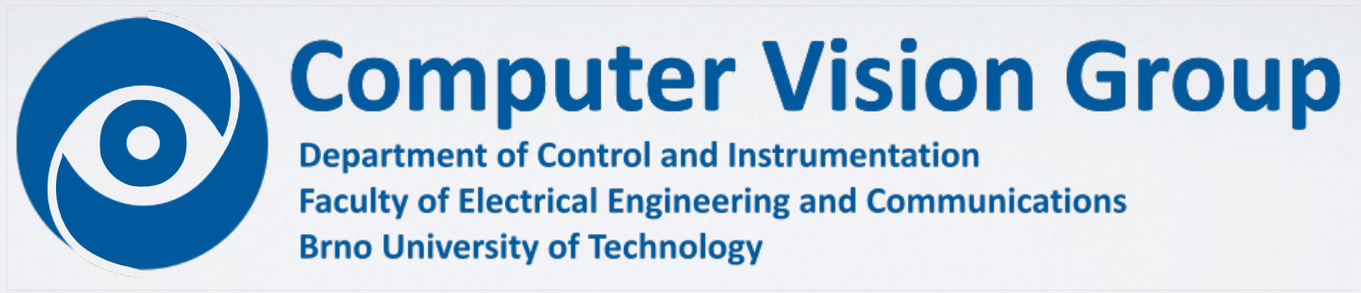

Q4 2010

PŘEDNÁŠKA KURZU BPC2A

Rozšíření jazyka C, ISO C99, ISO C11

M. Richter (email: richter@feec.vutbr.cz)
P. Petyovský (email: petyovsky@feec.vutbr.cz)



Standardní hlavičkový soubor matematické knihovny `math.h`
Definice nových typů pomocí **`typedef`**

Hlavní rozšíření jazyka v normě C99
Inline funkce
Datový typ: `_Bool`
Datový typ: `_Complex`
Modifikátory proměnných
`const`
`volatile`
`restrict`
Hlavní rozšíření jazyka v normě C11 (ISO/IEC 9899:2011)

Programovací styly
Defenzivní programování
Nástroje pro pomoc při vytváření programů / projektů
Dotazy

Standardní hlavičkový soubor matematické knihovny `math.h`

-makrem `FLT_EVAL_METHOD` lze nastavit přesnost (interních) výpočtů pro **`float`**, **`double`**
-makra `INFINITY`, `NAN` (`isfinite`, `isinf`, `isnan`) dle IEEE754
-makra pro řešení chybových stavů
-trigonometrické funkce – `acos` (`acosf`, `acosl`), `asin`, `atan`, `atan2`, `cos`, `sin`, `tan`, hyperbolické verze, `exp`, `exp2`, `log10`, `log` (základ e), `log2`
-`frexp` – rozdělí reálné číslo na normalizovanou mantisu a exponent
-`fabs`, `pow`, `hypot` (odmocnina ze sumy kvadrátů), `sqrt`,
-`ceil`, `floor`, `round`, `trunc`, `remainder` práce s desetinnou částí čísla

Definice nových typů pomocí **`typedef`**

-klíčové slovo **`typedef`** umožňuje nadefinovat nový typ,
-nový typ je odvozen od typu stávajícího a je z/na něj konvertovatelný,
-dále je výhodné jej použít při složitých definicích jako například při použití ukazatelů na funkce,
-používá se v případě, že chceme některý typ měnit, například máme řadu funkcí, které mají pracovat pouze s jedním typem, ale až při překladu se chceme rozhodnout s jakým

```
//nový typ vzniklý odvozením z int*
typedef int* NOVY_TYP;
// následné použití je stejné jako u jiných typů

NOVY_TYP funkce(NOVY_TYP aa, NOVY_TYP *bb)
    {… NOVY_TYP a = NULL, b = NULL, c = NULL;
      // všechny tři proměnné jsou typu int *
    return aa;
    }
```

Hlavní rozšíření jazyka v normě C99
(ne vše je v překladači VS2013 implementováno, M$ má asi stále rok 1995)

- pokud není uveden typ, již se nepředpokládá, že je to `int` (vznikne warning)
- `inline` funkce
- proměnné nemusí být definovány na začátku bloku (pravidlo minimalizace neinicializovaných proměnných - proměnná vzniká až v okamžiku, kdy víme co jí přiřadit)
- Nové typy `long long int` - minimálně 64 bitů, knihovní `bool` a `complex`
- pole proměnné délky (automatická proměnná s délkou danou proměnnou) - v C11 změněno na nepovinnou součást normy
- řádkové komentáře //
- nové hlav. soubory (`stdbool.h`, `complex.h`, `tgmath.h`, `inttypes.h`)
- alternativní pohled na celočíselné typy: `int8_t,uint8_t stdtypes.h`
- generická makra, variadická makra
-IEEE floating point / fixed point aritmetika
-možnost psaní proměnných v "neanglických" znacích – (pokud možno nevyužívat)

