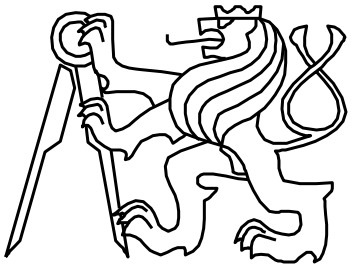
Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.
Praha & EU: Investujeme do vaší budoucnosti

Struktury, qsort, mergesort



BI-PA1 Programování a algoritmizace 1, ZS 2012-2013
Katedra teoretické informatiky
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

Obsah

- datový typ struktura
- vnitřní reprezentace struktury
- pole struktur
- binární soubor struktur
- příklady vytváření, čtení a modifikací binárních souborů struktur
- standardní funkce pro řazení

Úvodní příklad

- Program, který z textového souboru přečte jména a příjmení studentů a jejich studijní průměr a vypíše studenta s nejlepším průměrem
- Příklad vstupního souboru:

Josef Novak 2.7
Katerina Mala 1.3
Sarka Pospisilova 1.9
Karel Nemec 2.8

- Pro tento vstupní soubor program vypíše:

Katerina Mala 1.3

Úvodní příklad

- Hrubé řešení:

```
„inicializace údajů o nejlepším studentovi“
„otevři vstupni soubor“
while („přečteny údaje o dalším studentovi“) {
  „vypiš údaje o dalším studentovi“
  if („průměr dalšího studenta je lepší než
          průměr doposud nejlepšího“)
    „nejlepší student = další student“
}
„vypiš nejlepšího studenta“
```

Úvodní příklad

- Podrobné řešení (prog11-studenti1.c):
  - V hlavní funkci *main* deklarujeme, kromě proměnné pro reprezentaci vstupního souboru, šest proměnných pro uložení údajů o dalším a nejlepším studentovi:

```
#define MAXDELKA 20
int main(void) {
        char dalsiJmeno[MAXDELKA];
        char dalsiPrijmeni[MAXDELKA];
        float dalsiPrumer;
        char nejlepsiJmeno[MAXDELKA];
        char nejlepsiPrijmeni[MAXDELKA];
        float nejlepsiPrumer = 5;
        FILE *vstup = otevriSoubor("studenti.txt", "r");
        ...
}
```

Úvodní příklad

- Podrobné řešení (prog11-studenti1.c):
  - Pak čteme vstupní soubor a aktualizujeme údaje o nejlepším studentovi

```
int main(void)
{  …
   FILE *vstup = otevriSoubor("studenti.txt", "r");
   while ( fscanf(vstup, "%s%s%f",
                          dalsiJmeno,  dalsiPrijmeni, &dalsiPrumer)!=EOF )
   {  printf("%20s %20s %.1f\n",
                 dalsiJmeno, dalsiPrijmeni, dalsiPrumer);
      if (dalsiPrumer<nejlepsiPrumer)
      {  strncpy(nejlepsiJmeno,dalsiJmeno,MAXDELKA-1);
         strncpy(nejlepsiPrijmeni,dalsiPrijmeni,MAXDELKA-1);
         nejlepsiPrumer = dalsiPrumer;
      }
   }
   fclose(vstup);
   printf("nejlepsi student:\n");
   printf("%20s %20s %.1f\n",
            nejlepsiJmeno, nejlepsiPrijmeni, nejlepsiPrumer);
            system("PAUSE"); return 0;
}
```

