Procedurálne programovanie

slido slido.com
#1104437
PrPr – P3

Ján Zelenka

Ústav Informatiky

Slovenská akadémia vied

Obsah prednášky

slido slido.com
#1104437
PrPr – P3

1. **Funkcie**
2. **I/O - Vstup a výstup**
   - Štandardný vstup a výstup
   - Vstup a výstup v jazyku C
   - Práca so súbormi (textovými, binárnymi)

Spätná väzba: *https://forms.gle/5ft8RFuLtVb4gRXm8*

Opakovanie

Cyklus `for` vs Cyklus `while` (do-while)

```
for(inicializacny_vyraz; podmienka_vykonania; inkrementalny_vyraz){
   //telo cyklu...prikazy;
}
```

```
inicializacny_vyraz;
while(podmienka_vykonania){
   //telo cyklu...prikazy;
   inkrementalny_vyraz;
}
```

```
inicializacny_vyraz;
do{
   //telo cyklu...prikazy;
   inkrementalny_vyraz;
}while(podmienka_vykonania);
```

Predčasné ukončenie cyklu

- *break* – ukončenie cyklu a pokračovanie za cyklom
- *continue* – ukončenie tela cyklu a pokračovanie ďalšou iteráciou
- *goto* – neodporúča sa používať (v štruktúrovanom programovacom jazyku sa mu dá vždy vyhnúť)

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

Zdroj:
https://www.programiz.com/c-programming/c-break-continue-statemer

Opakovanie

Úloha 2: Majme daný nasledovný program v jazyku C.

```
#include <stdio.h>
int main(void)
{
  int k;
  for (k = 'i'; k>'a'; k = k-1)
        printf("%c", k);
  return 0;
}
```

Napíšte, čo bude výstupom programu na obrazovku:

i, h, g, f, e, d, c, b,

Funkcie

slido slido.com
#1104437
PrPr – P3

Funkcie

- jazyk C je založený na funkciách
  - kratšie programy majú jednu funkciu **`main()`**
  - väčšina má viac funkcií
- spracovanie programu
  - začína volaním funkcie **`main()`**
  - končí opustením funkcie **`main()`**
- funkcie nemôžu byť vhniezdené
- nie procedúry - všetky funkcie vracajú hodnotu
  - dajú sa použiť aj ako procedúry (vrátia **`void`**)

Funkcie

- rozdelenie riešenia problému na podproblémy
- logicky samostatná časť programu (podprogram), ktorá spracuje svoj vstup a vráti výstup
  - pomenovaná skupina príkazov, ktorú vieme spustiť
  - výstup v návratovej hodnote
  - pomocou vedľajších efektov
    - zmena argumentov, globálnych premenných, súborov...

```
int sucet(int a, int b){
    return a + b;
}
int main (void){
...
    int c = sucet(a, b);
...
}
```

definícia fukncie

volanie funkcie

Funkcie

Argumenty: lokálne premenné, predávajú si hodnotou

Návratová hodnota: výstup funkcie (za kľúčovým slovom ***return***)

```
návratový_typ meno_funkcie(argumenty)
{
    telo_funkcie;
    return(hodnota_typu_zhodného_s_návratový_typ);
}
```

Kľúčové slovo ***void***

žiadne argumenty

```
int f(void);
```

bez návratovej hodnoty (procedúra)

```
void f(int a, int b);
```

Prototyp funkcie (deklarácia funkcie)

slido slido.com #1104437 PrPr – P3

Deklarácia funkcie poskytuje kompilátoru:

- meno funkcie
- počet a typ jej argumentov
- typ návratovej hodnoty

- nemôže byť vykonaná viackrát
- nepriraďuje sa pamäť

Neúplná deklarácia

```
int sucet(int, int);
```

Úplná deklarácia

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

Definícia funkcie:

- implementácia tela funkcie
- musí zodpovedať prototypu funkcie (ak existuje)
  - mená premenných môžu byť iné
- kučeravé zátvorky reprezentujú blok (kód funkcie)
  - premenné definované v bloku existujú iba v rámci neho
- každá funkcia má práve jednu definíciu
- pre funkciu sa vyhradí pamäť

```
int sucet(int a, int b){
   return (a+b);
}
```

