1918
TALLINNA
TEHNIKAÜLIKOOL

Programmeerimine II IAG0582

(keele C kasutamine algoritmide realiseerimisel)
III+ moodul

Vladimir Viies viis@ati.ttu.ee, Margit Aarna margit@pld.ttu.ee,
Lembit Jürimägi lembitjyrimagi@gmail.com

Tallinna Tehnikaülikool
2016

Euroopa Liit
Euroopa Sotsiaalfond

Eesti tuleviku heaks

Dünaamiline mälujaotus 1

MÄLU DÜNAAMILINE ERALDAMINE

Dünaamiline mälujaotus I

- Funktsioonid: ***malloc()***, ***calloc()***, ***realloc()*** ja *free()*.
- Kõik nimetatud funktsioonid reserveerivad mälu programmi **kohalikust andmesegmendist**. See on suhteliselt kiire, kuid sellised blokid ei tohi ületada suuruselt **64** KB. Seejuures tuleb meeles pidada, et seda mälu vajab ka programmi pinu. Kui pinu enam suureneda ei saa, kuna te olete reserveerinud kogu mälu andmesegmendist, siis lõpeb programmi töö veateatega.

Dünaamiline mälujaotus II

- Dünaamilist mälu reserveeritakse kahes kohas: programmi andmesegmendis peale initsialiseerimata muutujate piirkonna lõppu (**HEAP**) ja peale programmi kõikide segmentide lõppu üldisest vabast mälust (**FAR HEAP**). Dünaamilise mälu reserveerimine programmi andmesegmendist on natuke kiirem.

Dünaamiline mälujaotus III

- Kui te peate salvestama suure hulga andmeid, siis tuleks reserveerida mälu *üldisest vabast mälust* (FAR HEAP). Selleks kasutatakse funktsioone:
- void far* **farcalloc**(unsigned long nitems, unsigned long size);
- void far* **farmalloc**(unsigned long nsize);
- void far* **farrealloc**(void far* pBuf, unsigned long newsize);
- void **farfree**(void far pBuf);

Dünaamiline mälujaotus IV

- Teadmaks***, kui palju te võite mälu reserveerida***, kasutage funktsioone ***coreleft()*** või ***farcoreleft()*.**
- /* tiny, small ja medium mälumudeliga programmides */
- unsigned coreleft(void);
- /* compact, large ja huge mälumudeliga programmides */
- unsigned long coreleft(void);
- /* kasutatav kõikides programmides väljaarvatud tiny mälumudeliga programmides */
- unsigned long farcoreleft(void);
- Funktsioon *coreleft()* teatab, kui palju mälu on programmi andmesegmendis veel vaba. Funktsioon *farcoreleft()* hangib samad andmed üldise vaba mälu kohta.

TL mälu eraldamine C++

- Mälu tellimiseks on vajalik viitmuutuja.
- C++ tehted **new** ja **delete** vastavalt eraldavad ja vabastavad mälu. Mälueraldus toimub tavaliselt omistuslausega, milles näidatakse kas ainult andmetüüp või lisaks andmetüübile andmeelementide arv:
- viit = new andmetüüp;
- viit = new andmetüüp[n];

Dünaamiline mälujaotus 2

DÜNAAMILISED ANDMESTRUKTUURID

Miks on vaja dünaamilisi andmestruktuure?

Oletame, et programmis on vaja kasutada massiive, mille **elementide (objektide) arv selgub aga** alles programmi töö käigus.

Lahendus:
Kui mingi (näiteks reaalarvude) massiivi M elementide arv ***k*** on programmi töö käigus mingil ajahetkel lõpuks selgunud, siis moodustame programmis massiivi dünaamilises mälus:

```
double *M;
M = malloc(k * sizeof(double));(free(M) -     vabastame mälu)
```

Plussid:
•kasutame seda dünaamilist massiivi täpselt sama moodi nagu „tavalist" massiivi (st nagu massiivi, mille elementide arv on teada juba programmi kompileerimisel) – näiteks: `M[i] = 123.89;`
•kui programmis selline massiiv pole enam vajalik, võime iga hetk vabastada selle jaoks reserveeritud mälu: `free(M);`

Miinused:
•programmi töö käigus on võimatu (ilma suuri „lisakulutusi" tegematta!) muuta **elementide arvu** sellises massiivis

Dünaamilised andmestruktuurid

on programmi osad, mis võimaldavad programmis luua ja kasutada andmeelementide kogumeid kuhu saab programmi töö ajal nii lisada kui ka kõrvaldada mingit tüüpi andmeelemente (objekte).

