Pracovní list 8: Obecné datové struktury. Moduly

Co už máme znát

- práce s dynamickými proměnnými a obecný ukazatel;
- definice datových typů;
- direktivy překladače pro definici makra a test existence makra;
- princip implementace modulu;
- vazba konstrukce modulu na diagram signatury a na sémantické definice ATD;
- odložená definice podprogramu;
- datový typ ukazatele na podprogram;
- princip vnější operace modulu.

Kontrolní otázky

8.1 Jak se definuje obecný ukazatel?

8.2 Jak lze přistupovat k dynamické proměnné, na niž ukazuje obecný ukazatel?

8.3 Z jakých dvou souborů se skládá implementace programového modulu a co je v nich obsaženo?

8.4 Jakými direktivami překladače můžeme zabránit dvojnásobnému překladu určité části zdrojového textu?

8.5 Jak se z diagramu signatury vytvoří odpovídající část modulu?

8.6 Jak se definuje ukazatel na podprogram?

8.7 K čemu může sloužit vnější operace modulu?

8.8 Do kterých míst lze vložit tělo vnější operace modulu při jeho použití?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv08` na serveru akela. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 8.1 Implementujte funkci, která zjišťuje hodnotu nulového bodu zadané funkce v parametru v zadaném intervalu. Předpokládejte, že v tomto intervalu má daná funkce právě jeden nulový bod. Správnost výpočtu ověřte pro knihovní funkci sinus v počátečním intervalu $\langle 1, 4 \rangle$.

Řešení: K výpočtu pozice nulového bodu použijeme algoritmus, který vždy rozpůlí zadaný interval a zjistí, ve které části se nachází nulový bod funkce, tj. ve které části intervalu platí, že funkce mění znaménko. Ten pak dále rozpůlí a znovu zjistí část, v níž je nulový bod funkce. Proces se opakuje tak dlouho, dokud interval nulového bodu není menší než uvažovaná přesnost výpočtu, kterou zde stanovíme na $1 \cdot 10^{-5}$.

Vytvoříme datový typ reálné funkce jedné proměnné, která bude parametrem funkce pro zjištění nulového bodu. Zjištění nulového bodu v zadaném intervalu provedeme testem, zda je součin funkčních hodnot krajních bodů intervalu záporný – tehdy se mění v tomto intervalu znaménko funkčních hodnot.

```
316 #include <iostream>
317 #include <cmath>
318 using namespace std;
319
320 typedef double (*realfce)(double X);
321
322 double nulbod(realfce R, double A, double B){
323   const double EPS = 1E-5;  //požadovaná přesnost
324   double C;
325   if (R(A)*R(B)>0){
326      cerr << "chybně zadaný interval"<<endl;
327      return 0;
328   }
329   while ((B-A)>EPS) {
330     C = (B+A)/2;  //střed intervalu
331     if ((R(A)*R(C))<0) B=C; //nulový bod vlevo, mezi A-C
332     else if ((R(C)*R(B))<0) A=C;  //nulový bod pravo, mezi C-B
333          else A=B; //chyba -- není vůbec nulový bod
334   }
335   return (B+A)/2; //střed nalezeného intervalu
336 }
337
338 int main(){
339   double dolni, horni;
340   cin >> dolni >> horni;
341   cout << "Nulový bod je: " << nulbod(sin, dolni, horni) << endl;
342   return 0;
343 }
```

Příklad 8.2 Vytvořte modul implementující pole obecných složek s operacemi inicializace, vložení prvku na konec, výpis prvku z pozice *i* na standardní výstup. Pro výpis definujte vnější operaci. Modul použijte pro tuto úlohu: Na vstupu se nachází řada desetinných čísel. Uložte je do pole a celé pole pak vypište na standardní výstup v obráceném pořadí. Pro vyzkoušení činnosti použijte soubor `cisla.txt` a pro kontrolu výsledku přesměrujte výsledek do souboru `cisla_obrat.txt`.

Řešení: Pro implementaci potřebujeme hlavičkový soubor `pole.h` a implementační soubor `pole.cpp`. V hlavičkovém souboru nadefinujeme příslušný typ pole a hlavičky operací včetně datového typu vypisovací procedury:

```
#ifndef POLE_H
#define POLE_H

const unsigned int MaxPole = 1000;
typedef void *DatPole[MaxPole]; // pole obecných ukazatelů

typedef struct {
   DatPole pole;  // data
   unsigned int obsaz; // počet obsazených prvků
} TypPole;

typedef void (*TypJak)(void *A);

void Init(TypPole &P);
void Pridej(TypPole &P, void *E);
void Vypis(TypPole P, unsigned int i, TypJak Jak);
#endif
```

V implementačním souboru `pole.cpp` přidáme těla operací:

```
#include <iostream>
#include "pole.h"
using namespace std;

void Init(TypPole &P){
  P.obsaz=0;
}

void Pridej(TypPole &P, void *E){
  P.obsaz++;
  P.pole[P.obsaz-1]=E;
}

void Vypis(TypPole P, unsigned int i, TypJak Jak){
  if (i<P.obsaz) Jak(P.pole[i]);
}
```

Pro použití modulu napíšeme hlavní program v souboru s názvem `pouzij.cpp`:

```
#include <iostream>
#include <iomanip>
#include ”pole.h”
using namespace std;

void Velke(void *X){
   float *M = (float*)X;  //potřebné přetypování
   cout << setprecision(10) << *M << endl; //výpis
}

int main(){
  unsigned int kolik=0;
  float X, *U; //potřebujeme ukazatel na data
  TypPole A;
  Init(A);
  while (cin>>X) {
    U = new float; //nový ukazatel na data
    *U=X;   //naplnění dynamické proměnné
    kolik++;
    Pridej(A, U); //vložení do pole
  }
  for (int i=kolik-1; i>=0; i--) Vypis(A, i, Velke);
  return 0;
}
```

