmince
//prvniho while-cyklu!
```

Eukleidův algoritmus II

samotný výpočet

```
    if(a<b)
    {   c=a;
        a=b;
        b=c;
    }
    while(b!=0)
    {   c=a%b;
        a=b;
        b=c;
    }
    printf("%d\n",a);
}
```