Řešení pomocí struktury

- Místo šesti proměnných zavedeme dvě proměnné typu *struktura*, které budou obsahovat tři položky – *jméno, příjmení* a *průměr*

```
/* prog11-studenti2.c */
#define MAXDELKA 20
struct student {
    char jmeno[MAXDELKA];
    char prijmeni[MAXDELKA];
    float prumer;
 };
int main(void) {
    struct student dalsi, nejlepsi;
    FILE *vstup = otevriSoubor("studenti.txt", "r");
    nejlepsi.prumer = 5;
    …
}
```

značka (jméno) struktury není jménem typu

jména položek struktury student

Řešení pomocí struktury

- Pak čteme vstupní data, ukládáme je do položek proměnné *dalsi* a položku *průměr* porovnáváme s položkou *průměr* struktury *nejlepsi*

```
int main(void)
{  struct student dalsi, nejlepsi;
   FILE *vstup = otevriSoubor("studenti.txt", "r");
   nejlepsi.prumer = 5;
   while (fscanf(vstup, "%s%s%f",
                          dalsi.jmeno, dalsi.prijmeni, &dalsi.prumer)!=EOF)
   {  printf("%20s %20s %.1f\n",  dalsi.jmeno, dalsi.prijmeni, dalsi.prumer);
      if (dalsi.prumer<nejlepsi.prumer) nejlepsi = dalsi;
   }
   fclose(vstup);
   printf("nejlepsi student:\n %20s %20s %.1f\n",
           nejlepsi.jmeno, nejlepsi.prijmeni, nejlepsi.prumer);
   system("PAUSE"); return 0;
}
```

pro struktury je definováno přiřazení

Drobná úprava

- Pro strukturu *struct student* můžeme zavést jméno typu deklarací *typedef* /* prog11-studenti3.c */

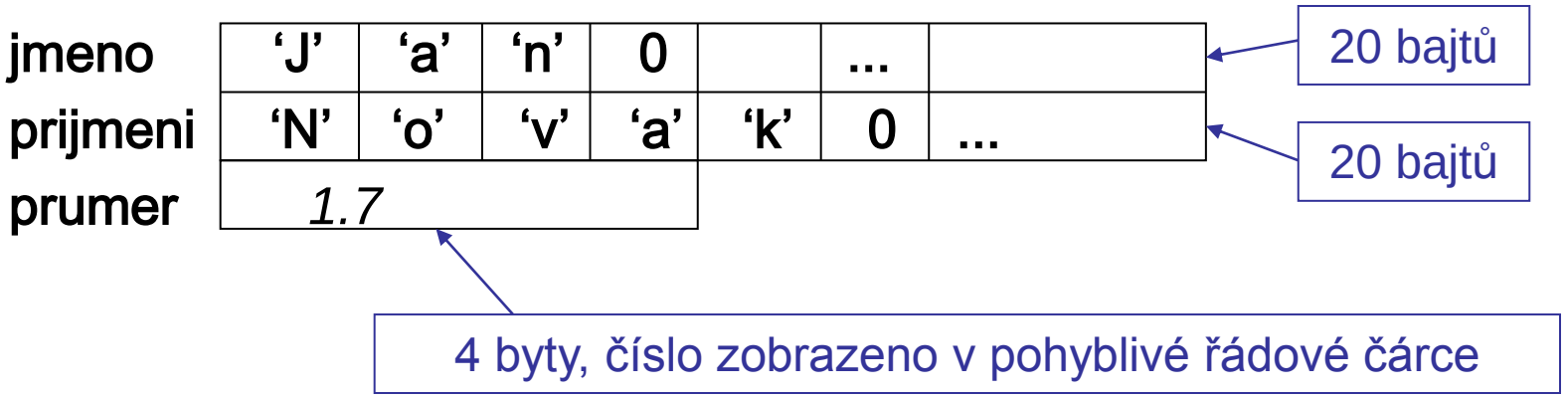
```
#define MAXDELKA 20
typedef struct student {
    char jmeno[MAXDELKA];
    char prijmeni[MAXDELKA];
    float prumer;
} Student;

int main(void) {
    Student dalsi, nejlepsi;
    nejlepsi.prumer = 5;
    …
}
```

identifikátor *Student* můžeme dále používat jako jméno typu

zde jsme to využili

Vnitřní reprezentace struktury

```
typedef struct student {
        char jmeno[MAXDELKA];
        char prijmeni[MAXDELKA];
        float prumer;
 } Student;
Student student = {"Jan", "Novak", 1.7};
```



