КЛАСОВЕ

Класовете са нови типове данни, дефинирани от потребителя. Те могат да обогатяват възможностите на вече съществуващ тип или да представят напълно нов тип данни.

Класовете са подобни на структурите, даже може да се каже, че в някои отношения са почти идентични. В C++ класът може да се разглежда като структура, на която са наложени някои ограничения по отношение на правата на достъп. Отначало ще разгледаме основното, което е приложимо както за класовете, така и за структурите. Затова ще останем в познатите означения на структурите.

Основен принцип на процедурното програмиране е модулния. Програмата се разделя на “подходящи” взаимносвързани части (функции, модули), всяка от които се реализира чрез определени средства. Важен е обаче начинът, по който да става определянето на частите и връзките помежду им. Целта е, следващи промени в представянето на данните да не променят голям брой от модулите на програмата. Разсъждения в тази посока довеждат до подхода *абстракция със структури от данни*, който вече разгледахме. Ще напомним, че при него методите за използване на данните се разделят от методите за тяхното конкретно представяне. Програмите се конструират така, че да работят с “абстрактни данни” – данни с неуточнено представяне. След това представянето се конкретизира с помощта на множество функции, наречени **конструктори,**

мутатори
и
функции за достъп
, които реализират “абстрактните данни” по конкретен начин. Така при решаването на даден проблем се оформят следните нива на абстракцията:

Добра реализация на подхода е тази, при която всяко ниво използва единствено средствата на предходното. Предимствата са, че възникнали промени на едно ниво ще се отразят само на следващото над него. Например, промяна на представянето на данните ще доведе до промени единствено на реализацията на някои от конструкторите, мутаторите или функциите за достъп.

работа с рационални числа:

конструиране на рационално число по зададени две цели числа, представящи съответно неговите числител и знаменател;

извличане на числителя на дадено рационално число;

извличане на знаменателя на дадено рационално число.

Към тях ще добавим и функциите:

промяна на стойността на рационално число чрез въвеждане, например;

извеждане на рационално число.

Реализиране на примитивните операции

Като компоненти на структурата rat ще добавим набор от примитивни операции: конструктори, мутатори и функции за достъп. Ще ги реализираме като член-функции.

а) конструктори

Конструкторите са член-функции, чрез които се инициализират променливите на структурата. Имената им съвпадат с името на структурата. Ще дефинираме два конструктора на структурата rat:

rat() – конструктор без параметри и

rat(int, int) – конструктор с два цели параметъра.

Първият конструктор се нарича още **конструктор по подразбиране**. Използва се за инициализиране на променлива от тип rat, когато при дефиницията й не са зададени параметри. Ще го дефинираме така:

```
rat::rat()

{numer = 0;

denom = 1;

}
```

б) мутатори

Това са функции, които променят данните на структурата. Ще дефинираме мутатора read(), който въвежда от клавиатурата две цели числа (второто различно от нула) и ги свързва с абстрактните данни numer и denom.

```
void rat::read()

{cout > numer;

do

{cout > denom;

} while (denom == 0);

}
```

След обръщението

```
p.read();
```

стойността на p *се променя* като полетата й numer и denom се свързват с въведените от потребителя стойности за числител и знаменател съответно.

в) функции за достъп

Тези функции **не променят** член-данните на структурата, а само извличат информация за тях. Последното е указано чрез използването на запазената дума const, записана след затварящата скоба на формалните параметри и пред знака ;. Ще дефинираме следните функции за достъп:

```
int get_numer() const;

int get_denom() const;

void print() const;
```

Първата от тях извлича числителя, втората – знаменателя, а третата извежда върху екрана рационалното число numer/denom.

Езикът C++ позволява да се ограничи свободата на достъп до членовете на структурата като се поставят подходящи спецификатори на достъп в декларацията й. Такива спецификатори са private и public. Записват се като етикети. Всички членове, следващи спецификатора на достъп private, са достъпни само за член-функциите в декларацията на структурата.Всички членове, сменящи спецификатора на достъп public, са достъпни за всяка функция, която е в областта на структурата. Ако са пропуснати спецификаторите за достъп, всички членове са public(както в случая). Има още един спецификатор за достъп- protected, който е еднакъв със спецификатора private, освен ако структурата не е част от йерархия на класове. Спецификаторът private, забранява използването на член-данните numer и denom извън класа. Получава се *скриване* на информация, което се нарича още **капсолиране на информация**. Член-функциите на класа rat са обявени като public. Те са видими извън класа и могат да се използват от външни функции. Затова public-частта се нарича още
интерфейсна част на класа
или само
интерфейс
. Чрез нея класът комуникира с външната среда. Освен функции, интерфейсът може да съдържа и член-данни, но засега ще се стараем това да не се случва.

