Vaje 18

Klavdija LUNAR

16.3.1998

Danes si bomo ogledali:

1. domačo nalogo
2. DEBUG
3. sklad z
   - „brez“ max_Sklad, pri katerem si bomo ogledali:
     - malloc
     - free
     - new
     - delete
   - sklad z realnimi števili
   - sklad znakov

Static pomeni, da pozna definirano stvar samo tisti modul, v katerem se nahaja.

DEBUG

Debug ali razhroščevanje. Zgodovina Debug-a:

> Ta izraz se je rodil leta 1948. Takrat so bili računalniki narejeni še iz elektronk. Elektronke pa delajo na 3 oz. 5-ih voltih, včasih pa so delali na 100-ih voltih. V tistem času so naredili program, ki je zmeraj delal narobe (dajal je napačne rezultate). Nekdo je šel pogledat, če je kaj narobe z računalnikom in med žicami so našli majhnega hrošča. In zato se to imenuje debug oz. razhroščevalnik.

Namen debuggiranja je, da poiščemo napako v programu. S testiranjem preverimo, če je program pravilen. Če ni takih napak, da se program vsaj prevede, preizkusimo, če nam npr. pravilno odšteva. Če ugotovimo napake v funkcijah, si pomagamo z delnimi rezultati tako, da jih izpisujemo.

PRIMER: Kako najdemo napako pri programu ART.C?

To naredimo tako, da program najprej prevedemo v knjižnico. Potem gremo pogledat še testen program TESTART.C. Naredimo projekt in napišemo TESTART ter mu dodamo osnovni in objektni program. Potem pa prevedemo v standard. Napisali smo:

```
V123
V456
S
I
```

računalnik izpiše: `579`.

Naredili smo še nekaj preizkusov. Radi pa bi ugotovili, kaj je narobe.

Npr. po scanf se želimo ustaviti. To naredimo tako, da gremo na *Debug*, potem izberemo *Toggle breakpoint oz. Ctrl+F8.*

Watches izpisuje vrednost in v programu so razne spremenljivke, ki lahko opazuješ kako se spreminjajo.

Toggle breakpoint ko izbereš vrstico, se vrstica rdeče obarva in program bo tekel toliko časa, dokler ne pride do te vrstice. Potem te bo vprašal, kaj hočeš.

breakpoints posredujeje spisek vseh točk, kjer se bo Debug ustavil.

`V 100` - program se ustavi, kjer smo prej zahtevali (nastavili).

Radi bi pogledali, če je vrednost 100 v resnici vpisana.

Gremo na:

- *Debug*
- *Toggle breakpoint*
- *Inspect*

Še drugi način: F8 pomeni, da izvajamo program korakoma – vrstica za vrstico. Pritisnemo F8 in se prestavimo, potem Debug, Inspect in napišemo `stev1`, potem vnesemo še `stev2`. Potem spet pritisnemo F8. Ševili seštejemo.

Potem gremo na Run, napišemo I, vendar se še ne bo izpisalo.

V123 Ctrl+F9
V222 Ctrl+F9
O Ctrl+F9
I

Izpiše nam pravilen rezultat `99`.

Radi bi pogledali, kako program dela:

- *Run*
- *Trace into*

ali F7 in smo v programu. Lahko pogledamo kakšni sta `num1` in `num2`. To naredimo tako, da gremo na:

- *Debug*
- *Inspect.*

V Inspect-u napišemo `num1` in Enter ter vidimo, da je `num1` enako 212. Potem naredimo isto še za `num2` in vidimo, da je `num2` enako 131. Potem gremo korak naprej z F8.

Predznak števila ugotovimo tako, da gremo na:

- *Debug*
- *Inspect*
- napišemo `num1.sign`

in isto naredimo za `num2`.

Radi bi videli, kako bi delala funkcija `ABSVecje`. Gremo na Run in potem na Trace (oziroma F7). Pogledali smo vrednost po že znani poti, `num1`, `num2`, stevec v funkciji `ABSVecje`.

Pri `for` (stevec) smo postavili *Toggle breakpoint* – ustavi se, kjer je bila prekinitvena točka.

Vsak program ima za razhroščevanje vsaj 3 ukaze:

Run – izvajaj naprej

Step – izvedi eno vrstico

Trace – izvedi naslednjo vrstico, ki je lahko v podprogramu

Poleg teh pa še ukaz Watch, ki je v povezavi z Inspect.

Ugotovili smo tudi, da ima naš program eno napako pri odštevanju. To napako moramo poiskati.