Funkcie bez parametrov

definícia funkcie

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

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

volanie funkcie

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

Funkcie

Prenesenie hodnoty mimo funkcie:

- návratová hodnota
- pomocou smerníkov (neskôr)
- globálna premenná (nepoužívať)

Vracanie viacerých hodnôt:

- globálne premenné
- štruktúrovaný typ (struct, smerník)
- modifikácia vstupných parametrov (smerník)

Ukončenie funkcie (aj cyklu)

- **`return`** – ukončenie celej funkcie
  - snažte sa mať jeden return na konci funkcie
- **`exit`** – ukončenie celého programu
  - funkcia **exit** (#include<stdlib.h>)
    - parametrom ja návratová funkcia celého programu
    - EXIT_FAILURE alebo EXIT_SUCCESS
  - funkcia **assert** (#include<assert.h>)
    - ak podmienka nie je splnená, program sa zastaví s chybou
    - #define NDEBUG – vypnutie assert

Umiestnenie definícií funkcií

slido slido.com
#1104437
PrPr – P3

- funkciu umiestňujeme pred funkciu main
- ak je funkcia umiestnená po funkcii main pred funkciou main musí byť uvedená aspoň deklarácia funkcie (hlavička funkcie)
- ak funkcia volá inú funkciu musí byť volaná funkcia pred volajúcou funkciou, alebo pred volajúcou funkciou musí byť aspoň deklarácia volanej funkcie

Použitie funkčného prototypu

```
#include <stdio.h>

float B(float r);

int A(int x) {
    int y;

    y = B(x);
    return y;
}

float B(float r) {
    return (r * 2* 3.14);
}
```

funkčný prototyp na globálnej úrovni

Umiestnenie definícií funkcií

slido slido.com
#1104437
PrPr – P3

- funkcia **nemôže** byť definovaná vo vnútri inej funkcie neexistujú vhniezdené funkcie (t.j. do tela jednej funkcie nemôžem umiestniť celú druhú funkciu (hlavička + telo funkcie), ale iba jej volanie)

Funkcia `main()`

- program pozostáva z funkcií
  - **musí byť aspoň** jedna funkcia: `main` -> vstupný bod každého programu v jazyku C

Najjednoduchšia forma:
- žiadne vstupy
- vráti 0 ak nenastala chyba
  inak vráti nenulovú hodnotu

```
int main(void);
```

Prístup k údajom zadaných z konzoly:
- **`argc`** - počet argumentov
- **`argv`** - hodnoty argumentov

```
int main(int argc, char **argv);
```

- viac funkcií:
  - ak je potrebné opakovať nejaký výpočet, vytvorí sa funkcia obsahujúca kód pre tento výpočet - funkcia sa potom volá z inej funkcie (napr. `main`)
  - ak je program príliš dlhý - kvôli prehľadnosti ho rozdelíme do menších častí

Hlavný program

- funkcia **`main`**
  - vždy musí byť uvedená v programe
  - v programe sa nemôže volať, jej volanie je automatické (začína ním vykonávanie programu)
  - má predpísané argumenty, ktoré sa dajú vynechať (vysvetlíme si neskôr)

`int` - znamená, že vracia celočíselnú hodnotu (nemusí)

telo funkcie uzatvorené v **`{`** **`}`**

```
int main()
{
    int i, j;

    i = 5;
    j = -1;
    j = j + 2 * i;

    return 0;
}
```

funkcia **`main()`** má vynechané argumenty

funkcia **`main`** vracia hodnotu 0

Bloky

Uzatvárajú:

- zložený príkaz: zoznam príkazov
- blok: definície a zoznam príkazov
- premenné deklarované v bloku, automaticky zanikajú na jeho konci

```
{
    i = 5;
    j = 6;
}
```

```
{
    int i;

    i = 5;
    j = 6;
}
```

```
int main()
{
    int i = 5,
        j = 6;

    j = j + 2 * i;
    return 0;
}
```

inicializácia priamo v definícii

zložený príkaz

blok (obsahuje definíciu)

Rozsah platnosti premenných

- časť kódu, v ktorom je premenná použiteľná (**scope**)
- v mnohých prípadoch korešponduje s blokom, kde je definovaná

