# Kompilace

* **Prepocesor**: include, # makra (define), smazání komentářů, spojení řádků místo znaku „/“
  + Hlídá, zda nebyla nějaká část dosazena vícekrát
  + Lze spustit samostatně: přidáme přepínač -E při kompilaci
* **Kompilace**: zdrojový kod do binární podoby (.o)
  + Není spustitelný – nejsou vyřešeny závislosti s jinými propojenými soubory
* **Linkování**: vkládání absolutních hodnot adres (proměnné a funkce)
  + Výsledek: spustitelný soubor
* Překladače jazyka C: **gcc, clang**, tcc, icc

# Knihovny

* **Standardní knihovny**
  + Stdio.h (fce pro vstup a výstup), math.h (matematické funkce), stdbool.h, stdlib.h (k alloccům), string.h (fce pro práci se stringama)…

errno.h: na začátku errno = 0, error number najde chybu a vypíše její číslo

assert.h: umožňuje zápis diagnostických informací na stderr (🡪 zjednodušení debugování)

* + **Dynamické linkování** (= knihovna zavolána v momentě spuštění)
* **Mé vlastní knihovny**
  + **Statické linkování** (zkompilováno samostatně, spojeno při spuštění)
* #include **<**stdbool.h**>** 🡪 prohledá **jen standartní knihovny** linuxu
* #include **"**queue\_e.h**"** 🡪 prohledá **standartní knihovny linuxu a složku** ve které je soubor
* Zahrnutím knihovny stdbool.h můžeme používat bool a = true

# Funkce

* **Deklarace**: říkáme že **fce existuje**; **jméno fce, argumenty a návratová hodnota**
* **Definice**: celá fce; v { }
* Název: nezačínat číslem, obsahuje písmena, čísla a '\_', neshoduje se s klíčovým slovem, omezená délka
* Návratová hodnota: void – fce nemá návratovou hodnotu
* int main(int argc, char\* argv[]) {…; return 0;}
* goto – lze použít pro vyskočení z vnořených cyklů; syntaxe: goto LABEL; 🡪 LABEL:
  + nelze vyskočit z funkce někam do mainu třeba – lze použít setjmp() a longjmp() místo toho
    - tyto fce slouží ke skokům mezi funkcemi
* Chybový výstup: -1 nebo NULL nebo EOF

# Operátory

* Logické operátory: && (AND), || (OR), ! (Negation)
* Binární operátory: >> (right shift), <<, & (AND), | (OR), ^ (XOR), ~ (inverse, NOT)
* Násobit dvěma: i<<1,dělení dvěma: i>>1

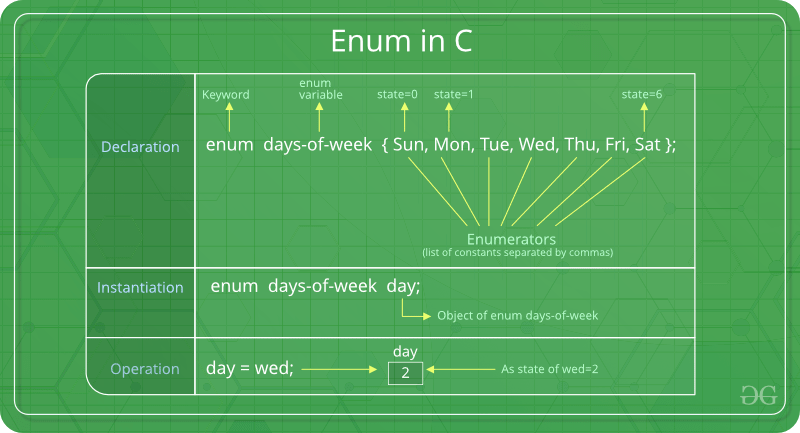
# Datové typy a další

* Datové typy: Celočíselné: int, long, long long, short, signed char
  + Neceločíselné: float a double: vždy: double 2x větší než float, zpravidla: 32 a 64 bitů
    - Vnitřní reprezentace: sign, exponent, mantisa
* Velikost datového typu můžeme zjistit pomocí sizeof(typ);
* String je pole charů, o 1 delší než délka vkládaného textu (poslední char: '**\0**')
* NULL – použití: prázdný pointer
* const – neměnná hodnota (konstanta), hodnotu lze změnit pomocí pointeru
* Přetypování: syntaxe: (float) promenna\_double
* Paměťové třídy:
  + static – proměnná je globální – je dostupná z daného modulu, je zachována i po opuštění funkce
  + auto
  + extern – proměnná viditelná ve všech modulech
  + register – proměnná může být umístěna do registru 🡪 rychlejší přístup

