Programování v jazyce C a C++
Příklad na tvorbu třídy

Richter[1]

4. prosince 2017

[1]Ing. Richter Miloslav, Ph.D., UAMT FEKT VUT Brno

Dvourozměrné pole pomocí tříd

Zadání

Navrhněte a napište třídu pro realizace dvourozměrného pole. Napište testovací funkci, která otestuje základní vlastnosti třídy a zároveň bude demonstrovat její použití.

Dvourozměrné pole pomocí tříd

Upřesňující poznámky k řešení

- Pro pole vytvořte třídu CPole.
- Dvourozměrné pole realizujte jako pole prvků třídy CVector, která bude použitá pro jeden řádek pole.
- Pole realizujte pomocí dynamických proměnných - tzn. bude umožňovat libovolný rozměr pole.
- Pro pole navrhněte a napište základní operace (vznik a inicializaci pole, zánik pole, změny rozměrů pole, transponování pole, nastavení a získání hodnoty prvku na dané pozici, ...). Navrhněte operace společné pro všechny datové typy: tj. pole by mělo pracovat i jako pole ukazatelů nebo struktur či tříd (tj. nerealizujte operace jako násobení, sčítání ..., které nejsou společné všem datovým typům).

Dvourozměrné pole pomocí tříd

Upřesňující poznámky k řešení

- autor - autor kódu třídy a jejích funkcí.
- uživatel - ten kdo využívá danou knihovnu (třídu) - ten kdo využívá kód napsaný někým jiným. Používá pouze interface - zveřejněné rozhraní knihovny/třídy.
- Uživatel si tedy musí přečíst návod ke kódu a jeho používání a dodržovat pravidla stanovená autorem. Uživatel nezasahuje do autorova kódu.

Dvourozměrné pole pomocí tříd

Upřesňující poznámky k řešení II

- Následující stránky vás postupně provedou realizací.
- Při řešení uvedou důležité jevy a vlastnosti jazyka

Dvourozměrné pole pomocí tříd

Rozbor řešení

- Je možné rozdělit projekt na více celků (tříd)? Uvažujte použití šablon (stejný kód ale různý datový typ) nebo dědění (mírná obměna kódu). Alternativou je využití existující třídy jako členského prvku jiné třídy.

Dvourozměrné pole pomocí tříd

Rozbor řešení

- Navrhněte hierarchii tříd pro řešení pole a matematického pole (maticový počet).

Vektor
řešení pomocí jednotlivých základních datových typů
styl jazyka C

- Založte si projekt.
- V projektu vytvořte soubor s funkcí main().

Vektor
řešení pomocí jednotlivých základních datových typů
styl jazyka C

- Jaké proměnné budeme potřebovat pro realizaci vektoru? Uvažujte variantu proměnného/dynamického pole.

Vektor
řešení pomocí jednotlivých základních datových typů
styl jazyka C

- Proměnné použité pro pole budou jakého (základního) datového typu a jaká bude jejich funkce?

Vektor
definice proměnných
styl jazyka C

- Nadefinujte pomocí typedef nový datový typ `T`. Ve funkci main vytvořte ukazatel `iData` na tento typ a proměnnou `iLen` pro počet prvků.
- Proměnné vhodně inicializujte.

Vektor
inicializace ukazatele "do nikam"
styl jazyka C

- V jazyce C se pro inicializaci nepoužitého ukazatele používá předdefinovaná hodnota `NULL`. Jelikož její definice nebyla jednotná bylo místo této hodnoty v jazyce C++ zavedeno klíčové slovo `nullptr`.
- Upravte inicializaci ukazatele pomocí tohoto nového klíčového slova.

Vektor
Složený datový typ
styl jazyka C

- Pro vektor máme dvě proměnné, ale ve skutečnosti se váží k proměnné jedné, vektoru, který má pouze více hodnot/složek.
- Jaké jsou nevýhody použití více samostatných proměnných, které ve skutečnosti reprezentují jedinou proměnnou jinou?

- Existuje v jazyce C způsob jak z několika proměnných udělat proměnnou jednu? Jaký?

Vektor
Struktura
styl jazyka C

- Rozšiřte stávající kód o definici struktury CVector se složkami size_t a iData.
- Přepište kód funkce main za použití struktury - včetně inicializace. Proměnnou nazvěte `Vector`.
- Původní kód:

```
typedef double T;
int main() {
size_t iLen = 0;
T* iData = nullptr;
```

Vektor
Inicializace proměnných
styl jazyka C

- Jakou nevýhodu má použitý způsob inicializace?