Vnitřní reprezentace struktury

- Program prog11-student.c vypíše adresy položek proměnné *student* a počet bytů vnitřní reprezentace:

```
#define MAXDELKA  20
typedef struct student {
        char jmeno[MAXDELKA];
        char prijmeni[MAXDELKA];
        float prumer;
} Student;
Student student = {"Jan", "Novak", 1.7};
int main(void) {
        printf("adresa student.jmeno: %u\n", (unsigned)student.jmeno);
        printf("adresa student.prijmeni: %u\n", (unsigned)student.prijmeni);
        printf("adresa student.prumer: %u\n", (unsigned)&student.prumer);
        printf("celkovy pocet bajtu: %d\n", sizeof(Student));
        system("PAUSE");
        return 0;
}
```

například

```
4202496
4202516
4202536
44
```

Vnitřní reprezentace struktury

- Změníme velikost polí znaků na 15:

```
#define MAXDELKA 15
typedef struct student {
    char jmeno[MAXDELKA];
    char prijmeni[MAXDELKA];
    float prumer;
} Student;
```

a program spustíme. Tentokrát vypíše

4202496

4202511

4202528

36

- Proč: mezi položku *prijmeni* a *prumer* překladač vložil výplň 2 bajty, aby položka *prumer* typu *float* byla na adrese dělitelné 4, což zjednodušuje její použití jako operandu ve strojových instrukcích v cílovém programu

Pole struktur

Příklad: program, který přečte textový soubor obsahující jména, příjmení a studijní průměry studentů, uloží je do pole, pole seřadí vzestupně podle průměru a vypíše /* prog11-studenti4a.c */

```
#define MAXDELKA 20
#define MAXPOCET 20
typedef struct student {
        char jmeno[MAXDELKA];
        char prijmeni[MAXDELKA];
        float prumer;
} Student;
FILE *otevriSoubor(char *jmeno, char* mode) {…}
void vypisStudenta(Student student) {…}
void vymena(Student *p, Student *q) {…}
void seradPodlePrumeru(Student a[], int n) {…}
int main(void) {…}
```

Pole struktur – hlavní funkce:

```
/* prog11-studenti4a.c */
int main(void) {
    Student studenti[MAXPOCET];
    int pocet = 0, i;
    FILE *vstup = otevriSoubor("studenti.txt", "r");
    while (pocet<MAXPOCET &&
           fscanf(vstup,"%s%s%f",studenti[pocet].jmeno,
                                 studenti[pocet].prijmeni,
                                 &studenti[pocet].prumer)!=EOF)
        pocet++;
    fclose(vstup);
    seradPodlePrumeru(studenti, pocet);
    for (i=0; i<pocet; i++) vypisStudenta(studenti[i]);
    system("PAUSE");
    return 0;
}
```

Pole struktur seřazení pole studentů podle prospěchu

```
void vymena(Student *p, Student *q) {
        Student pom;
        pom = *p; *p = *q; *q = pom;
}


void seradPodlePrumeru(Student a[], int n) {
        int i, j, imin;
        for (i=0; i<n-1; i++) {
                imin = i;
                for (j=i+1; j<n; j++)
                        if (a[j].prumer<a[imin].prumer) imin = j;
                        if (imin!=i) vymena(&a[imin], &a[i]);
        }
}
```

Pole struktur

Pro čtení údajů o studentovi z textového souboru zavedeme proceduru s výstupním parametrem typu *Student** /* prog11-studenti4b.c */

```
int ctiStudenta(FILE *vstup, Student *ps) {
        int vysl;
        Student s;
        vysl = fscanf(vstup, "%s%s%f", s.jmeno, s.prijmeni, &s.prumer);
        *ps = s;
        return vysl;
}
int main(void) {
        Student studenti[MAXPOCET];
        int pocet = 0, i;
        FILE *vstup = otevriSoubor("studenti.txt", "r");
        while (pocet<MAXPOCET &&
                ctiStudenta(vstup, &studenti[pocet])!=EOF) pocet++;
        fclose(vstup);
…}
```

