2

Основни елементи от програмирането на C++

C++ е език за обектно-ориентирано програмиране. Създаден е от Бярне Страуструп от AT&T Bell Laboratories в края на 1985 година. C++ е разширение на езика C в следните три направления:

- създаване и използване на абстрактни типове данни;
- обектно-ориентирано програмиране;
- подобрения на конструкции на езика C (производни типове, наследяване, полиморфизъм).

През първите шест месеца след описанието му се появиха над 20 търговски реализации на езика, предназначени за различни компютърни системи. От тогава до сега C++ се разраства чрез добавяне на много нови функции и затова процесът на стандартизацията му продължава и до момента. C++ е пример за език, който с времето расте и се развива. Всеки път, когато потребителите му са забелязвали някакви пропуски или недостатъци, те са го обогатявали със съответните нови възможности.

За разлика от C++, езикът Паскал е създаден планомерно главно за целите на обучението. Проф. Вирт добре е проектирал и доказал езика. Тъй като Паскал е създаден с ясна цел, отделните му компоненти са логически свързани и лесно могат да бъдат комбинирани. Разрастващите се езици, към които принадлежи C++ са доста объркани тъй като хора с различни вкусове правят различни нововъведения. Освен това, заради мобилността на програмите, не е възможно премахването на стари конструкции, даже да съществуват удобни техни подобрения. Така разрастващият се C++ събира в себе си голям брой възможности, които не винаги добре се съвместяват.

Езиците, създадени от компетентни хора, по принцип са лесни за научаване и използване. Разрастващите се езици обаче държат

монопола на пазара. Сега C++ е водещия език за програмиране с общо предназначение. Лошото е, че не е много лесен за усвояване, има си своите неудобства и капани. Но той има и огромни приложения – от програми на ниско, почти машинно ниво, до програми от най-висока степен на абстракция.

Целта на настоящия курс по програмиране е не да ви научи на всички възможности на C++, а на изкуството и науката програмиране.

При началното запознаване с езика, възникват два естествени въпроса:

- *Какво е програма на C++ и как се пише тя?*
- *Как се изпълнява програма на C++?*

Ще отговорим на тези въпроси чрез пример за програма на C++, след което ще дадем някои дефиниции и основни означения.

2.1. Пример за програма на C++

Задача 1. Да се напише програма, която намира периметъра и лицето на правоъгълник със страни 2,3 и 3,7.

Една програма,която решава задачата е следната:

```
// Program Zad1.cpp
#include <iostream.h>
int main()
{double a = 2.3;
 double b = 3.7;
 double p, s;
 /* намиране на периметъра
    на правоъгълника */
 p = 2*(a+b);
 /* намиране на лицето на правоъгълника */
 s = a*b;
/* извеждане на периметъра */
 cout << "p= " << p << "\n";
 /* извеждане на лицето */
 cout << "s= " << s << "\n";
 return 0;
}
```

Първият ред

```
// Program Zad1.cpp
```

е коментар в езика C++. Започва с // и завършва в края на реда (ENTER). След знаците // е записан текст, предназначен за програмиста и подсещащ за смисъла на следващото действие. В случая коментарът информира, че следващият фрагмент е програма на езика C++ с име Zad1.cpp и по-точно, че следващата програма е записана във файл с име Zad1.cpp.

Линията

```
#include <iostream.h>
```

е **директива към компилатора на C++**. Чрез нея към файла Zad1.cpp, съдържащ програмата, се включва файлът с име iostream.h (При някои реализации на C++ разширението ".h" се пропуска). Този файл съдържа различни дефиниции и декларации, необходими за реализациите на операциите за поточен вход и изход. В програма Zad1.cpp се нуждаем от тази директива заради извеждането върху екрана на периметъра и лицето на правоъгълника.

Конструкцията

```
int main()
{ …
   return 0;
}
```

дефинира **функция**, наречена **main** (главна). Всяка програма на C++ трябва да има функция main. Повечето програми съдържат и други функции освен нея.

