# PB071 Úvod do jazyka C

# Přednáška 11

Klíčová slova

C11

Testování

Co dál?

Jiří Weiser

28. dubna 2015

# Organizační

- 11. týden (27. 4. — 3. 5.)
  - normální přednáška
  - normální cvičení

# Organizační

- 11. týden (27. 4. — 3. 5.)
  - normální přednáška
  - normální cvičení
- 12. týden (4. 5. — 10. 5.)
  - zvaná přednáška – Juraj Michálek
  - zápočtový příklad **nanečisto**

# Organizační

- 11. týden (27. 4. — 3. 5.)
  - normální přednáška
  - normální cvičení
- 12. týden (4. 5. — 10. 5.)
  - zvaná přednáška – Juraj Michálek
  - zápočtový příklad **nanečisto**
- 13. týden (11. 5. — 17. 5.)
  - poslední přednáška – Petr Švenda
  - zápočtový příklad **naostro**

# Organizační

- 11. týden (27. 4. — 3. 5.)
  - normální přednáška
  - normální cvičení
- 12. týden (4. 5. — 10. 5.)
  - zvaná přednáška – Juraj Michálek
  - zápočtový příklad **nanečisto**
- 13. týden (11. 5. — 17. 5.)
  - poslední přednáška – Petr Švenda
  - zápočtový příklad **naostro**
- 14. týden (18. 5. — ...)
  - předtermín
  - opravný zápočtový příklad

# Organizační

- zápočtový příklad
  - každé cvičení bude mít jiné zadání
  - hodnocení: dal/nedal
  - 0 bodů

# Organizační

- zápočtový příklad
  - každé cvičení bude mít jiné zadání
  - hodnocení: dal/nedal
  - 0 bodů
- zápočet
  - tolerovány 2 — 3 neomluvené neúčasti
  - 65 bodů a více
  - napsaný zápočtový příklad

# Organizační

- zápočtový příklad
  - každé cvičení bude mít jiné zadání
  - hodnocení: dal/nedal
  - 0 bodů
- zápočet
  - tolerovány 2 — 3 neomluvené neúčasti
  - 65 bodů a více
  - napsaný zápočtový příklad
- zkouška
  - nutné mít zápočet
  - skládá se na počítačích
  - 80 bodů
    - průběžné testíky
    - co vypíše následující kód ...

# Zbývající klíčová slova

# Klíčová slova

## auto

```
void foo() {
    auto int i = 5;
    printf( "%d", i++ );
}
int main() {
    foo(); foo(); foo(); // 555
    return 0;
}
```

- umístí proměnnou na zásobník
  - což je defaultní umístění
- není vhodné uvádět
  - v C++11 má jiný význam

# Klíčová slova

## static – proměnná

```
void foo() {
    static int i = 5;
    printf( "%d", i++ );
}
int main() {
    foo(); foo(); foo(); // 567
    return 0;
}
```

- dá proměnné statickou lokaci
- proměnná je globální
  - ale dostupná jenom ve "své" funkci
- není vhodné používat moc často

# Klíčová slova

## static – funkce, globální proměnná

```
static void foo() {
    ...
}
int main() {
    foo();
    return 0;
}
```

- vytvoří soukromý symbol v rámci překladové jednotky
  - takto označená funkce se nelinkuje s dalšími překladovými jednotkami

# Klíčová slova

## volatile

```
volatile int a = 0;
```

- proměnná může být měněna vnějším prostředím
  - externí HW
  - jiný proces
  - jiné vlákno
- znemožňuje optimalizaci
- není vhodný pro paralelismus

# Klíčová slova

## register

```
for ( register int i = 0; i < 10; ++i ) {
    doSmth( i );
}
```

- realizace proměnné pouze registrem
  - pouze doporučení
  - proměnnou nelze referencovat
- nepoužívat
  - překladač umí optimalizovat mnohem lépe než vy
- v C11 je deprecated

# Klíčová slova

## restrict

```
void foo( int * restrict a, int * restrict b ) {
    ...
}
```

- nepřekrývající se ukazatele
  - větší optimalizace → rychlejší kód
  - možnost nedefinovaného chování
- používat s rozmyslem
  - v C++ není **restrict** klíčovým slovem