- Lokálna premenná
  - obmedzený rozsah platnosti (funkcia, blok…)
- Globálna premenná
  - deklarovaná mimo funkcie
  - nezaniká medzi volaniami funkcií

Funkcie s viacerými argumentami

```
void fn( int a, int b)
{
    printf("%d %d \n",a,b); //2 3
}
int main(void)
{
   int i=0;
   fn(i++, i+=2);
   return 0;
}
```

V prípade funkcii s viac argumentami, poradie vyhodnocovania je v štandarde jazyka ANSI C nešpecifikovane: argumenty môžu byť vyhodnotené **v ľubovoľnom poradí**. Poradie vyhodnocovania závisí od kompilátora a môže závisieť aj od spôsobu optimalizácie programu pri kompilácii.

Poučenie: *Argumenty, ktoré dávame do funkcii by mali obsahovať len výrazy, ktoré hodnoty čítajú. Akékoľvek úpravy hodnôt premenných treba vykonať predtým ako ich použijeme v argumentoch pri volaní funkcii.*

Vstup a výstup

slido slido.com
#1104437
PrPr – P3

Štandardný vstup a výstup

- Koncept štandardného vstupu a výstupu
  - program nevie kto mu dáva vstupné dáta
  - program nevie kam zapisuje výstupné dáta
  - zvyčajne je vstup == `klávesnica`
  - zvyčajne je výstup == `obrazovka`
- To sa dá ľahko zmeniť
  - štandardný vstup zo súboru
    - Windows: `program.exe < subor.txt`
    - Unix: `./program < subor.txt`
  - štandardný výstup do súboru
    - Windows: `program.exe > vystup.txt`
    - Unix: `./program > vystup.txt`

Vstup a výstup v jazyku C

- Základné možnosti vstupu a výstupu
  - výstup na obrazovku (`puts, printf`)
  - vstup z klávesnice (`gets, scanf`)
- Funkcie pre vstup a výstup sú poskytované cez knižnicu `stdio.h`
  - nie sú súčasťou jazyka
  - štandardná knižnica je (takmer) vždy dostupná
    - okresané platformy, jadro OS…
    - kompilovanie bez štandardných knižníc `gcc -nostdlib`

Vyrovnávacia pamäť pre vstup a výstup

Dáta medzi producentom a konzumentom sa nemusia preniesť ihneď

- text na obrazovku je písaný po riadkoch
- dáta na disk sú zapísané po blokoch
- konzument sa nevolá pri každom elementárnom znaku

Produkované dáta sú ukladané do vyrovnávacej pamäti (buffering)

- Prečítanie prebehne pri jej zaplnení
  - nastavenie aplikácie, OS…
  - vynútené prečítanie (bez ohľadu na zaplnenie buffera) -> `fflush(stdout);`

Práca so súbormi

slido slido.com
#1104437
PrPr – P3

Súbor

- postupnosť bytov uložených na médiu (disku) v niekoľkých blokoch (nie nutne za sebou)
- prístup k blokom - operačný systém
- vstup zo súboru
  - naraz sa prečíta celý blok z disku do pamäte (buffer) - položky sa potom čítajú z pamäte (rýchlejšie)
- výstup
  - dáta sa zapisujú do bufferu a keď je plný, zapíše sa na disk
  - napr. v UNIXE sa dá používať aj nebufrované vstupné a výstupné operácie (`io.h`)
- koniec súboru
  - často špeciálny znak (napr. "`Ctrl Z`")

Typy súborov

Binárne súbory:

- čítanie/zápis postupnosti bytov (zodpovedajúcej reprezentácii vstupných/výstupných dát v pamäti)
- Problém s prenositeľnou

Textové súbory:

- Binárny súbor interpretovaný ako text
- Čísla sú uložené ako postupnosť znakov (pamäť)
- Strata pri práci s reálnymi číslami
- Potrebná textová reprezentácia pri užívateľských dátových štruktúrach (obslužné funkcie)

