Súbory - pokračovanie
Funkcie

Základy procedurálneho programovania 1, 2020

Ing. Marek Galinski, PhD.

Opakovanie

2

Práca so súborom v C

- **Základný dátový typ: `FILE *`**

- Pointer (ukazovateľ) na object typu FILE
  - Zapísaná adresa, kde začína súbor na disku
  - Case sensitive

- Definícia premennej f pre prácu so súborom (môže ich byť viac)

```
FILE *f;
```

```
FILE *fr, *fw;
```

Práca so súborom v C

- **Otvorenie súboru v C**

- Na čitanie

```
fr = fopen("pokus.txt", "r");
```

- Na zápis

```
fw = fopen("pokus.txt", "w");
```

- Aj ďalšie režimy …
- Binárny súbor – rb, wb

Práca so súborom v C

- **Testovanie správnosti otvorenia súboru**

- Čo ak otváram neexistujúci súbor? Čo ak nemám práva čítať súbor?

- **`fopen()`**
  - V úspešnom prípade vráti ukazovateľ na súbor
  - V neúspešnom prípade vráti NULL (0)

```
if((fr = fopen("test.txt", "r")) == NULL)
   printf("Subor sa nepodarilo otvorit.\n");
```

Súbory - pokračovanie
Funkcie

Základy procedurálneho programovania 1, 2020

Ing. Marek Galinski, PhD.

Súbory - pokračovanie

7

Rôzne režimy práce so súborom

- **Stále používame na otvorenie `fopen()`**

r	textový súbor pre čítanie
w	textový súbor pre zápis alebo pre prepisovanie
a	textový súbor pre pripojenie na koniec
r+	textový súbor pre čítanie a zápis
w+	textový súbor pre čítanie, zápis alebo prepisovanie
a+	textový súbor pre čítanie a zápis na koniec

Rôzne režimy práce so súborom

- **Stále používame na otvorenie `fopen()`**

požiadavky / režim otvorenia	"r"	"w"	"a"	"r+"	"w+"	"a+"
súbor musí existovať	+			+		
existujúci súbor bude vymazaný		+			+	
existujúci súbor bude rozšírený			+			+
neexistujúci súbor bude vytvorený		+	+		+	+
čítať - z ľubovoľného miesta v súb.	+			+	+	+
zapisovať - na ľubovoľné miesto v súb.		+		+	+	
zapisovať - iba na koniec súb.			+			+

Vrátenie prečítaného znaku do bufferu

- **Ak prečítame o jeden znak viac, ako sme potrebovali, dá sa vrátiť**

- Ako vrátiť prečítaný znak do bufferu?

- **`ungetc(c,subor)`**
  - V úspešnom prípade vráti znak
  - V neúspešnom prípade vráti EOF

```
int c, hodnota = 0;

while ((c = getchar()) >= '0' && c <= '9') {
  hodnota = hodnota * 10 + (c - '0');
}
ungetc(c, stdin);
```

Vrátenie prečítaného znaku do bufferu

- **Ak prečítame o jeden znak viac, ako sme potrebovali, dá sa vrátiť**

```
while ((c = getc(fr)) != EOF) {
    if (c >= '0' && c <= '9') {
        ungetc(c, fr);
        fscanf(fr, "%lf", &x);
        sum += x;
    }
}
printf("Sucet cisel v subore: %f", sum);
```

Nastavenie sa na pozíciu

- **V súbore vieme nastaviť pozíciu ”kurzora” - odkiaľ čítame, kam chceme zapisovať**

```
int fseek(FILE *stream, long offset, int whence)
```

- Stream – ukazovateľ na súbor
- Offset – relatívna pozícia oproti whence, kam sa máme posunúť
- Whence – k čomu je offset relatívny
  - SEEK_SET – začiatok
  - SEEK_CUR – aktuálna pozícia
  - SEEK_END – koniec

Nastavenie sa na pozíciu

- **V súbore vieme nastaviť pozíciu ”kurzora” - odkiaľ čítame, kam chceme zapisovať**

