Struktury a unie

další věc známá z Pascalu

- V Pascalu byl datový typ `record` a dal se do něj vložit `case`.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.
- Použití (definované) struktury:
  `struct spojak * hlava;`
- Prvky struktury jsou v paměti reprezentované za sebou, prvky unie přes sebe.
- Protože je otrava pořád psát `struct spojak`, můžeme definovat vlastní typ:
  `typedef int integer;`
- anebo `typedef struct pom_spojak{int hod; struct pom_spojak*next;} spojak;`
- A pak: `spojak hlava;`

Přístup do spojáku

a zatracené priority operátorů

- Do struktury se přistupuje jako v Pascalu (operátor tečky).
- Pointer se dereferencuje (unární prefixní) hvězdičkou.
- ale ve výrazu `*a.hod` má vyšší prioritu tečka (než hvězdička).
- Takže `(*a).hod`...
- ... nebo `a->hod`.

Zde udělat příklad na spoják

Definovat strukturu a napsat funkce `pridej` a `uber`.

Práce se soubory

je také podobná jako v Pascalu, jenom ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`
- Režim může být zejména: `r, w, a, r+, w+, a+`
- Použijeme "a" nebo "a+" chceme-li volat `fseek, fsetpos, rewind`.
- Chceme-li soubor v binárním režimu, přidáme znak `b`:
  `"rb", "r+b", "rb+"` – binární režim se stará o konce řádků.
- C11 zavádí ještě "x" k režimu "w"...
- ... vybouchni, pokud soubor už existuje.

Soubory II

zavřít soubor jde rychleji než otevřít

- `int fclose(FILE*);`
- `int feof(FILE*);`
- `int fgetc(FILE*)`
  `char*fgets(char*s,int pocet,FILE*)`
- `fgetc, fputc, fputs, fprintf`
- `fscanf` formátované načítání (je zrádné).

Ternární operátor

- `if(a>b)c=a; else c=b;`
- alternativně `c=(a>b?a:b);`
- Operátor sestává z otazníku a dvojtečky.
- Před otazníkem podmínka, před dvojtečkou hodnota v případě splněné podmínky, za otazníkem v případě nesplněné.
- Oceníme při povídání o makrech.

Preprocessor

a jeho schopnosti

- Překlad probíhá jednoprůchodově zleva doprava.
- Porůznu chceme kontrolovat správnost volání funkcí,
- občas chceme pracovat s konstantami,
- jindy by se nám hodilo něco jako makro,
- každou chvíli chceme kus kódu použít jen za nějakých okolností (známých při kompilaci).

Preprocessor

zase

- je prvním automatem zpracovávajícím zdrojový kód,
- po něm se teprve rozjíždí jádro překladače,
- jeho výstupem je čitelný kód,
- mimo jiné includuje hlavičkové soubory.

Preprocessor

ještě pořád

- V hlavičkových souborech jsou především hlavičky funkcí
- a definice konstant (a maker).
- Preprocesor reaguje na znak křížek (#)
- za křížkem následuje direktiva preprocesoru.
- include, define, if, ifdef, ifndef, else, endif, undef.

Preprocessor

pořád nám nedá pokoj

- `#include<stdio.h>` – známe, do zdrojáku vloží obsah daného souboru.
- `#undef ID` – oddefinuje symbol ID.
- `#if, #else, #endif` – compile-timeové podmínky (kód se ve zdrojáku nechá jen pokud je podmínka splněna (resp. nesplněna).
- `#ifdef, #ifndef` – speciální případy `#if: #if defined ID`.
- `#define NULL 0` – definuje například konstanty.
- Všechny výskyty symbolu NULL budou při preprocessování nahrazeny nulou.

Direktiva define

definuje nejen konstanty

- umožňuje definovat makra.
- Makro říká, co se čím má nahradit.
- Motivace: Chci funkci pracující s různými datovými typy (třeba počítající maximum).
- Příklad: `#define max(a,b) (a>b?a:b)`
- Použijeme: `x=max(10,15);`
- Ale pozor: `x=max(atoi(gets(a)),atoi(gets(a)));`
- A také pozor na priority operátorů!
- Makra voláte častěji, než myslíte!

Proměnlivý počet parametrů

zvládne v jazyku C každý trouba

- `int printf(const char *,...);`
- typ `va_list` makra `va_start, va_arg, va_end`
- `va_start(va_list list,last_arg)` – zinicializuje `va_list`, `last_arg` je poslední pevný argument.
- `va_arg(list,typ)` – vrátí další argument typu `typ`.
- `va_end(list)` ukončí práci s daným seznamem.

Příklad

názorný

```
void tiskni(int kolik,...)
{    va_list l;
     va_start(l,kolik);
     for(int i=0;i<kolik,i++)
          printf("%d ",va_arg(l,int));
     va_end(l);
}
```