Práca so súborom v jazyku C

- základný dátový typ: **`FILE *`**
  - ukazovateľ (pointer - **`*`**) na objekt typu FILE
    - ukazovateľ obsahuje adresu objektu typu FILE
    - ako adresár - zapísaná adresa, kde začína súbor na disku
  - dodržať veľké písmená (**`FILE *`**, nie **`file *`**)
- definícia premennej **`f`** pre prácu so súborom:

```
FILE *f;
```

  - aj pre čítanie, aj pre zápis rovnaké
  - pre viac premenných:

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

pre čitateľnosť je vhodné používať **`fr`** pre čítanie, **`fw`** pre zápis

Otváranie súborov

- súbory sa otvárajú stále rovnakou funkciou **`fopen()`**
  - či ide o textový alebo binárny súbor
  - či ide o zápis alebo čítanie

Prototyp funkcie:

- **`const`**: len vstupný argument, nebude sa meniť vo funkcii
- **`char *`** - reťazec znakov

```
FILE *fopen(const char *path, const char *rezim)
```

vráti ukazovateľ (adresu) na otvorený súbor alebo **`NULL`**

cesta +meno súboru

aký typ súboru a na akú činnosť sa bude otvárať

Otváranie súboru

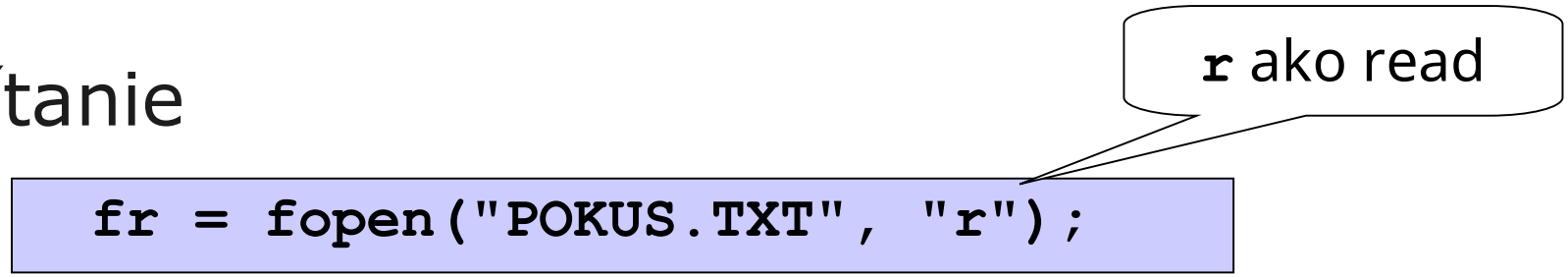
`path` obsahuje cestu k súboru

- Relatívnu: `subor.txt, ../subor.txt`
- Absolútnu: `C:\subor.txt`

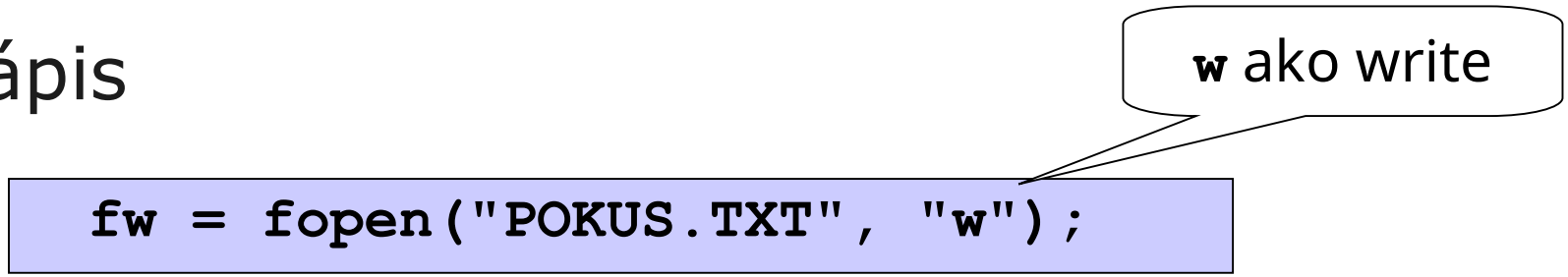
- Pozor na znak '\' v reťazci obsahujúcom cestu
  - V jazyku C je \ špeciálny znak, je potrebné použiť escape sekvenciu \\
  - `"C:\subor.txt" vs "C:\\subor.txt"`