Дефиницията на main започва с **думата** int (съкращение от integer), показваща, че main връща цяло число, а не дроб или низ, например. Между фигурните скоби { и } е записана редица от **дефиниции** и **изпълними изрази** (**оператори**), която се нарича **тяло на функцията**. Компонентите на тялото се отделят със знака ; и се изпълняват последователно. С оператора return се означава краят на функцията. Стойността 0 означава, че тя се е изпълнила успешно. Ако програмата завърши изпълнението си и върне стойност различна от 0, това означава, че е възникнала грешка.

Конструкциите

```
double a = 2.3;
double b = 3.7;
double p, s;
```

дефинират **променливите** a, b, p и s от реалния тип double, като в първите два случая се дават начални стойности на a и b (2.3 и 3.7

съответно). Казва се още, че **a и b са инициализирани с 2.3 и 3.7 съответно**.

Променливата е място за съхранение на данни, което може да съдържа различни стойности по време на изпълнение на програмата. Означава се чрез редица от букви, цифри и долна черта, започваща с буква или долна черта. Променливите имат три характеристики: **тип**, **име** и **стойност**. Преди да бъдат използвани, трябва да бъдат дефинирани.

C++ е строго типизиран език за програмиране. Всяка променлива има тип, който *явно* се указва при дефинирането ѝ. Пропускането на типа на променливата води до сериозни грешки. Фиг. 2.1. илюстрира непълно дефинирането на променливи.

Дефиниране на променливи

Синтаксис

<име_на_тип> <променлива> [= <израз>]$_{опц}$
{, <променлива> [= <израз>]$_{опц}$}$_{опц}$;

където

<име_на_тип> е дума, означаваща име на тип като int, double и др.;

<израз> е правило за получаване на стойност – цяла, реална, знакова и друг тип, съвместим с <име_на_тип>.

Семантика

Дефиницията свързва променливата с множеството от допустимите стойности на типа, от който е променливата или с конкретна стойност от това множество. За целта се отделя определено количество оперативна памет (толкова, колкото да се запише най-голямата константа от множеството от допустимите стойности на съответния тип) и се именува с името на променливата. Тази памет е с неопределена стойност или съдържа стойността на указания израз, ако е направена инициализация.

Пример:

```
double a = 2.3;
double b, p, s;
```

Не се допуска една и съща променлива да има няколко дефиниции в рамките на една и съща функция.

Фиг. 2.1 Дефиниране на променливи

Забележка: При описанието на синтаксиса, означенията […] и {…} от езика на Бекус-Наур, ще завършваме с индекса опц.

В случая на програмата Zad1.cpp за a, b, p и s се отделят по 8 байта оперативна памет и

ОП

a	b	p	s
2.3	3.7	-	-
8 байта	8 байта	8 байта	8 байта

Неопределеността на p и s е означена с -.

След дефинициите на променливите a, b, p и s е разположен коментарът

/* намиране на периметъра на правоъгълника */

Той е предназначен за програмиста и подсеща за смисъла на следващото действие.

Коментарът (Фиг. 2.2) е произволен текст, ограден със знаците /* и */ или от // и ENTER. Игнорира се напълно от компилатора. Ще напомним, че средите за програмиране включват специална програма, наречена *транслатор (компилатор или интерпретатор),* която превежда програмата в програма, записана на машинен език. В средата Visual C++ 6.0 транслаторът е реализиран като компилатор.

Коментар

Синтаксис

<коментар> ::= /* <редица_от_знаци> */ |
// <редица_от_знаци> ENTER

Семантика

Пояснява програмен фрагмент. Предназначен е за програмиста. Игнорира се от компилатора на езика.