- Jaké by bylo dobré řešení inicializace?

Vektor
Inicializace proměnných
styl jazyka C

- Jak je prováděna inicializace (autorem kódu v jazyce C) u typu int? Ukažte vznik proměnné bez inicializace, inicializaci hodnotou typu int a hodnotou typu double. Co se při jednotlivých inicializacích děje?

Vektor
Struktura - manipulace s daty
styl jazyka C

- Vezměme část z předchozího příkladu a dopišme řádek. Tento řádek vede k chybě. O jakou chybu se jedná?
- ```
  int main() {
  CVector Vector = {0, nullptr};
  Vector.iLen = 10; // doplněný řádek
  ```
  ```

Vektor
Přístupová práva - skrytí proměnných
styl jazyka C++

- Výhodou by bylo, kdyby přístup k proměnným ve struktuře bylo možné kontrolovat.
- Objektové programování přináší vlastnost *Přístupových práv*. Toto umožňuje rozlišit, ke kterým prvkům (members) může přistupovat pouze autor a které může používat i uživatel - tyto se nazývají interface.
- Objektové programování přináší i možnost "funkcí ve třídě/struktuře". Tyto funkce se nazývají *metody*. Přes tyto metody uživatel manipuluje se skrytými proměnnými.
- Tyto vlastnosti plynou z toho, že data jsou považována za nejcennější součást programu a proto se je snažíme chránit před "běžným uživatelem", jehož činnost kontrolujeme pomocí metod.

Vektor
Přístupová práva - skrytí proměnných
styl jazyka C++

- Jelikož složený datový typ struktura nesplňuje uvedené požadavky na skrývání dat, a musí být zpětně kompatibilní s jazykem C, byl potřeba nový složený datový typ.
- Nový datový typ se jmenuje třída - class.
- Přepište v programu struct na class. K jakým změnám (chybám) při překladu došlo?
- ```
  typedef double T;
  struct CVector {
  size_t iLen ;
  T* iData ;
  };
  int main() {
  CVector Vector = {0, nullptr};
  Vector.iLen = 10;
  ```
  ```

Vektor
Přístupová práva - skrytí proměnných
styl jazyka C++

- Problém je v přístupu k proměnné ve funkci main. Jelikož proměnná je nyní skrytá pro uživatele, není možné s ní pracovat. Skryté části se říká privátní/soukromá.
- `Vector.iLen = 10; // ve funkci main nelze //přistupovat k soukromým prvkům`
- Pro autora třídy je proměnná přístupná. Proto metody ve veřejné sekci mohou přistupovat k soukromým proměnným.

Vektor
Přístupová práva - skrytí proměnných
styl jazyka C++

- Členské prvky (data a metody) ve třídě jsou implicitně privátní - `private` .
- Proveďte v rozpracované třídě přepnutí dat do veřejné sekce (Pouze pro vyzkoušení, v dalším data opět do private sekce). Přepnutí proveďte za začátkem bloku třídy.

Vektor
Přístupová práva - skrytí proměnných
styl jazyka C++ (platí i dále)

- Nyní máme opět data veřejně přístupná - tím jsme si ukázali funkci mechanizmu přístupových práv.
- Jelikož data musí být v sekci private, přesuňte přepnutí na veřejné až za definici proměnných.

Vektor
Terminologie třídy

- Pro manipulaci s proměnnými třídy se používají metody.
- Metodám pro nastavení a čtení členských proměnných se (někdy) říká setter a getter.

- Jak se říká proměnným datového typu třída?

- Jaký je rozdíl mezi metodou a funkcí?

Vektor
Přístup k proměnným (getter)

- Napište do třídy metodu pro získání počtu prvků ve vektoru - GetLen.
- Zavolejte tuto metodu ve funkci main.

Vektor
Inicializace - konstrukce

- Co je to konstruktor. Jaké má vlastnosti a k čemu slouží?

Vektor
Inicializace - konstrukce

- Napište konstruktor bez parametrů - implicitní.

- Ukažte jeho "volání" ve funkci main.

Vektor
Inicializace - konstrukce

- Konverzní konstruktor má jeden parametr. Převádí z hodnoty typu parametru na výsledný typ třídy.
- Napište konverzní konstrukotr z intu, který znamená počet pvků v poli (pvky vynulujte).

- Ukažte "volání" ve funkci main.