Čítanie a zápis z/do súboru

čítanie



zápis



- aj ďalšie režimy otvorenia súboru (nielen "**r**" a "**w**")
- režimy "**r**" a "**w**" - otvorenie textového súboru
- režimy "**rb**" a "**wb**" - otvorenie binárneho súboru

Významy parametra `rezim`

slido slido.com
#1104437
PrPr – P3

r	textový súbor pre čítanie
w	nový textový súbor otvorený pre zápis
a	zapisovanie na koniec existujúceho súboru
r+	existujúci textový súbor pre čítanie a zápis
w+	nový textový súbor pre čítanie a zápis
a+	textový súbor pre čítanie a zápis na koniec

Pri práci v binárnom režime treba pridať b (napr. "wb")

Významy parametra `rezim`

- niektoré implementácie umožňujú explicitne určiť, že ide o textový režim: **`"rt"`** **`"wt"`** **`"at"`**
- ak otvoríme existujúci súbor v režime **`"w"`**, tak sa tento súbor najprv vymaže a potom sa vytvorí nový
- ak otvoríme existujúci súbor v režime **`"a"`** tak sa tento súbor otvorí a ukazovateľ sa presunie na koniec súboru (rozširovanie existujúceho súboru)
- ak použijeme režim rozšírený o znak **+**, je možné do súboru aj zapisovať

Testovanie správnosti otvorenia súboru

Otvorenie súboru nemusí byť úspešné

`fopen()`

- ak sa podarí otvoriť súbor - vracia ukazovateľ na súbor,
- inak - vracia konštantu **`NULL`** (definovaná v **`stdio.h`**, má hodnotu 0)

Testovanie:

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

Otváranie súboru - Zhrnutie

- Štandardne sa súbor otvára ako textový
  - Na Unixe je textový aj binárny mód rovnaký
  - Na Windows pozor na konce riadkov (súbor vytvorený na Unixe zobrazte vo Windowse)
- Pozor na zmazanie existujúceho súboru
  - `fopen("subor.txt", "w")` → subor.txt veľkosť 0
- Pozor na situáciu, keď súbor neexistuje
  - `fopen("subor.txt", "r")== NULL`
- Ak je súbor otvorený na čítanie aj zápis ("rw"), medzi operáciou čítania a zápisu by sa mala vyprázdniť vyrovnávacia pamäť (`fflush()`)
  - Nie je odporúčané miešať čítanie a zápis

Aktuálna pozícia v súbore

- Po otvorení súboru sa interne uchováva aktuálna pozícia v súbore
  - Začiatok súboru (módy read "r" a write "w")
  - Koniec súboru (mód append "a")
- Čítanie a zápas prebieha na aktuálnej pozícii
- Pri čítaní a zápise dochádza automaticky k posunu o prečítané alebo zapísané znaky

Zistenie aktuálnej pozície v súbore

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

Funguje **iba na bežných súboroch**, nie na prúdoch (stdin...)

zistenie pozície ukazovateľa čítania, zápisu v otvorenom súbore relatívne k začiatku súboru

Použitie: zapamätať si pozíciu, na ktorú sa neskôr plánujete vrátiť (zapamätať si návratovú hodnotu a potom ju použiť vo `fseek()` relatívne k začiatku súboru)

Návratová hodnota: aktuálna pozícia alebo -1 v prípade neúspechu

Testovanie konca riadku

- Čítanie súboru po riadkoch
- postarať sa o testovanie konca riadku (EOLN - označenie, nie symbolická konštanta)
  - **testovanie** štandardného znaku pre koniec riadku v C: `\n`
  - `\n` - aj pre čítanie, aj pre zápis
  - `\n` - význam určuje prekladač podľa systému (`<CR>`, `<LF>`, alebo `<CR><LF>`)

Základné práce s otvoreným súborom

- porovnanie s čítaním z klávesnice a zápisom na obrazovku

```
c = getc(f);        fgets(s, n, f);
c = getchar();
```