Примери:

```
/* Намиране на лицето
    и периметъра на правоъгълник */
// Program Zad1.cpp
```

Фиг. 2.2 Коментар

Конструкциите

```
p = 2*(a+b);
s = a*b;
```

са **оператори за присвояване на стойност** (Фиг. 2.3). Чрез тях променливите p и s получават текущи стойности. Операторът

```
p = 2*(a+b);
```

пресмята стойността на аритметичния израз 2*(a+b) и записва полученото реално число (в случая 12.0) в паметта, именувана с p. Аналогично, операторът

```
s = a*b;
```

пресмята стойността на аритметичния израз a*b и записва полученото реално число (в случая 8.51) в паметта, именувана със s.

По-подробно ще разгледаме този оператор в следващата глава. На този етап оставяме с интуитивната представа за <израз>.

Оператор за присвояване

Синтаксис

<променлива> = <израз>;

като <променлива> и <израз> са от един и същ тип.

Семантика

Пресмята стойността на <израз> и я записва в паметта, именувана с променливата от лявата страна на знака за присвояване =.

Пример:

```
p = 2*(a+b);
```

Фиг. 2.3 Оператор за присвояване на стойност

Да се върнем към дефинициите на променливите a и b и операторите за присвояване и return на Zad1.cpp. Забелязваме, че в тях са използвани два вида числа: **цели** (2 и 0) и **реални** (2.3 и 3.7). Целите числа се записват като в математиката, а при реалните, знакът запетая се заменя с точка. Умножението е отбелязано със *, а събирането – с +. Забелязваме също, че изразите 2*(a+b) и a*b са реални, каквито са и променливите p и s от левите страни на знака = в операторите за присвояване.

Конструкциите

```
cout << "p= " << p << "\n";
cout << "s= " << s << "\n";
```

са оператори за извеждане. Наричат се още оператори за поточен изход. Еквивалентни са на редицата от оператори:

```
cout << "p= ";
```

```
   cout << p;
   cout << "\n";
   cout << "s= ";
   cout << s;
   cout << "\n";
```

Операторът **<<** означава “изпрати към”. Обектът (променливата) cout (произнася се “си-аут”) е името на стандартния изходен поток, който обикновено е екрана или прозорец на екрана.

Редица от знаци, оградена в кавички, се нарича **знаков низ**, или **символен низ**, или само **низ**. В програмата Zad1.cpp “p= “ и “s= “ са низове. Низът “\n“ съдържа двойката знаци \ (backslash) и n, но те представляват един-единствен знак, който се нарича **знак за нов ред**. Операторът

```
   cout << "p= ";
```

извежда върху екрана низа p=

Операторът

```
   cout << p;
```

извежда върху екрана стойността на p, а

```
   cout << "\n";
```

премества курсора на следващия ред на екрана, т.е. указва следващото извеждане да бъде на нов ред.

Фиг. 2.7. показва по-детайлно синтаксиса и семантиката на оператора за извеждане.

Изпълнение на Zad1.cpp

След обработката на директивата

```
   #include <iostream.h>
```

файлът iostream.h е включен във файла, съдържащ функцията main на Zad1.cpp. Изпълнението на тялото на main започва с изпълнение на дефинициите

```
   double a = 2.3;
   double b = 3.7;
   double p, s;
```

в резултат, на което в ОП се отделят по 8 байта за променливите a, b, p и s, т.е.

```
   ОП
   a     b     p     s
   2.3   3.7   -     -
```

Коментарите се пропускат. След изпълнението на операторите за присвояване:

```
p = 2*(a+b);
s = a*b;
```

променливите p и s се свързват с 12.0 и 8.51 съответно, т.е.

```
ОП
a     b        p         s
2.3   3.7     12.0      8.51
```

Операторите

```
cout << "p= " << p << "\n";
cout << "s= " << s << "\n";
```

извеждат върху екрана на монитора

```
p= 12
s= 8.51
```

Изпълнението на оператора

```
return 0;
```

преустановява работата на програмата сигнализирайки, че тя е завършила успешно.

Забележка: Реалните числа се извеждат с възможно минималния брой знаци. Така реалното число 12.0 се извежда като 12.