Dünaamilised andmestruktuurid realiseeritakse arvuti dünaamilises mälus viitmuutujate (*pointers*) abil. Dünaamilised andmestruktuurid kujutavad endast erinevate **abstraktsete andmetüüpide** realisatsioone.

Keerukuselt kõige lihtsam abstraktne andmestruktuur on magasin- e. pinu (ingl. Stack, LIFO - Last In, First Out)

Pinu kasutamine 1

```
#include <stdio.h>
#include <string.h>  /* for strlen() */
#include "stack.h"
#include <stdlib.h>  /* for dynamic allocation */
  // Author: Robert I. Pitts <rip@cs.bu.edu>
/********************** Funktsioonid *********************/
void StackInit(stackT *stackP, int maxSize)
{stackElementT *newContents;
 /* uus element */
newContents = (stackElementT *)malloc(sizeof(stackElementT) *
maxSize);
if (newContents == NULL) {
    fprintf(stderr, "Ei ole piisavalt ruumi pinule\n");
    exit(1);  /* Lahkun veateatega */
  }

  stackP->contents = newContents;
  stackP->maxSize = maxSize;
  stackP->top = -1;  /* tyhi  pinu*/
}
```

Pinu kasutamine 2

```
void StackDestroy(stackT *stackP)
{/* massiivi kaotamine */
  free(stackP->contents);
  stackP->contents = NULL;
  stackP->maxSize = 0;
  stackP->top = -1;  /* tyhi*/
}
void StackPush(stackT *stackP, stackElementT element)
{
  if (StackIsFull(stackP)) {
    fprintf(stderr, "Ei saa elemente panna pinusse - on täis\n");
    exit(1);  /* Lahkun veateatega */
  }
 /* Panen elemendi pinusse,massiivi */
 stackP->contents[++stackP->top] = element;

}
```

Pinu kasutamine 3

```
stackElementT StackPop(stackT *stackP)
{if (StackIsEmpty(stackP)) {
    fprintf(stderr, "Ei saa elemente pinust votta - on tyhi\n");
    exit(1);  /*  Lahkun,vea teade */}
  return stackP->contents[stackP->top--];
}
int StackIsEmpty(stackT *stackP)
{ return stackP->top < 0;}
int StackIsFull(stackT *stackP)
{return stackP->top >= stackP->maxSize - 1;}
/************************ Peaprogramm *********************/
int main(void)
{
  stackT stack;    /* pinu symbolitele,stringile */
  char str[101];   /* kasutaja strigile muutuja kirjeldus */
//viit strigis liikumiseks,stringi sisestus* mitte rohkem kui 100 symbolit
  char *traverse; printf("Sisesta string: ");
  gets(str);  /* loe rida */
  /* Initializeeri pinu ja seda piisavalt suurena. */
  StackInit(&stack, strlen(str));
```

Pinu kasutamine 4

```
//Liigu strigi pidi ja aseta iga symbol pinusse
  for (traverse = str; *traverse != '\0'; traverse++) {
    StackPush(&stack, *traverse);
    printf("%c", *traverse);
    printf("\n"); /*pinusse tulekud*/
  } /*Vota iga symbol pinust ja prindi*/
   printf("\nPinust  valjastatud symbolid: ");
 while (!StackIsEmpty(&stack)) {
    printf("%c", StackPop(&stack));
  }
printf("\n");
StackDestroy(&stack);
  getchar();
  return 0;
}
```

LIFO dünaamilisel massiivil C++ stiilis

```
void init( LiFo & stack, int max, char * title )
// algseisu viimine
{stack.array = new int[max];
   // reserveerime mälu massiivile
     // if( !stack.array ) pole mälu!
     stack.maxHigh = max;
   // maksimaalne elementide arv
     stack.count = 0;          // stack on tühi
     int length = strlen(title);
     stack.name = new char[length+1];
 // reserveerime mälu nimetusele
     strcpy( stack.name, title );
}
```