čítanie znaku/celého riadku zo súboru

```
putc(c, f);         fputs(s, f);
putchar(c);
```

zápis znaku/reťazca do súboru

```
fscanf(f, "format", argumenty);
scanf("format", argumenty);
```

formátované čítanie zo súboru

```
fprintf(f, "format", argumenty);
printf("format", argumenty);
```

formátovaný zápis do súboru

Testovanie konca súboru

Konštanta `EOF` (End Of File)

```
while ((c = getc(file)) != EOF){
    ...
}
```

Ukončenie práce so súborom

- keď už nebudeme zo súboru čítať ani doňho zapisovať - uzatvoriť súbor, premenná **`f`** je typu **`FILE *`**:

```
fclose(f);
```

- nespoliehať sa, že po skončení programu sa v mnohých systémoch automaticky uzavrie súbor
  - počet súčasne otvorených súborov je obmedzený
  - zápis bufferu do súboru (preto uzatvárať ihneď) - pri spadnutí programu by zostali dáta v bufferi a stratili by sa

Testovanie správnosti zatvorenia súboru

`fclose()`

- ak sa nepodarí zatvoriť súbor - vracia konštantu **`EOF`**

Testovanie:

```
if(fclose(fr) == EOF)
   printf("Subor sa nepodarilo zatvorit.\n");
```

Zmena aktuálnej pozície v súbore

Nastavenie ukazovateľa na pozíciu čítania alebo zápisu v otvorenom súbore.

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

Argumenty:

- `stream` – ukazovateľ na súbor
- `offset` – relatívna pozícia oproti origin, na ktorú sa má ukazovateľ posunúť (v Bytoch)
- `origin` – k čomu je offset relatívny
  - SEEK_SET: `offset` – relatívne k začiatku súboru
  - SEEK_CUR: `offset` – relatívne k aktuálnej pozícii
  - SEEK_END: `offset`– relatívne ku koncu súboru (treba používať negatívne hodnoty)

Návratová hodnota: 0 pri úspechu, -1 pri neúspechu

Nastavenie ukazovateľa na začiatok súboru

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

Funguje **iba na bežných súboroch**, nie na prúdoch (stdin...)

Štandardný vstup a výstup

C pracuje s klávesnicou a obrazovkou ako so súborom v **`stdio.h`** - definované dva konštantné ukazovatele:

```
FILE *stdin;
FILE *stdout;
```

nie **`stdio`**

- označujú štandardný vstupný/výstupný prúd (*standard intput-output stream*)
- je možné ich zmeniť pomocou presmerovania pri spúšťaní programu, napr.:

```
program < vstup.txt > vystup.txt
```

Štandardný vstup a výstup

- **`stdin`** a **`stdout`** môžu byť použité v programe ako argumenty operácií so súbormi:

`getc(stdin)`	je ekvivalentné	`getchar()`
`putc(c, stdout)`	je ekvivalentné	`putchar(c)`

- v **`stdio.h`** je definovaný ešte tretí prúd **`stderr`**, ktorý sa používa pri vypisovaní chybových správ

Vrátenie prečítaného znaku späť do bufferu

Často zistíme, že máme prestať čítať znak až potom, čo prečítame o znak naviac → vrátiť do bufferu

`ungetc(c, fr)`

vráti znak do vstupného bufferu
- ak je vrátenie úspešné, **`ungetc()`** vracia vrátený znak
- ak je vrátenie neúspešné, vráti **`EOF`**

Späť do bufferu môžeme zapísať aj iný ako práve prečítaný znak

Odstránenie, premenovanie a dočasný súbor

```
int remove (const char * filename)
```

- Odstránenie súboru s daným meno (cestou)

- `int rename (const char * oldname, const char * newname)`

- Premenovanie súboru (ak už existuje súbor s novým menom, tak bude zmazaný

- `FILE* tmpfile (void)`

- Otvorí sa dočasný jedinečný súbor
- Automaticky zaniká po skončení programu

Prehľad základných funkcií na prácu so súbormi

slido slido.com
#1104437
PrPr – P3

- `fopen()` - **otvorenie súboru** v rôznych režimoch (čítanie, zápis, textový binárny...)
- `fclose()` - **zatvorenie súboru** otvoreného fopen()
- `fseek()` - **nastavenie** ukazovateľa **na pozíciu** čítania, zápisu v otvorenom súbore
- `ftell()` - **zistenie pozície** ukazovateľa čítania, zápisu v otvorenom súbore