- *designated initializers:*

```
int a[6] = { [4] = 29, [2] = 15 };
struct point {int x; int y;} = { .y = 10, .x = 20};
```

-*compound literals:*

```
struct foo {int a; char b[2];} var;
var = ((struct foo) {x + y, 'a', 0});
```

- *restrict qualification* - umožňuje optimalizace práce s ukazateli (viz. dále)

- je-li definováno makro `__STDC__` je překladač C90
je-li definováno `__STDC_VERSION__` s hodnotou `199901L` je možné používat vlastnosti C99

Inline funkce

- **`inline`** je klíčové slovo jazyka
- „nejrychleji možné volání“, rozvinutí do kódu (obdoba makra s parametry)
- typová kontrola parametrů (a z toho plynoucí případné implicitní konverze)
- není nutné „ozávorkovat“ parametry
- netvoří kód -> je v hlavičkovém souboru. Je to předpis pro vytvoření kódu

```
inline int deleno2(double a) {return a/2;}
int bb = 4;
double cc;
cc = deleno2(bb); // v tomto místě překladač
// vloží (něco jako) cc = (int)(double(bb)/2);
```

Datový typ: `_Bool`

-Celočíselný datový typ s nejmenším rozsahem
-knihovnou a s podporou překladače definovaný typ pro logické proměnné společně s konverzemi
-definována makra `bool` (`_Bool`), `true` (1), `false` (0)
-Konstanty konvertovány na 0 a 1
-zajišťuje hlav. soubor: `stdbool.h`

Datový typ: `_Complex`

-knihovna pro práci s komplexními čísly včetně nejčastěji používaných funkcí
-tři verze: pro typ `float`, `double` a `long double`. `float _Complex`, `double _Complex`, `float complex`, `double complex`
-knihovna nemusí být implementována (zjištění makrem `__STDC_IEC_559_COMPLEX__`, někdy i `__STDC_NO_COMPLEX__`)
-definice `_Complex_I` (konstantní), `_Imaginary_I` – komplexní číslo pouze s imaginární částí.
-hlavičkový soubor: `complex.h`
-funkce `cacos` (ComplexArcCOSinus), `cacosf`, `cacosl`, `casin`, `catan`, `ccos` (ComplexCOSinus), `csin`, `ctan`, `cacosh`, `ccosh`, ... ,
-`cexp` (base e), `clog` (base e),
-`cabs`, `cpow`, `csqrt`, `conj`,
-`carg`, `cimag`, `creal`,

```
float _Complex cislo = 2.0f + 2.0f*_Complex_I;
```

Modifikátory proměnných

-v jazyce C jsou nejčastěji používány modifikátory **`const`**, **`volatile`**, **`restrict`**

-modifikátor **`const`** slouží k oznámení, že se proměnná nebude měnit a je to kontrolováno překladačem

-modifikátor **`volatile`** slouží k oznámení, že proměnná může být měněna (asynchronně) aniž by to bylo z kódu zřejmé

-modifikátor **`restrict`** říká, že proměnná typu ukazatel (všechny proměnné přístupné pomocí této proměnné/ukazatele) je jediná, pomocí které se manipuluje s danými daty

```
const int KMax = 10;
KMax++;      // Tohle nepůjde ani přeložit
```

`volatile` *proměnná*

-modifikátor, který v definici proměnné upozorňuje překladač, že proměnná může být asynchronně (na pozadí) měněna (například v přerušení)
-překladači je zakázáno předpokládat "neměnnost" proměnné, tj. při každém použití znovu načítá její obsah z paměti,
-takováto proměnná nemůže být optimalizována/ukládána do registru, protože druhý proces ji mění v paměti a tak musí být vždy používána z paměti

```
volatile int flag = 1; // definice globální proměnné
……
  { // uvnitř přerušovací rutiny
  flag = 0; // nastavení, že přišla událost na kterou se čeká v hlavní programu
  }
……
  {
  while(flag); // nekonečný cyklus.
  // Ukončí ho přerušení, které změní hodnotu proměnné.
  // pokud bude proměnná v registru, přerušení ji změní v paměti, k ukončení nedojde
  }
```