Soubor struktur - vytvoření

- Předchozí příklad doplníme o uložení pole studentů seřazeného podle průměru do binárního souboru (prog11-studenti5a.c):

```
int main(void) {
    Student studenti[MAXPOCET];
    int pocet = 0, i;
    char jmenoVystupu[];
    FILE *vystup ,*vstup… /* nacteni ze souboru, serazeni podle prumeru*/
    char jmenoVystupu[] = "studenti.bin";
    *vystup = otevriSoubor(jmenoVystupu, "wb");
    /* zapis pole studenti do souboru */
    fwrite(studenti, sizeof(Student), pocet, vystup);
    fclose(vystup);
    printf("pole studentu ulozeno do souboru %s\n", jmenoVystupu);
    system("PAUSE");
    return 0;
}
```

Soubor struktur - čtení

- Program, který přečte binární soubor vytvořený předchozím programem a vypíše údaje na obrazovku (prog11-5b.c)
- I když jsme soubor vytvořili zápisem pole, budeme ho číst záznam po záznamu

```
int main(void) {
    Student student;
    FILE *vstup = otevriSoubor("studenti.bin", "rb");
    while (fread(&student, sizeof(Student), 1, vstup)==1)
      vypisStudenta(student);
    fclose(vstup);
    system("PAUSE");
    return 0;
}
```

Zápis na konec souboru

- Program prog11-studenti5c.c zapíše na konec binárního souboru studentů záznam dalšího studenta, který přečte z klávesnice

```
int main(void) {
        Student novy;
        FILE *soubor = otevriSoubor("studenti.bin", "ab");
        printf(„Zadej jmeno, prijmeni a prumer noveho studenta\n");
        ctiStudenta(stdin, &novy);
        fwrite(&novy, sizeof(Student), 1, soubor);
        fclose(soubor);
        …
/* vypis doplneneho souboru */
}
```

mód *ab* vyjadřuje, že binární soubor má být otevřen pro zápis na konec

Poznámka: chceme-li zapisovat na konec textového souboru, otevřeme ho s módem *at*

Vložení záznamu dovnitř souboru

- Úkol: napsat program, který do seřazeného souboru studentů vloží záznam nového studenta tak, aby soubor zůstal seřazen
- Při sekvečním zpracování potřebujeme pomocný soubor
- Postup:
  1. přečteme nového studenta z klávesnice
  2. do pomocného souboru zkopírujeme všechny studenty, jejichž průměr není horší než průměr nového studenta
  3. do pomocného souboru zapíšeme nového studenta
  4. do pomocného souboru zkopírujeme zbytek souboru
  5. soubor přepíšeme zkopírováním pomocného souboru

Vložení záznamu dovnitř souboru

```
#define MAXDELKA 20                                   /* prog11-studenti6.c */
#define MAXPOCET 20
int main(void) {
        Student dalsi, novy;
        int p;
        FILE *soubor, *pomocny;  char jmenoSouboru[MAXDELKA];
        printf("zadej jmeno souboru: "); scanf("%s", jmenoSouboru);
        soubor = otevriSoubor(jmenoSouboru, "rb");
        pomocny = otevriSoubor("pomocny", "wb");
        /* cteni noveho studenta */
        printf("zadej jmeno prijmeni a prumer noveho studenta\n");
        ctiStudenta(stdin, &novy);
        /* zkopiruji se studenti, kteri nemaji horsi prumer nez novy */
        while ((p=fread(&dalsi, sizeof(Student), 1, soubor))==1 &&
                  dalsi.prumer<=novy.prumer)
               fwrite(&dalsi, sizeof(Student), 1, pomocny);
```