Ühendstruktuur

- Väliselt sarnane struktuurile, kuid komponendid jagavad sama mälupiirkonda ja ühendstruktuuri pikkuse määrab ära tema pikim komponent. Millist andmeelementi ühendstruktuur mingil hetkel sisaldab, peab olema võimalik programmis mingil moel otsustada. Järgmine deklaratsioon tähendab, et ‘sisu’ võib olla kas täisarv või viit tekstistringile:
-
- union sisu{
- int arv;
- char *tekst;};
-

Loendustüüp

- Programmi teksti loetavuse parandamiseks võib täisarvulistele väärtustele anda nimed. Loendustüübi deklaratsioon näitab vastavust nimede ja väärtuste vahel. Esimene väärtus jadas on vaikimisi 0 ja iga järgmine eelmisest ühe võrra suurem.
- enum nimi{
- väärtus1,
- väärtus2, ...};
- Arvväärtuste ja nimede vastavust saab deklaratsioonis näidata, kui vaikimisi nummerdusviis ei sobi. Järgmises kirjelduses samastatakse ‘esmaspaev’ ühega, ‘teisipaev’ kahega jne

TL näide, eritüübid

- Programmis moodustatakse ja väljastatakse elementaarne andmebaasikirjeldus. Lubatud andmetüüpe on kolm – täisarv, tekst ja tõeväärtus. Tähistatud on nad ühendstruktuuris ‘Ityyp’ vastavalt arvudega 1, 2 ja 3. Täisarvu puhul on andmebaasi kirjelduses tema suurim väärtus (‘maks’), teksti puhul pikkus (‘pikkus’) ja tõeväärtuse puhul stringina ‘valik’ tähed, mis tähistavad vastavalt jaatust ja eitust. Kuna nimetatud omadustest võib esineda iga andmeelemendi puhul vaid üks, on nad paigutatud ühendstruktuuri ‘omadus’. Objektiklass ‘andmeelement’ sisaldab andmetüüpi ‘atyyp’, tüübist sõltuvat ühendstruktuuri ‘omadus’ ja kahte samanimelist meetodit ‘kirjeldus’ nende andmete väärtustamiseks. Kumba neist meetodist kasutada, otsustatakse parameetrite tüüpide alusel. Massiiv ‘andmebaas’ sisaldab kolme objekti, mis väärtustatakse meetodite ‘kirjeldus’ abil ja seejärel väljastatakse, mida äsja loodud massiiv sisaldab.

TL näide C++

```
class andmeelement{
    public:
        int atyyp;
// 1 - int, 2 - string, 3 – logical
        union{
            int maks;
            int pikkus;
            char valik[3];}omadus;
        void kirjeldus(int a, int b)
                {atyyp=a; omadus.maks=b;}
        void kirjeldus(int a , char* b)
                {atyyp=a; strcpy(omadus.valik, b);}
    }andmebaas[3];
```

SILUMINE JA VEATÖÖTLUS

Silumine ja veatöötlus I

- Programmi silumisel kasutatakse enamasti kolme järgnevat silumisvahendit kombineeritult.
- Katkestuskoha (***Breakpoint*****)** määramine. Programmi töö peatab antud real ja avaneb võimalus uurida, mis on toimunud programmi senise töö tulemusena.
- Muutuja väärtuste jälgimine (***Watch***). Valitakse muutujad, mille väärtust kuvatakse eraldi aknas.
- Programmi sammhaaval täitmine (***Step***). Alates katkestuskohast täidetakse programmi lauseid ükshaaval. Tavaliselt saab valida, kas funktsiooni väljakutset pidada üheks lauseks (*Step Over*) või tuleb sammhaaval täita ka funktsiooni laused (*Step Into*).
-

Silumine ja veatöötlus II

- Eelprotsessori tingimuslik transleerimine lubab kerge vaevaga lisada või eemaldada silumislaused vaid ühe rea abil:
-
- #define DEBUG
-
- Silumislaused ümbritsetakse eelprotsessori käskudega #ifdef ja #endif:
-
- #ifdef DEBUG
- for(int j=0;j<=i;j++)printf("%3d",loos[j]);printf("\n");
- #endif

Silumine ja veatöötlus III

- Veatöötlusega programmiploki ette kirjutatakse võtmesõna **try**:
- try {programmikäsud} – käskude täitmine veakoodi väärtsustamiseni (tavaliselt väljakutsutavas funktsioonis).
- Vea tekkimisel “püütakse” ta kinni **catch**-lausega:
- catch (andmetüüp){programmiplokk} – plokki kuuluvad laused täidetakse näidatud tüüpi veakoodi puhul. Tavaliselt on veakood täisarv.
- Veast teatamine väljakutsunud funktsioonile toimub võtmesõnaga **throw**:
- throw veakood – veast teadaandmine.