# Klíčová slova

## extern

**main.c**

```
extern int i;

int main() {
    return i;
}
```

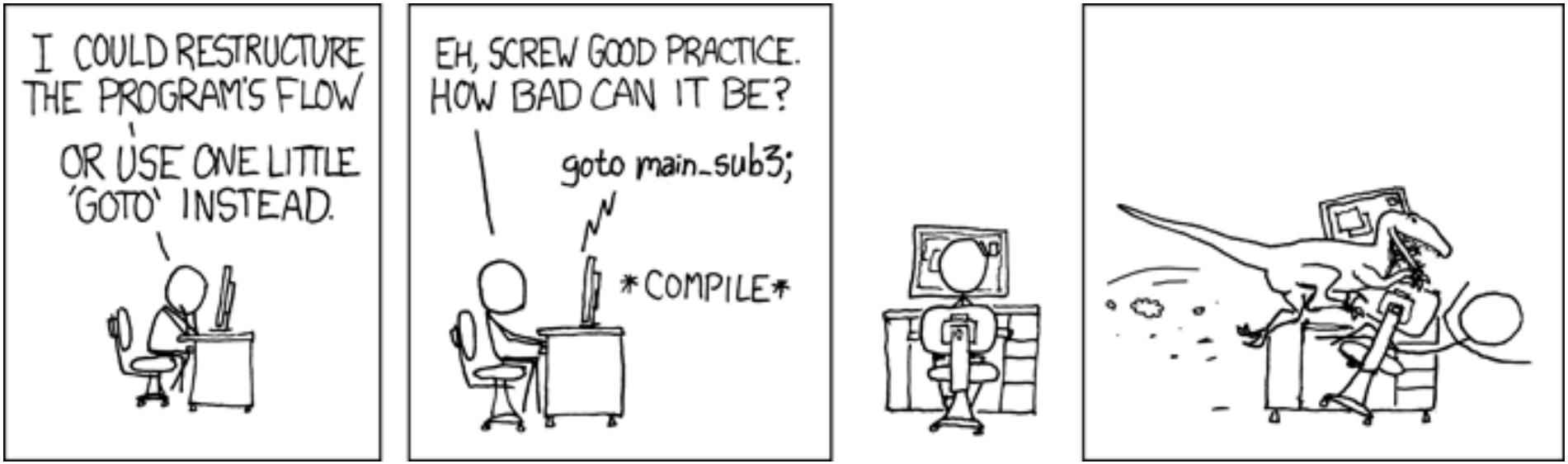
**foo.c**

```
int i = 5;
```

- "hlavička pro proměnné"
- adresa proměnné se zjistí při linkování

# Klíčová slova

## goto



# Klíčová slova

## goto

- problematické
- může být užitečné
- může velmi uškodit

# Klíčová slova

## goto

- problematické
- může být užitečné
- může velmi uškodit

- lze výhodně použít i u vyšších jazyků

# Klíčová slova – goto (špatné použití)

```
int main() {
    int i = 0, j = 10;
    goto L1;
L2:
    if ( i == 6 ) goto L4;
    if ( i == 9 ) goto L3;
    printf( "%i", i );
L4:
    ++i;
L1:
    if ( i < j )
        goto L2;
L3:
    return 0;
}
```

# Klíčová slova – goto (vhodné použití)

```
int main() {
    int *arr = malloc( 6 * sizeof( int ) );
    if ( !arr )
        goto failure;

    while( isOK( arr ) )
        doSmth( arr );
    goto cleanup;

failure:
    fprintf( stderr, "error\n" );
cleanup:
    free( arr );
    return 0;
}
```

# C11

# C11

- revize standardu z 2011
- standardizovány některé vychytávky překladačů
- revidováno s ohledem na C++11
  - podpora vláken
  - podpora atomických operací
  - podpora zarovnání
  - statické asserty
  - podpora Unicode
- spíše kosmetické změny

# C11

## Podpora vláken

- snaha sjednotit podporu vláken napříč platformami
  - ne všude jsou pthreads
- aktuální podpora v podstatě není
  - MSVC, GCC, clang **neumí**

- důležitější z pohledu paměťového modelu
  - standard vůbec poprvé definuje pojem vlákno
  - nové klíčové slovo `_Thread_local`

# C11

```
_Thread_local int counter = 1;

int worker( void *args ) {
    ++counter;
    printf( "worker %i; counter: %i\n",
        *(int*)args, counter );
    return 0;
}

int main() {
    thrd_t w;
    int id = 1;
    thrd_create( &w, worker, &id );
    printf( "main; counter: %i\n", counter );
    thrd_join( w, NULL );
    return 0;
}
```

# C11

## atomické operace

- používají se při komunikaci mezi vlákny
- sjednocení rozhraní překladačů
- operace `i++` není atomická
  - skládá se z: načtení, inkrementu, uložení
- mnohé další operace lze udělat atomicky

# C11

## atomické operace

- používají se při komunikaci mezi vlákny
- sjednocení rozhraní překladačů
- operace `i++` není atomická
  - skládá se z: načtení, inkrementu, uložení
- mnohé další operace lze udělat atomicky

- není vhodné pro normální proměnné
  - vyžaduje synchronizaci na procesoru
  - případně mezi procesory

# C11

## Podpora zarovnání

- zarovnaná paměť → rychlejší výpočet
- sjednocení rozhraní překladačů
- klíčová slova **_Alignof** a **_Alignas**
- používají se makra
  - alignof
  - alignas

---

```
struct Data {
    char c;
    alignas( 16 ) char array[ 16 ];
};
// sizeof( struct Data ) == 32
// alignof( struct Data ) == 16
```

# C11

## Unicode a static_assert

- zavedeny nové typy pro UTF-16 a UTF-32
  - **`char16_t`** a **`char32_t`**
- podpora prefixů ‘u’, ‘U’ a ‘u8’

# C11

## Unicode a static_assert

- zavedeny nové typy pro UTF-16 a UTF-32
  - **char16_t** a **char32_t**
- podpora prefixů ‘u’, ‘U’ a ‘u8’