В същност, описаните действия се извършват над машинния еквивалент на програмата Zad1.cpp. А как се достига до него?

Изпълнение на програма на езика C++

За целта се използва някаква среда за програмиране на C++. Ние ще използваме Visual C++ версия 6.0.

Изпълнението се осъществява чрез преминаване през следните стъпки:

1. Създаване на изходен код

Чрез текстовия редактор на средата, текстът на програмата се записва във файл. Неговото име се състои от две части – име и разширение. Разширението подсказва предназначението на файла. Различно е за отделните реализации на езика. Често срещано разширение за изходни файлове е “.cpp” или “.c”.

Примерната програма е записана във файла Zad1.cpp.

2. Компилиране

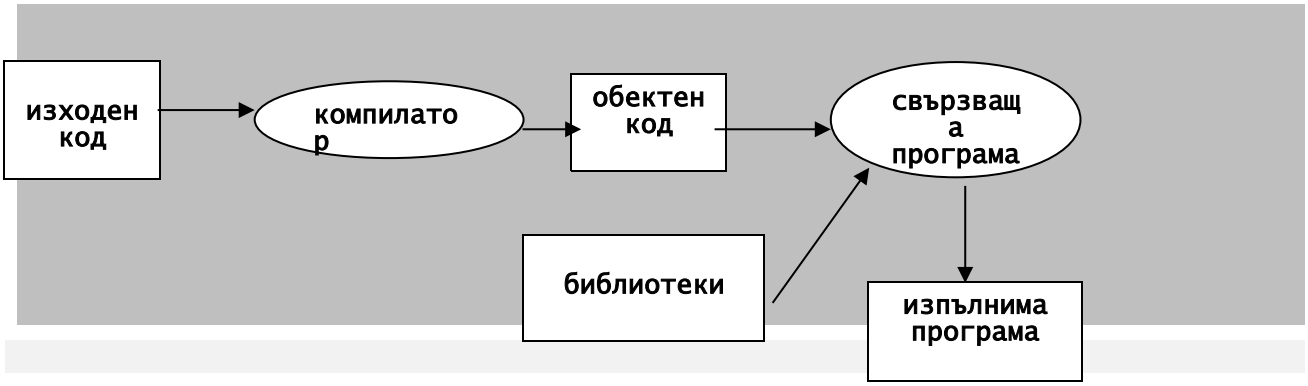
Тази стъпка се изпълнява от компилатора на езика. Първата част от работата на компилатора на C++ е откриването на грешки – синтактични и грешки, свързани с типа на данните. Съобщението за

грешка съдържа номера на реда, където е открита грешка и кратко описание на предполагаемата причина за нея. Добре е грешките да се коригират в последователността, в която са обявени, защото една грешка може да доведе до т. нар. “каскаден ефект”, при който компилаторът открива повече грешки, отколкото реално съществуват. Коригираният текст на програмата трябва да се компилира отново. Втората част от работата на компилатора е превеждане (транслиране) на изходния (source) код на програмата в т. нар. **обектен (object) код**. Обектният код се състои от машинни инструкции и информация за това, как да се зареди програмата в ОП, преди да започне изпълнението ѝ. Обектният код се записва в отделен файл, обикновено със старото име, но с разширение “.obj” или “.o”.

Обектният файл съдържа само “превода” на програмата, а не и на библиотеките, които са декларирани в нея (в случая на програмата Zad1.cpp файлът Zad1.obj не съдържа обектния код на iostream.h). Авторите на пакета iostream.h са описали всички необходими действия и са записали нужния машинен код в библиотеката iostream.h.