## Pole

* Předáváno funkcím jako ukazatel na první položku pole
* int pole[3][2] = { { 1, 2 }, { 3, 4 }, { 5, 6 } };
* int\* pole = malloc(20\*sizeof(int)); //velikost: 20\*4 bajtů
* int\* nove = realloc(pole, 30\*sizeof(int));

# Složené typy

* Struct: umožnuje kombinovat datové položky různých typů
  + Definování struktury: struct {int vek; char a} my\_struct;
  + Použití: struct my\_struct people;
  + Urychlení: **typedef** struct my\_struct {int vek; char a}; 🡪 my\_struct people;
  + Přístup: people.vek = 20; když ukazatel: people->vek
  + Kompilátor alokuje paměť pro každého člena (na rozdíl od unionu)
  + Velikost struktury >= součet velikostí položek struktury
* Union: pouze jeden člen může v daný moment obsahovat hodnotu
  + Velikost unionu = velikost největší položky; šetří paměť
  + Syntaxe jako u structu (např.: typedef union {int a; char b;} spolecna\_promenna;
* Enum
  + Syntaxe: enum week {Mon, Tue, Wed}; 🡪 enum week day; nebo enum week {Mon, Tue, Wed} day;
    - day = Wed; 🡪 day má hodnotu 2

# Paměť a velikosti

* Stack – staticky alokovaná pamet; po dokončení funkce se automaticky smaže
  + lokální proměnné, funkce deklarované v main,
* Heap – dynmicky alokovaná pamet, allocy, je třeba uvolňovat – free();
* Stack a Heap v RAM

# Znaky pro řízení vstupu/výstupu

* **%d, %i = celé číslo** (%u: bez znaménka, %o: v osmičkové soustavě, %x, %X: v šestnáctkové)
* **%f = float/double** (%e, %E, %g, %G: racionální číslo s exponentem)
* **%c = jeden znak; %s = řetězec; %p = pointer**

# Ukazatel (pointer)

* Velikost: dle architektury počítače, zpravidla 64-bitový systém používá 64 bitů, 32-bitový 32
* int\* p = &a;
* Tisk ukazatele (p): printf("%p", p), tisk hodnoty (\*p): printf("%d", \*p)

# Cykly a switch

* for(A = inicializace kontrolní proměnné, cokoliv co se má vykonat při vstupu do cyklu; B = podmínka;  
   C = aktualizace kontrolní proměnné, cokoliv co se má vykonat po každé iteraci)
* **do**: v **do-while** cyklu (do {…} while (…) ) – proběhne aspoň jednou
* **break**: opuštění cyklu nebo switche
* **continue**: skok na začátek cyklu + zvýšení itarace
* **case**: označuje jednu možnost v příkazu switch
* **default**: případ kdy ani jeden case nevyhovuje

# Práce se soubory

* Otevření soboru pro čtení: FILE\* soubor = fopen("text.txt", **"r"**); pro zápis: …**"w"**
* Nakonec: **fclose();** (k uvolnění paměti)
* Nalezení konce v soboru – správně: feof();

# Vlákna (threads)

* **Samostatně prováděné výpočetní toky**, paralelně
* i na jednom procesoru může běžet více vláken
* **Modely** vícevláknových aplikací: **pipeline** (nejlepší pro zpracování proudu dat), boss/worker, peer, producer/consumer
* Proces = instance programu který je spuštěn oper. systémem a běží ve vyhrazeném prostoru paměti
* **Primitiva pro synchronizaci více vláken**: základní: **pthred\_mutex\_lock + unlock**

dále: pthread\_cond\_signal, pthread\_cond\_wait, pthread\_cond\_broadcast

* **Thread-safe** funkce: lze vykonat část kódu bez způsobení synchronizačních problémů
  + Uděláme použitím synchronizačních primitiv
* **Reentrantní** funkce: může být v přerušena a bezpečně znovu zavolána
  + Nepíše do static
  + data, nepracuje s globálními proměnnými
* Synchronizační problémy
  + **Deadlock**: Pro dokončeni jedné operace je potřeba dokončit druhou – zacyklení se
  + **Race condition**
    - Více vláken přistupuje ke sdílenému zdroji (min. jedno z nich nepoužívá synchronizační mechanismus)
    - Jedno vlákno čte hodnotu, druhé zapisuje – může dojít k tomu, že mezi zápisem a čtením dojde k přepsání
    - Prevence (většinou): používat mutexy vždy ve stejném pořadí