Příklady

Příklad 8.3 Je dána procedura:

```
void ZpracujKvadr(float a, float b, float c, TypZpusob Zpusob){
  cout << ”Vstupní hodnoty jsou: a =” << a <<
    ”b =” << b << ”c =” << c << endl;
  cout << ”Výsledek zpracování:” << endl
    << Zpusob(a, b, c) << endl;}
```

a) Definujte datový typ „TypZpusob“. Všechny informace, které k tomu potřebujete, vyplývají z uvedeného zdrojového textu.

b) Vytvořte tři podprogramy, které mohou být vloženy jako 4. parametr procedury `ZpracujKvadr` a které vypočítávají: i) objem kvádru, ii) povrch kvádru, iii) délku tělesové úhlopříčky kvádru.

c) Přečtěte ze standardního vstupu tři reálná čísla představující rozměry nějakého kvádru a zavolejte třikrát proceduru `ZpracujKvadr`, pokaždé s jinou operací v parametru.

Příklad 8.4 Je dán datový typ:

```
typedef signed char (*TypMax) (void *X, void *Y);
```

Vytvořte dvě funkce vyhovující tomuto datovému typu, které jako výsledek předají hodnotu 1, ukazuje-li první parametr na hodnotu větší než druhý, hodnotu 0, jsou-li hodnoty shodné, a hodnotu −1 ve zbylých případech.

a) První funkce předpokládá, že `X` a `Y` ukazují na hodnoty typu longint; větší číslo je to, které je na číselné ose více vpravo.
b) Vytvořte vlastní datový typ Komplex – záznam představující komplexní čísla ve složkovém tvaru; druhá funkce předpokládá, že `X` a `Y` ukazují na komplexní čísla, kde větší komplexní číslo je to, které má větší absolutní hodnotu.

Příklad 8.5 Předpokládejte, že existuje lineární jednosměrný dynamický seznam, jehož datovými složkami jsou obecné ukazatele. S využitím datového typu úlohy 8.4 napište podprogram, který seřadí data v seznamu řadicí metodou Bubble sort.

Příklad 8.6 Nakreslete diagram signatury pro seznam z úlohy 8.5 – operace: inicializace, vložení nového prvku na konec, seřazení, výpis hodnot na standardní výstup. Pak podle něj vytvořte programový modul implementující tento seznam. Pro řazení použijte podprogram z úlohy 8.5. Pro výpis implementujte veškerou přípravu na použití vnější procedury zpracovávající jednu datovou složku a vypisující do standardního výstupu.

Příklad 8.7 Použijte programový modul z úlohy 8.6 na řešení následující úlohy: Na standardním vstupu je posloupnost komplexních čísel zadaných ve složkovém tvaru. Vypište na standardní výstup tato komplexní čísla seřazená podle absolutní hodnoty. Pro ověření činnosti můžete použít soubor `komplex.txt` přesměrovaný na standardní vstup.

Příklad 8.8 Použijte programový modul z úlohy 8.6 bez jakékoliv úpravy na řešení také následující úlohy: Na standardním vstupu je řada řetězců, na každém řádku jeden. Vypište na standardní výstup tyto řetězce seřazené podle délky vzestupně. Pro ověření činnosti můžete použít soubor `retezce.txt` přesměrovaný na standardní vstup.

Příklad 8.9 Reimplementujte modul z úlohy 8.6 (tj. vytvořte nový modul), který ukládání dat řeší polem s obecnými ukazateli místo lineárním seznamem s obecnými ukazateli. Předpokládejte, že implementační konstanta (maximální počet položek pole) bude stanovena tak, aby vyhověla všem uvažovaným případům. Ponechte stejné operace (podprogramy budou mít stejný počet i název datového typu všech parametrů).

Příklad 8.10 Ověřte, že bez zásahu do programu (kromě výměny názvu modulu) reimplementovaný modul funguje identicky s programy z úloh 8.7 a 8.8.

Příklad 8.11 V modulu z úlohy 8.9 doplňte operaci `Error`, která bude aktivována, nastane-li nějaká chyba při použití modulu (pokus o vložení nové hodnoty do plného pole, pokus o vložení nového prvku s hodnotou `NULL`). Operace bude vypisovat odpovídající chybové hlášení do standardního chybového výstupu.

Příklad 8.12 Proveďte další úpravu modulu z úlohy 8.9: Připravte vše tak, aby tělo operace `Error` mohl alternativně definovat uživatel modulu. Pokud ovšem nic neudělá, bude operace fungovat podle zadání příkladu 8.11.

Co máme po cvičení umět

- Práci s obecnými ukazateli na data v datových strukturách.
- Práci s datovým typem podprogram.
- Vytvoření modulu implementujícího abstraktní datový typ.
- Využití vnějších operací modulu s obecnými daty.

Kontrolní otázky

8.9 Jak se vytvoří a použije datový typ podprogram?

8.10 Jakou výhodu mají datové struktury s obecnými daty?

8.11 Jak se projeví potřeba určité vnější operace v diagramu signatury?

8.12 Jak se definuje a použije programový modul implementující abstraktní datový typ?

8.13 Proč byla použita v lineárním seznamu právě řadicí metoda Bubble Sort?

8.14 Kdy je výhodné uložit tělo vnější operace do dané struktury a kdy je naopak vhodné je předat jako parametr operace?

8.15 Jak se zajistí, že při reimplementaci modulu nebude potřebná změna používajícího programu?