Vektor
Inicializace - konstrukce

- Konstruktory mohou dělat různé věci. Záleží na autorovi.
- Napište konverzní konstrukotr z double, který vytvoří pole s jedním prvkem s touto hodnotou.

- Ukažte "volání" ve funkci main.

Vektor
ukončení života proměnné

- Co je to destruktor? Jaké má vlastnosti a k čemu slouží?

Vektor
ukončení života proměnné

- Destruktor může dělat různé věci. Záleží na autorovi.
- Napište destruktor, který vrátí alokovanou paměť.

- Ukažte "volání" ve funkci main.

Vektor
ukončení života proměnné

- Jelikož se nedoporučuje volání destruktoru je lepší napsat metodu pro vyčištění/uložení ... samostatně (Destroy, Clear, Null, Save ...)
- Napište metodu Destroy a použijte ji v destruktoru.

- Ukažte "volání" metody Destroy ve funkci main.

Vektor
Inicializace - konstrukce

- Často je nutné vytvořit kopii existujícího objektu (např. předávání parametrů hodnotou, návratová hodnota). K tomu se využívá kopykonstruktor.
- Kopyk. má jako parametr referenci na třídu jejímž je členem.
- Napište kopykonstruktor.

- Ukažte "volání" ve funkci main.

Vektor
operátory

- Co je to operátor? Jaké má vlastnosti a k čemu slouží?

Vektor
Operátory

- Operátory můžeme rozdělit podle počtu parametrů. Ty s jedním parametrem jsou unární. V hlavičce ale nemají žádný parametr, protože ten se to metody dostane jako "this".
- Napište unární operátor ! . Většinou se používá k signalizaci zda je s proměnnou možné pracovat.

- Ukažte "volání"ve funkci main.

Vektor
Operátory

- Zvláštním typem unárního operátoru je operátor konverzní.
- Konverzní operátor mění objekt třídy na hodnotu jiného typu. Na rozdíl od konverzního konstruktoru, který přemění jiný typ na typ vytvářené třídy.
- Má název výsledného typu a nemá návratovou hodnotu.
- Napište konverzní op. na int, vracející počet prvků vektoru.

- Ukažte "volání" ve funkci main.

Vektor
operátory

- Co je to binární operátor? Jaké má vlastnosti a k čemu slouží?

Vektor
operátory

- Jaký je rozdíl mezi operátorem $=$ a kopykonstruktorem?

Vektor
Operátory

- Napište operátor $=$.

- Ukažte "volání" ve funkci main.

Vektor
Operátory

- Bude operátor fungovat pro a = a; ? Pokud nebude, opravte kód.

- Ukažte "volání" ve funkci main.

Vektor
operátory

- Jaký je rozdíl mezi metodou, která má tělíčko v hlavičkovém souboru a která má metodu definovanou ve zdrojovém (cpp) souboru?

Vektor
Operátory

- Naznačte přesun tělíčka operátoru = do zdrojového souboru a úpravu v hlavičkovém souboru.

Vektor
Operátor výstupu - stream

- Napište operátor pro tisk proměnné. Formát vhodně zvolte.

- Ukažte "volání" ve funkci main.

Vektor
Znovupoužitelnost kódu - šablony

- Vektor je obecný mechanizmus, který se hodí pro jakýkoli datový typ. Jak zajistit, aby mohl být CVector použit pro jakýkoli datový typ? I pro několik různých datových typů současně? Jak se jsmenuje mechanizmus, který to umožňuje? Jak funguje?

Vektor
Šablony

- Přepište třídu CVector na šablony.

Vektor
Šablony

- Kód ze zdrojového souboru se přenese za definici třídy a označí se jako šablona příslušná k datovému typu T.

Vektor
Dědění

- Jaký je účel dědění? Lze využít při tvorbě vektoru s přidáním matematických funkcí?

Vektor
Dědění

- Pomocí dědění ze třídy CVector vytvořte základ třídy CMathVector tak aby byly veřejné metody bázové třídy pro uživatele přístupné i ve třídě odvozené.

Vektor
Využití třídy jako parametru

- Pomocí šablonové třídy CVector vytvořte třídu pro dvourozměrné pole CMatrix. Datovým typem řádkových vektorů bude T. Datovým typem řádků potom CVector<T>.

Vektor
Využití třídy jako parametru

- Přepište šablonovou třídu pro pole tak, aby mohla obsahovat třídu CVector, nebo CMathVector. Typ vektoru bude U. Datovým typem řádkových vektorů bude T. Datovým typem řádků potom U<T>.