Prehľad základných funkcií na prácu so súbormi

slido slido.com
#1104437
PrPr – P3

Textový súbor:

- `getc()` - čítanie **znaku** z otvoreného súboru
- `putc()` - zápis **znaku** do otvoreného súboru
- `fgets()` - čítanie **celého riadku** z otvoreného súboru
- `fputs()` - zápis **reťazca** do otvoreného súboru
- `fprintf()` - analógia printf() a sprintf(), ale s výstupom do súboru -> pracujeme s **číslami, slovami...**

Binárny súbor:

- `fread()` - čítanie **bloku** z otvoreného súboru
- `fwrite()` - zápis **bloku** do otvoreného súboru

Ako sa zlepšovať

- **CodeForces**: dvojhodinové súťaže, ktoré riešia ľudia z celého sveta. Úlohy sú po anglicky a po rusky. http://codeforces.com/
- **TopCoder**: ako CodeForces. Úlohy sú po anglicky. https://www.topcoder.com/
- **Sphere Online Judge**: rôzne programátorské úlohy. Úlohy sú po anglicky https://www.spoj.com/
- **HackerRank**: https://www.hackerrank.com/

Zdroj: https://www.ksp.sk/trening/

- Naladťe sa na https://www.se-radio.net/
- Práca na open source projekte
- Odstránenie chyby + PULL request
- Implementácia funkčnosti z Issues (good first issue)

Typová konverzia

slido slido.com
#1104437
PrPr – P3

Typový systém

- Každá hodnota je na najnižšej úrovni reprezentovaná ako postupnosť bitov
- Každá hodnota má počas výpočtu priradený typ
- Typ hodnoty dáva postupnosti bitov význam – ako sa má intrepretovať

Sú pravidlá:

- pre zmenu typu
- definujúce ktorý typ môže byť použitý danou operáciou

C je statický typový systém

- typ je kontrolovaný počas prekladu

Typová konverzia

- prevod premennej určitého typu na iný typ
  - napr. `int` na `float`

- dva druhy konverzií:
  - implicitná - samovoľná, automatická
  - explicitná - vynútená, požadovaná

Implicitná typová konverzia

Pred vykonaním operácie sa samostatné operandy konvertujú:

- kedykoľvek sa objaví typ **`char`** alebo **`short int`**, konvertuje sa na **`int`**
- všetky operandy **`unsigned char`** a **`unsigned short`** sa konvertujú na **`int`** - ak **`int`** nepretečie, inak na **`unsigned int`**

Implicitná typová konverzia

Ak majú dva operandy jednej operácie rôzny typ, operand s nižšou prioritou je konvertovaný na typ s vyššou prioritou podľa hierarchie:

```
int            ⇒ unsigned int
unsigned int   ⇒ long
long           ⇒ unsigned long
unsigned long  ⇒ float
float          ⇒ double
double         ⇒ long double
```

- `int` má najnižšiu prioritu
- jeden operand typu `float` a druhý nižšiu prioritu, druhý ⇒ `float`

Implicitná typová konverzia

V priraďovacích príkazoch je typ na pravej strane konvertovaný na typ z ľavej strany.

```
double x;

x = 5;
```

x bude mať hodnotu 5.0

```
int i;

i = 5.0;
```

i bude mať hodnotu 5

Explicitná typová konverzia

Pretypovanie

- jazyk C dovoľuje takmer ľubovoľnú konverziu → riziko, že to bude nevhodné

Syntax: `(typ) výraz`

Príklady:

`(int) char_vyraz`	- prevod znaku na ordinálne číslo
`(char) int_vyraz`	- prevod ordinálneho čísla na znak
`(int) double_vyraz`	- odrezanie desatinnej časti
`(double) int_vyraz`	- prevod celého čísla na reálne
`(double) float_vyraz`	- zväčšenie presnosti

Ďakujem vám za pozornosť!

Spätná väzba: *https://forms.gle/5ft8RFuLtVb4gRXm8*