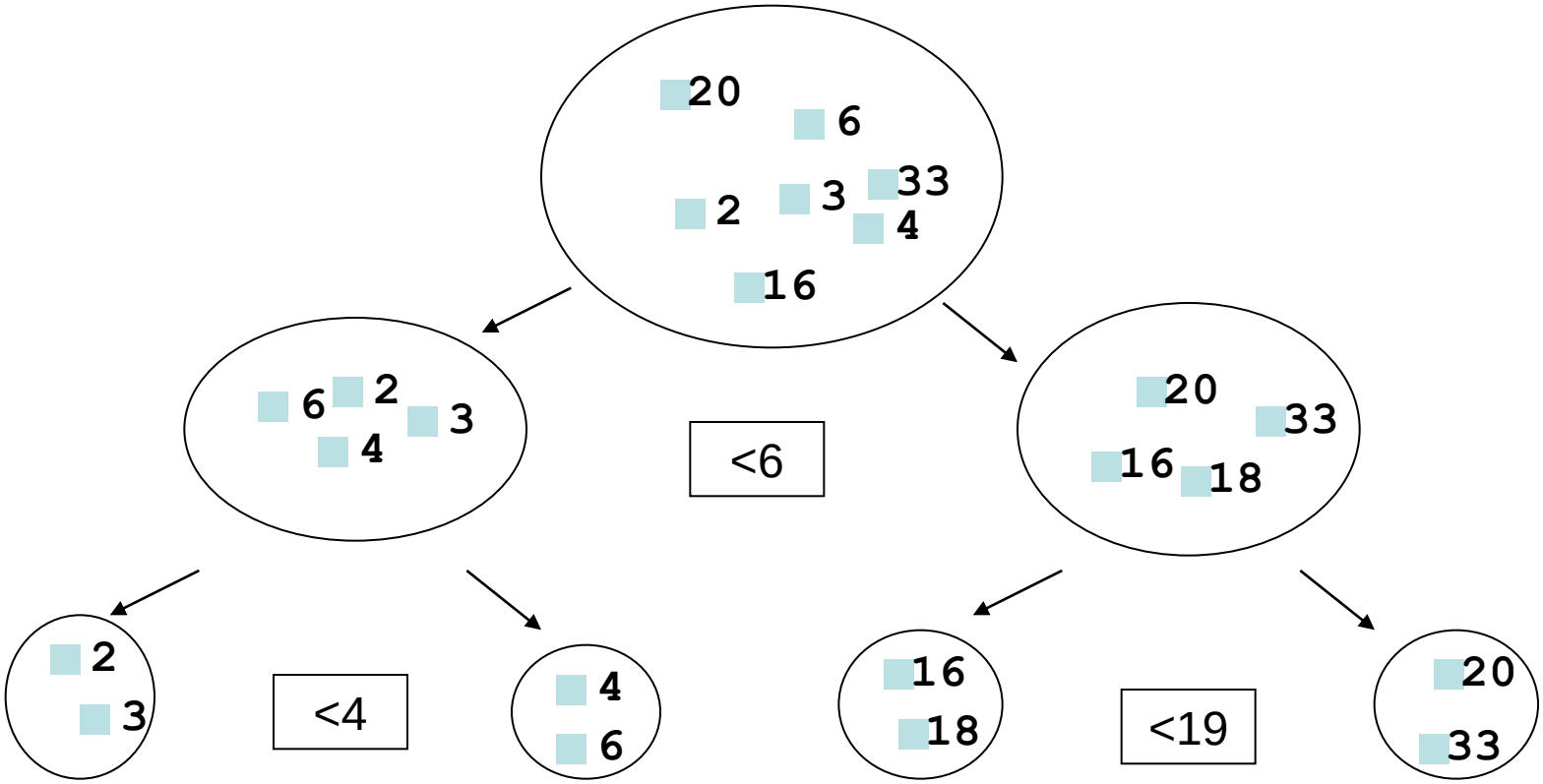
Vložení záznamu dovnitř souboru

```
/* zapise se novy student */
fwrite(&novy, sizeof(Student), 1, pomocny);
/* zkopiruje se zbytek studentu, ale jen tehdy, kdyz v predchozim
                                        cyklu byl dalsi student precten */
if (p) {
      fwrite(&dalsi, sizeof(Student), 1, pomocny);
      while (fread(&dalsi, sizeof(Student), 1, soubor)==1)
            fwrite(&dalsi, sizeof(Student), 1, pomocny);
       }
/* soubory se uzavrou */
fclose(pomocny);
fclose(soubor);
/* a otevrou pro prepsani souboru pomocnym */
soubor = otevriSoubor(jmenoSouboru, "wb");
pomocny = otevriSoubor("pomocny", "rb");
```