- Na začiatok súboru sa dá vrátiť aj jednoduchšie

```
void rewind(FILE *stream)
```

Nastavenie sa na pozíciu

- **V súbore vieme nastaviť pozíciu ”kurzora” - odkiaľ čítame, kam chceme zapisovať**

```
fseek(fp, 100, SEEK_SET);

fseek(fp, -30, SEEK_CUR);

fseek(fp, -10, SEEK_END);
```

```
fseek(fp, 0, SEEK_SET);

rewind(fp);
```

Nastavenie sa na pozíciu

- **Ako zistiť, kde sa práve nachádzame?**

```
long ftell(FILE *stream);
```

- Zistenie pozície ukazovateľa čítania, zápisu v otvorenom súbore
- Relatívne k začitaku súboru, t.j. kde nastane nasledujúca operácia

- Návratová hodnota: aktuálna pozícia alebo -1 v prípade neúspechu

Funkcie a práca s pamäťou

16

Globálne a lokálne premenné

- **Stanovuje, kde bude premenná dostupná**

- Globálne premenné
  - Platnosť od miesta definície po koniec súboru
  - Sú automaticky inicializované na 0, ale netreba sa na to spoliehať
  - Ideálne nepoužívať, kým netreba

- Lokálne premenné
  - Definované vo funkciách
  - Platnosť od definície po koniec funkcie
  - Nie sú automaticky inicializované

Alokácia pamäte

- **Každá premenná má počas svojej existencie pamäťový priestor**

- Vyhradenie miesta v pamäti sa nazýva **alokácia**
  - Statická alokácia
  - Dynamická alokácia

Alokácia pamäte

- **Statická alokácia pamäte**

- Keď vieme prekladaču vopred povedať, aké máme na premenné pamäťové nároky
  - napr. vieme, že budeme potrebovať dve prenenné typu double a jednu premennú typu char

- Prekladač sám určí požiadavky pre všetky definované premenné
- Pri spustení programu sa pre ne alokuje miesto
- Behom programu sa nemanipuluje s touto pamäťou

- Premenné majú alokované miesto od začiatku programu do jeho konca (v zásobníku)
- Ruší ich operačný systém

Alokácia pamäte

- **Dynamická alokácia pamäte**

- Pamäťový priestor sa vymedzuje v hromade (heap)
- Za behu program je možné dynamicky prideliť oblast pamäte určitej veľkosti
- Prístup pomocou ukazovateľov (neskôr)

Alokácia pamäte

- **Vymedzenie pamäte v zásobníku**

- Má na starosti kompilátor pri volaní funkcie
- Rieši sa takto väčšina lokálnych premenných definovaných vo funkciách
- Existencia premenných je viazaná na funkciu
- Medzi jednotlivými volaniami funkcie zaniká

Funkcie

- **Jazyk C je založený na funkciách**

- Vždy funkcia main()
  - Program v nej začína aj končí
- Funkcie šetria čas - vyhýbame sa písaniu toho istého kódu viackrát
- Zlepšuje sa čitateľnosť program
- Nedajú sa vnárať

- Vracajú hodnotu
  - Alebo aj nie - void

Funkcie

- **Deklarovanie, definovanie a použitie**

```
int max(int a, int b);
```

```
int max(int a, int b)
{
   return (a > b ? a : b);
```

```
x = max(10 * i, j - 15);
```

Funkcie

```
#include <stdio.h>

int max(int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}

int main() {
   int x, y;

   printf("Zadajte 2 cisla: ");
   scanf("%d %d", &x, &y);
   printf("Maximum: %d\n", max(x, y));

   return 0;
}
```

Funkcie

- **Deklarácia vs. Definícia**

- Ak funkciu najprv deklarujeme môžeme ju volať aj pred definovaním.
  - Bez toho funkcia pozná iba tie, ktoré sú pred ňou.