- **static_assert** – kontrola při překladu
  - součást jazyka C, nikoliv preprocessoru

---

```
static_assert(
    sizeof( long ) == 8,
    "no way bro" );
```

# C11

## Další novinky

- možnost exkluzivně vytvořit soubor
  `FILE *f = fopen( "path", "wx");`

# C11

## Další novinky

- možnost exkluzivně vytvořit soubor
  `FILE *f = fopen( "path", "wx");`

- typově generický výraz
  `_Generic( X, int: fooI, default: foo)(X)`

# C11

## Další novinky

- možnost exkluzivně vytvořit soubor
  `FILE *f = fopen( "path", "wx");`
- typově generický výraz
  `_Generic( X, int: fooI, default: foo)(X)`
- nepojmenované struktury/uniony uvnitř struktur/unionů

```
struct T {
    int tag;
    union { int i; float f; };
};
```

# Testování

# Testování

**Různé druhy testování**

- manuální, automatické
- unit testování
  - elementární komponenty
  - takto si testujete vaše úlohy

# Testování

**Různé druhy testování**

- manuální, automatické
- unit testování
  - elementární komponenty
  - takto si testujete vaše úlohy
- integrační testování
  - spolupráce více komponent
  - test dodržení rozhraní

# Testování

**Různé druhy testování**

- manuální, automatické
- unit testování
  - elementární komponenty
  - takto si testujete vaše úlohy
- integrační testování
  - spolupráce více komponent
  - test dodržení rozhraní
- systémové testy
  - testy celého programu
  - ověření chování vůči specifikaci

# Testování

## Unit testování

**Jak na to?**

# Testování

## Unit testování

**Jak na to?**

- vybrat funkce, které testovat
  - (ideálně všechny)

# Testování

## Unit testování

**Jak na to?**

- vybrat funkce, které testovat
  - (ideálně všechny)
- pro každou funkci napočítat scénáře

# Testování

## Unit testování

**Jak na to?**

- vybrat funkce, které testovat
  - (ideálně všechny)
- pro každou funkci napočítat scénáře
- každý scénář implementovat jako samostatný test
- kontrolovat validitu pomocí assertů

# Testování

## Unit testování

**Jak na to?**

- vybrat funkce, které testovat
  - (ideálně všechny)
- pro každou funkci napočítat scénáře
- každý scénář implementovat jako samostatný test
- kontrolovat validitu pomocí assertů
- každý test vhodně pojmenovat

# Testování

## Unit testování

**Jak na to?**

- vybrat funkce, které testovat
  - (ideálně všechny)
- pro každou funkci napočítat scénáře
- každý scénář implementovat jako samostatný test
- kontrolovat validitu pomocí assertů
- každý test vhodně pojmenovat
- opakovaně spouštět všechny testy při jakékoliv změně kódu

# Testování

## Unit testování

**To nám ale nestačí!**

- testy by se měly spouštět nezávisle na ostatních testech
  - jeden test by neměl poškodit paměť pro ostatní testy
- test, který skončí segfaultem, by neměl shodit testování
- chceme spouštět jenom některé testy
- chceme mít statistiku testů

# Testování

## Unit testování

**To nám ale nestačí!**

- testy by se měly spouštět nezávisle na ostatních testech
  - jeden test by neměl poškodit paměť pro ostatní testy
- test, který skončí segfaultem, by neměl shodit testování
- chceme spouštět jenom některé testy
- chceme mít statistiku testů

- $\rightarrow$ každý test spustit jako samostatný proces
  - zde končí multiplatformnost
- $\rightarrow$ rozšířit spouštění testů o výběr

# Testování

**Pár poznámek na konec**

- pište si testy (i když je to otrava)
- nejlépe použijte nějaký framework
  - ale jde to i bez něho
- testujte pomocí assertů
  - ale třeba také kontrolujte výstupy na obrazovku
- nebojte se dát asserty i do výkonného kódu
  - může výrazně ušetřit čas pro tvorbu testů
  - zvýší se pokrytí kódu testy
  - v release režimu nebude assert přítomný

# Co dál?

# Co dál?

## předměty

- Tématicky vývoj aplikací v C/C++ (PB173)
  - mnoho skupin: systémové programování (Linux, Windows), ovladače Linuxu, Zpracování obrazu, Aplikovaná bezpečnost
  - nově běží každý semestr
  - lze zapisovat opakovaně
- Programování v jazyce C++ (PB161)
- Úvod do vývoje v C#/.NET (PV178)
- Programování v jazyce Java (PB162)
- Paralelní technické systémy (PV192)
  - zajímavý, poměrně náročný předmět
- Projekt z programování paralelních aplikací (IV112)
  - zajímavý, dost náročný předmět
- seznam programovacích předmětů na FI

# Co dál?

## laboratoře, projekty

- laboratoře na škole
  - zajímavá témata
  - snadný výsledek v podobě bakalářské práce
- open source projekty
  - zkuste odstranit reportovanou chybu
  - zkuste implementovat nějakou TODO funkčnost
- brigáda/praxe ve firmě
- nebát se jiných jazyků