`restrict` *proměnná*

-modifikátor spojený s definicí ukazatele
-tento modifikátor říká, že daný ukazatel je jediný určený pro „aktivní“ přístup (zápis) k daným datům v daném okamžiku.
-prototyp funkce informuje uživatele, jaké relace mají mít paměťové bloky. Je-li uvedeno klíč. slovo **`restrict`**, potom to například znamená, že se nekontroluje překrytí dvou prostorů na které ukazují různé restringované ukazatele a s tím vzniklé kolize zapříčiněné přepisováním dat ve špatném pořadí.
-zároveň se jedná se o informaci, že přístup k datům může překladač provádět libovolně (optimalizovat na rychlost, měnit pořadí přístupu a to bez kontroly kolizí
-splnění této podmínky je závazkem pro programátora a není překladačem kontrolováno

```
int memory_move(restrict char *aSrc, restrict char *aDst)
```

definice říká, že `aSrc` a `aDst` jsou jediné, které ukazují na své data. Jinými slovy ale také, že blok dat daný `aSrc` a `aDst` se nepřekrývá a není tedy nutné řešit přesun tak, aby nedošlo k překrytí dat, která budeme ještě potřebovat, daty přesouvanými.

Hlavní rozšíření v normě C11 (ISO/IEC 9899:2011)

-*multi-threading* (vlákna) a „příslušenství“ (mutex, ukládání a manipulaci s proměnnými)
-některé vlastnosti C99 přesunuty do volitelných součástí jazyka
-zarovnání proměnných (alignment) a alokovaných prostorů
-*type-generic* výrazy umožňující reagovat na typ proměnné (v makrech) – klíčové slovo `_Generic`
-vylepšení práce s *Unicode* – typy `char16_t` a `char32_t` pro manipulaci s UTF-16 a UTF-32. V headeru `uchar.h` jsou i funkce pro práci s těmito typy
-funkce `gets` nahrazena za funkci `gets_s`
-*static assertions* – vyčíslení `#if` a `#error` již při překladu (pokud je to možné)
-při otevírání souboru další příznak „`x`“ – pro přístupová práva k souboru.
-Makro `__STDC_VERSION__` je definováno jako `201112L`

Programovací styly

Z hlediska možností jazyka
-strojový jazyk – jednoduchý, omezený počet příkazů
-strukturované programování – využití struktur/programových bloků
-objektově orientovaný návrh

Z hlediska struktury kódu
-pravidla pro psaní programu, funkcí, názvů proměnných, grafické struktury
-má usnadnit orientaci v cizích textech
-je závislá na konkrétním programovacím jazyku
-definovaný osobně, komunitou, firemními doporučeními (např. MISRA C)

Definice stylu může obsahovat:

-pravidla rozdělení do zdrojových souborů
-formát souborů (hlavička, obsah …)
-komentáře – co a jak se komentuje
-grafický styl programu: kolik příkazů na řádek, jak a kdy odsazovat začátky řádků, kde definovat proměnné, použití mezer či tabelátorů, prázdných řádků
-pravidla pro vytváření názvů proměnných, funkcí …
-nedoporučené, nebo zakázané součásti jazyka

Defenzivní programování

-programátor vytváří program s „vědomím“ , že okolí ho bude někdo „napadat“
-předpokladem je, že může nastat jakákoli varianta (vstupů, běhu programu) a je tedy nutné tyto stavy identifikovat a program na ně už předem připravit
-*sanitizace* veškerých vstupů, *“tainted”* režim chodu programu

-součástí je i řešení těchto stavů tak, aby program nemusel být ukončen,
-důležitou součástí je ochrana a kontrola dat, které program „vlastní“
-korektní správa sdílených zdrojů (paměť, síťová spojení, grafické kontexty…)

-metody tvorby programu: důkladný rozbor, trvání na stanovených pravidlech, (znovu)použití odladěného kódu, testování, regrese ve funkčnosti a testech
-důležité je využití testerů, které nemají ponětí o kódu
-testy mohou být prováděny i osobami, vstupními soubory, nebo programy
-existují knihovny pro automatické testování

Nástroje pro pomoc při vytváření programů / projektů

Doxygen
-pomocí formátovacích znaků uvnitř standardních komentářů zdrojového textu lze pomocí tohoto programu vytvořit dokumentaci
-výsledkem je html, pdf kód s vazbami mezi stránkami, proměnnými, soubory…
-automaticky jsou vybírány funkce, proměnné, jsou popisovány vazby mezi proměnnými a soubory …

SVN (GIT, Mercurial)
-*Subversion* – nástroj pro spravování verzí souborů se zdrojovými texty
-na serveru existuje „aktuální“ verze zdrojových textů
-jednotliví autoři si stáhnou tuto verzi a nezávisle na druhých vytváří/mění kód
-po odladění menších celků jsou tyto nahrány zpět na server. SVN umí spojit jednotlivé verze tak, že může pracovat současně více autorů
-doplňkové nástroje pro sledování změn (pamatuje si historii), vývoje, autorství…
-možnost vývojových „větví“ , možnost vracet se ke starším verzím

ODKUD JDEME A KAM...

DOTAZY?

DĚKUJI ZA POZORNOST