- Zvyk: Deklarovať všetky funkcie na začiatku programu

```
#include <stdio.h>

int max(int a, int b);

int main() {
   int x, y;

   printf("Zadajte 2 cisla: ");
   scanf("%d %d", &x, &y);
   printf("Maximum: %d\n", max(x, y));

   return 0;
}

int max(int a, int b) {
   return (a > b ? a : b);
}
```

Funkcie

- **Návratová hodnota**

- Každá funkcia má definovanú návratovú hodnotu
- Môže to byť čokoľvek, okrem polí a funkcií

- Nemusí to byť nič – void() - procedúra

```
void vypis_int(int i)
{
   printf("%d", i);
}
```

```
vypis_int(a + b);
```

Funkcie

- **Funkcia bez parametrov**
- Typ void – význam “nič”
- Prázdny zoznam parametrov nie je to isté, ako void

```
int scitaj(void) {
   int a, b;

   scanf("%d %d", &a, &b);
   return (a + b);
}
```

```
j = scitaj();
```

Funkcie

- **Odovzdávanie parametrov**

- Hodnotou – vytvára sa lokálna kópia premennej v zásobníku
  - Na konci funkcie sa lokálna kópia stráca

- Odkazom – v C sa rieši pomocou ukazovateľov (neskôr)
  - Vtedy sa z vnútra funkcie manipuluju prostredníctvom pamäťového miesta s pôvodnou hodnotou

Rekurzia

- **Definícia rekurzie**

> - rekurzia: viď rekurzia

- Funkcia, ktorá volá samú seba (zmyslom sú väčšinou rôzne parametre)
- Musí obsahovať rekurzívnu aj nerekurzívnu vetvu

Rekurzia

- **Faktoriál**

```
1! = 1
2! = 2.1! = 2.1 = 2
3! = 3.2! = 3.2.1 = 6
4! = 4.3! = 4.3.2.1 = 24
5! = 5.4! = 5.4.3.2.1 = 120
```

Rekurzia

- **Faktoriál**

```
1! = 1
2! = 2.1! = 2.1 = 2
3! = 3.2! = 3.2.1 = 6
4! = 4.3! = 4.3.2.1 = 24
5! = 5.4! = 5.4.3.2.1 = 120
```

```
long fakt(int n)
{
   return ((n <= 0) ? 1 : n * fakt(n - 1));
}
```

Rekurzia

- **Faktoriál**

`main()`

`fakt(3)`:

`(3 <= 0)` neplatí

→ `return(3 * fakt(2))`

`fakt(2)`:

`(2 <= 0)` neplatí

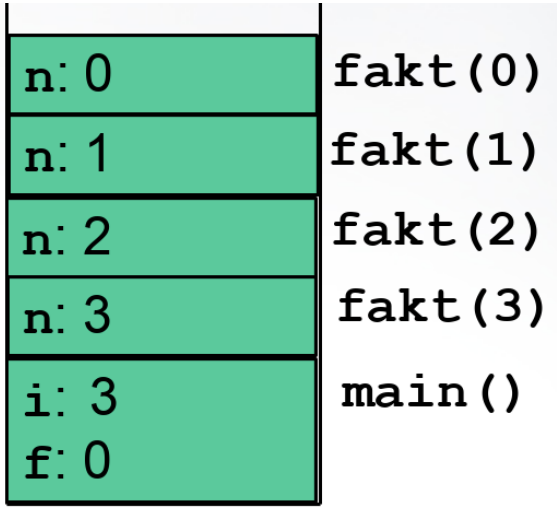
→ `return(2 * fakt(1))`

`fakt(1)`:

`(1 <= 0)` neplatí

→ `return(1 * fakt(0))`

`fakt(0)`:



Rekurzia

- **Faktoriál – iteratívne, bez použitia rekurzie**

```
long fakt(int n)
{
   return ((n <= 0) ? 1 : n * fakt(n - 1));
}
```

```
long fakt(long n)
{
   long i, f=1;

   for(i=1; i<=n; i++)
      f *= i;
   return f;
   }
}
```

Footnotes

- **Prednáška je dostupná na YouTube:**

https://www.youtube.com/watch?v=CKpWxJTCBaY

V prednáške boli použité materiály zo slidov prednášok ZPrPr1 od Gabriely Grmanovej.

Q?

35