*Ще отбележим, че конструкторите се използват само когато се създават обекти. Опитите за **промяна** на обект чрез обръщение към конструктор предизвикват грешки.*

Дефиниране на класове

Класовете осигуряват механизми за създаване на напълно нови типове данни, които могат да бъдат интегрирани в езика, а също за обогатяване възможностите на вече съществуващи типове. Дефинирането на един клас се състои от две части:

декларация на класа и

дефиниция на неговите член-функции (методи).

1. Декларация на клас

Декларацията на клас се състои от заглавие и тяло. Заглавието започва със запазената дума class, следвано от името на класа. Тялото е заградено във фигурни скоби. След скобите стои знакът “;” или списък от обекти. В тялото на класа са декларирани членовете на класа (член-данни и член-функции) със съответните им нива на достъп. Фиг. 14.1 илюстрира *непълно* (но достатъчно за целите на настящите разглеждания) синтаксиса на декларацията на клас.

Декларация на клас

::=

::= class $[]_{опц}$

::= {;

$\{;\}_{опц}$

$[]_{опц}$;

::=

||

|

::=

$[:]_{опц}()$

::=

[:]$_{опц}$

()

::=

[:]$_{опц}$

() const;

::= private | public | protected

:: | void |

{, }$_{опц}$

::= [&|$_{опц}$ * [const]$_{опц}$] $_{опц}$

::= {, }$_{опц}$

::= |

::=

[= ()]опц

{,[=()]опц }опц

{, ()}опц

{, = }опц

където , , , и са идентификатори, а е определено в Глава 8.

Фиг. 14.1 Декларация на клас

За имената на класовете важат същите правила, които се прилагат за имената на всички останали типове и променливи. Също като при структурите името на класа може да бъде пропуснато.

Препоръчва се член-данните да се декларират в нарастващ ред по броя на байтовете, необходим за представянето им в паметта. Така за повечето реализации се получава оптимално изравняване до дума.

Забележка: Типът на член данна на клас не може да съвпада с името на класа, но типът на член функция на клас не може да съвпада с името на класа.

В тялото, някои декларации на членове могат да бъдат предшествани от **спецификатор**

ите на достъп
private, public или protected. Областта на един спецификатор на достъп започва от спецификатора и продължава до следващия спецификатор. Подразбиращ се спецификатор за достъп е private. Един и същ спецификатор на достъп може да се използва повече от веднъж в декларация на клас.

Препоръчва се, ако секция public съществува, да бъде първа в декларацията, а секцията private да бъде последна в тялото на класа.

Достъпът до членовете на класовете може да се разгледа на следните две нива:

- По отношение на *член-функциите в класа* е в сила, че те имат достъп до всички членове на класа.

- По отношение на *функциите, които са външни за класа*, режимът на достъп са определя от начина на деклариране на членовете.

Членовете на даден клас, декларирани като private (декларирани след запазената дума private) са видими (достъпни) само в рамките на класа. Външните функции нямат достъп до тях. По подразбиране членовете на класовете са private.

Чрез използването на членове, обявени като private, се постига скриване на членове за външната за класа среда. Процесът на скриване се нарича още **капсолиране на информацията** .

Членовете на клас, които трябва да бъдат видими извън класа (да бъдат достъпни за функции, които не са методи на дадения клас) трябва да бъдат декларирани като public (декларирани след запазената дума public). Всички методи на класа rat са декларирани като public и следователно могат да се използват навсякъде в програмата за работа с рационални числа.

Освен като private и public, членовете на класовете могат да бъдат декларирани и като protected. Тъй като този спецификатор на достъп има отношение към производните класове и процеса на наследяване, разглеждането му засега ще бъде отложено. Ще отбележим, че ако в класа rat заменим private с protected, поведението на класа няма да се промени.

Дефиниране на методите на клас

След декларирането на клас, трябва да се дефинират неговите методи. Дефинициите са аналогични на дефинициите на функции, но името на метода се предшества от името на класа, на който принадлежи метода, следвано от оператора за принадлежност :: (Нарича се още оператор за област на действие). Такива имена се наричат **пълни**. (Операторът :: е ляво-асоциативен и с един и същ приоритет със (), [] и ->). На Фиг. 14.2 е даден синтаксисът на дефиницията на метод на клас.

Дефиниция на метод на клас

::=

$[]_{опц}$ ::() $[const]_{опц}$

{}

::=

където и са идентификатори, а се определя както в дефиниция на функция.

Фиг. 14.2 Дефиниция на метод на клас

Ще отбележим, че дефиницията на конструктор **не започва** с , а запазената дума const може да присъства само в дефинициите на функциите за достъп.

Допълнение (вградени функции) С цел повишаване на бързодействието, езикът C++ поддържа т.нар. вградени функции. Кодът на тези функции не се съхранява на едно място, а се копира на всяко място в паметта, където има обръщение към тях. Използват се като останалите функции, но при деклариране и дефиниране им заглавието им се предшества от модификатора inline.

Пример:

#include

inline int f(int, int); // декларация на вградената функция f

void main()

{cout