Silumine ja veatöötlus IV

- Veatöötluse näiteks on funktsioon ‘minutid’, mille parameetriks on kellaaeg stringina kujul “hh:mm” ja väärtuseks aeg minutites päeva algusest. Kellaaja teisendamine tundideks ja minutiteks toimub funktsiooniga ‘atoi’, funktsioon ‘strchr’ leiab kooloni asukoha parameetrina edastatud stringis. Funktsioon avastab kolm võimalikku viga ja annab neist teada järgmiste veakoodidega:
- 1 – tundide arv üle 23;
- 2 – minutid üle 59;
- 3 – kellaaja koosseisus puudub koolon.

Silumine ja veatöötlus V

```
int minutid(char *p){
    int tund, minut;
    char *pos;
    tund=atoi(p); // teksti alguses on tunnid
    if (tund>23) throw 1; // vale tundide arv
    pos=strchr(p,':'); // leiame tekstis kooloni
    minut=(pos==NULL)?-1:atoi(pos+1);
        // teisendame minutid, kui võimalik
    if (minut>59) throw 2; // minutid valed
    if (minut<0) throw 3; // koolon puudus
    return tund*60+minut;
}
```

Silumine ja veatöötlus VI

```
#include <iostream>
using namespace std;

int main(){
    int kogus;
    cout << "Palun arv: ";
    cin >> kogus;
    if(kogus>=10){ throw "Liiga suur";}
    cout << kogus << endl;
    //system("pause");
    return(0);
```

Silumine ja veatöötlus VII

```
#include <iostream>
using namespace std;
int main(){
   int kogus;
   try{
    cout << "Palun arv: ";
    cin >> kogus;
    if(kogus>=10) throw 1; //lihtsalt üks arv teatena
    cout << kogus << endl;
   } catch(int nr){
     cout << "Probleem " << nr << endl;
   }
   //system("pause");
   return(0);
}
```

Silumine ja veatöötlus VIII

```
#include <iostream>
#include <stdexcept>
using namespace std;
class SuurusViga : public runtime_error { //tavavea alamklass
public:
  SuurusViga(const string& teade = "") : runtime_error(teade) {}
};
int main() {
  try {
    int kogus;
    cout << "Palun arv: ";
    cin >> kogus;
    if(kogus>=10)throw SuurusViga("Liiga suur");
    cout << kogus << endl;
  }
  catch (SuurusViga& v) {
    cout << v.what() << endl;
  }//system("pause");
  return(0);
}
```

Silumine ja veatöötlus IX

```
#include <iostream>
#include <stdexcept>
using namespace std;
class SuurusViga : public runtime_error { //tavavea alamklass
public:
  SuurusViga(const string& teade = "") : runtime_error(teade) {}
};
int loeArv(){
    int kogus;
    cout << "Palun arv: ";
    cin >> kogus;
    if(kogus>=10)throw SuurusViga("Liiga suur");
    return kogus;}
int main() {
  try {
    int a=loeArv();
    cout << a << endl;
  }
  catch (SuurusViga& v) {
    cout << v.what() << endl;
  } //system("pause");
  return(0);
}
```

Küsimused, iga vastus annab punkte

- 1.Selgitada slaididel „Silumine ja veatöötlus VI-IX“ toodud programmide tulemused algväärtuse 50 sisestuse korral? Saab2x2p
- 2.Failist loetakse kuupäevad(programmis kasutada class-e ja erindeid),sorteerida kõik kuupäevad kolme faili: Euroopa formaat, USA formaat,“katkine“ kuupäev. 2x3p
- *(kui kuupäev sobib mõlemisse formaati, eelistad Euroopa formaati)*
- Soovides edu!!

Täname, et läbisid mooduli III!
Jätka aine omandamist ja
lahenda kodutööd !

Tutvu ainetega Infotehnoloogia teaduskonna
õppematerjalide kodulehel www.tud.ttu.ee