Sklad „brez“ max_Sklad

Sklad je definiran kot skladišče, ki ima 100 možnih elementov.

$$\text{Software engineering} = \begin{cases} \texttt{sklad\_100} \\ \texttt{sklad\_200} \\ \texttt{sklad\_250} \\ \vdots \\ \text{poljubno mnogo} \end{cases}$$

V našem skladišču moramo imeti prostor za naročene izdelke vseh strank. Če najdemo v kodi, jo moramo popraviti. Naša želja je, da to napako popravimo samo enkrat. Želimo imeti samo en modul.

Želimo narediti program, da bodo vse stranke imele svoje kode še naprej.

Vse kar lahko spreminjamo je, da spremenimo private, ostalo mora biti popolnoma enako.

`malloc` in `free`

To sta takoimenovana c-jevska ukaza. Ko narediš en objekt, se ta objekt naseli v pomnilnik. Nekje v pomnilniku se nahaja naš sklad. Moti nas številka `max_sklad`, ki je v naprej določena. Včasih želimo, da je ta majhna, včasih pa velika.

Definirali bomo, da skladišče ne bo več skladišče, ampak kazalec na skladišče.

```
int * ptrSkladisce;
```

Zvezdica „`*`“ v glavi funkcije pomeni referenco na objekt; tu pa to pomeni, da je kazalec na objekt tipa int.

Ko v stari izvedbi ustvarimo sklad, se nam prostor za skladišče sam od sebe rezervira. Funkcija, ki jo bomo napisali, nam bo to naredila po naših potrebah: `n` – zlogov pomnilnika.

```
Sklad::Sklad() {
  ptr_Skladisce= (int*)malloc (max_Sklad * sizeof(int));
} //Sklad
Sklad::~Sklad() {
 free(ptr_skladisce);   // s tem smo vrnili, kar smo dobili
} //~Sklad
```

Pri `malloc` je int osnovni tip spremenljivke, ki bo ustvarjen. Velikost tega tipa je `sizeof(int)` zlogov – ena spremenljivka zasede toliko zlogov pomnilnika. V našem primeru jih bomo dobili `max_Sklad` krat toliko.

Kako s tem delati? Napisali bomo novo funkcijo z našim novim skladiščem.

```
void Sklad::Nalozi(int element) {
  ptr_skladisce[zadnji]= element;
  zadnji ++;
 } //Nalozi
```

Podrobno si lahko pogledate na:

`Z:\PREDAVANJA\POLDE\CODE\SKLAD\SKLAD.CPP`

V c-ju in c++ so polja čudna stvar – ne obstajajo. Polje je predstavljeno kot *kazalec na prvi element polja.* Če hočeš priti do zadnjega, prišteješ temu kazalcu velikost polja. Da bo pravilno delal, mora biti spremenljivka kazalec na prvi element polja.

`new` in `delete`

Ko naredimo objekt, moramo vsakega od njih kreirati. Namesto `malloc` bomo imeli `new`, namesto `free` pa `delete`. Sedaj nam ni potrebno vec *spreminjati tipa* vrnjene spremenljivke. Namesto:

```
ptr_Skladisce= (int*)malloc (max_Sklad * sizeof(int))
```

napišemo

```
ptr_Skladisce= new int [max_Sklad]
```

Spremenljiva velikost skladišča

Kako bi rešili problem?

Ideja:

> Imamo mladoporočenca, ki trenutno živita v garsonjeri. Ko bosta dobila svojega prvega otroka, se bosta z vso svojo kramo iz garsonjere in z otrokom, preselila v enosobno stanovanje. Ko bosta dobila drugega otroka, se bodo, vsi skupaj in z vso kramo, preselili v dvosobno stanovanje...
>
> Potem pa otroci zrastejo in odidejo od doma, zato se moz in zena preselita, z vso svojo kramo, postopoma nazaj v garsonjero.

To je opis kako bomo uporabljali ukaza `new` in `delete`.

Funkcija `Poln` vrne true natanko takrat, ko je polna. V popravljeni inačici pa funkcija, ko ugotovi, da je skladišče polno, poskuša dobiti nov prostor s pomočjo ukaza `malloc` (`new`). Če ni dovolj velikega pomnilnika, vrneta obe funkciji nič (`NULL`). V tem primeru bomo sporočili, da je sklad poln.

```
bool Sklad::Poln() {
  if(zadnji < Vel_Skladisce) return false;
      // dobi novo skladisce, ki je veliko 2* Vel_Skladisce.
      // ce nismo dobiliskladisca, return true.
      // sicer preselimo skladisce, sprostimo staro skladisce.
      // Vel_Skladisce= 2* Vel_Skladisce, return false.
} // Poln
```

Podrobno si lahko pogledate na:

`Z:\PREDAVANJA\POLDE\CODE\SKLAD\NOVISKLAD.CPP`