Vložení záznamu dovnitř souboru

```
    /* prepsani souboru pomocnym */
    while (fread(&dalsi, sizeof(Student), 1, pomocny)==1)
        fwrite(&dalsi, sizeof(Student), 1, soubor);
    fclose(pomocny);
    fclose(soubor);

    /* a na zaver vypis doplneneho souboru */
    printf("vypis doplneneho souboru\n");
    soubor = otevriSoubor(jmenoSouboru, "rb");
    while (fread(&dalsi, sizeof(Student), 1, soubor)==1)
        vypisStudenta(dalsi);
    fclose(soubor);
    system("PAUSE");
    return 0;
}
```

Quick Sort



Quick Sort

Existují různé heuristiky pro určení pivota, aby byl, co nejblíže mediánu

Pro proházení prvků v poli se používají dva ukazatele jdoucí proti sobě a hledající prvky, které svou velikostí neodpovídají, tedy zleva se hledá první prvek větší roven pivotovi a zprava menší. Pokud se najdou, prohodí se a pokračuje se dokud se oba ukazatele nepotkají

Časová složitost algoritmu QuickSort:

průměrně : $1{,}39n \log_2 n$,

nejhůře $O(n^2)$

Prostorová složitost algoritmu QuickSort: $O(n)$

Viz http://en.wikipedia.org/wiki/Quicksort

qsort

```
void qsort (void * base, size_t num, size_t size,
            int ( * comparator ) (const void *, const void * ) );
```

- Funkce v stdlib.h
- Implementuje quick sort

```
int main(int argc, char *argv[]) {
    int pole[9] = {1,5,9,3,6,4,89,23,11};
    int i;
    qsort((void*)pole,9,sizeof(int),porovnejInt);
    for(i=0;i<9;i++) printf("%3d, ",pole[i]);
    printf("\n");
    system("PAUSE");
    return 0;
}
```

Qsort, definice porovnávací funkce

```
int porovnejInt(const void *a, const void *b) {
    int aa = *(int*)a;
    int bb = *(int*)b;
    if (aa<bb) return -1;
    if (aa>bb)return 1;
    return 0;
}
```

- parametry jsou netypované ukazatele
- vrací int, jestliže je první prvek menší, pak záporné číslo
- jestliže je větší, pak kladné číslo
- pokud jsou stejné, nulu

Qsort, definice porovnávací funkce

```
/*quicksortStudent.c*/
typedef struct student {
        char jmeno[MAXDELKA];
        char prijmeni[MAXDELKA];
        float prumer;
} Student;

int porovnejPodleProspechu(const void *a, const void *b) {
        Student aa = *(Student*)a;
        Student bb = *(Student*)b;
        if (aa.prumer<bb.prumer)return -1;
        if (aa.prumer>bb.prumer)return 1;
        return 0;
}
qsort((void*)studenti,5,sizeof(Student),porovnejPodleProspechu);
```

Další řadící funkce

```
void  qsort (void *base, size_t nmemb, size_t size,
                int (*compar ) (const void *, const void * ))
void qsort_r (void  *base, size_t nmemb, size_t size, void *thunk,
                int (*compar ) (void *, const void *, const void * ))
int heapsort (void *base, size_t nmemb, size_t size,
                int (*compar ) (const void *, const void * ))
int mergesort (void *base, size_t nmemb, size_t size,
                int (*compar ) (const void *, const void * ))
```

Průměrně platí qsort je rychlejší než mergesort a ten je rychlejší než heapsort

qsort a heapsort jsou nestabilní

mergesort je stabilní (zachovává původní pořadí stejných prvků, vhodné pro řazení podle více kritérií, nejprve podle jména, pak podle příjmení)