3. Свързване

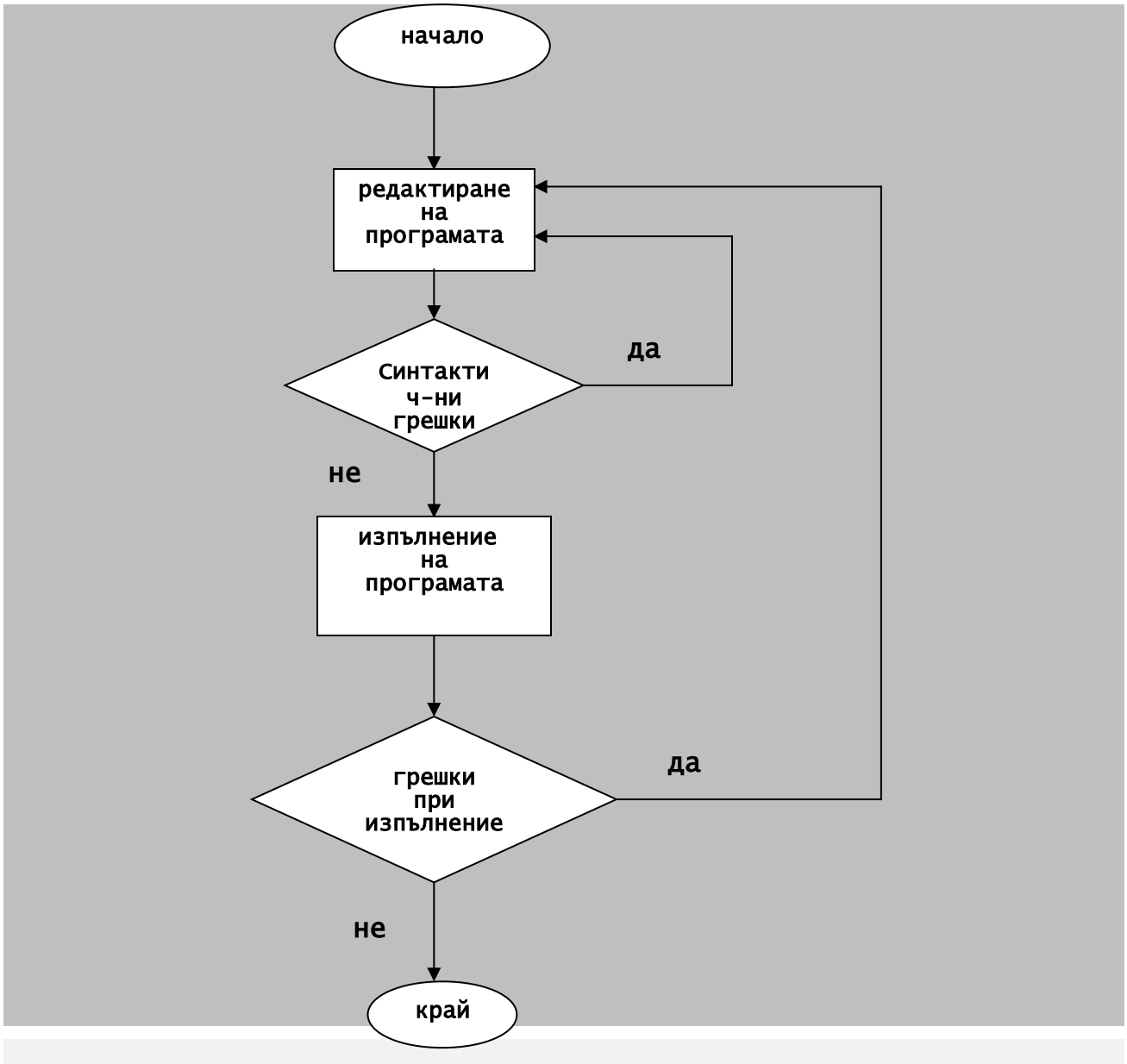
Обектният файл и необходимите части от библиотеки се свързват в т. нар. **изпълним файл**. Това се извършва от специална програма, наречена **свързваща програма** или **свързващ редактор** (**linker**). Изпълнимият файл има името на изходния файл, но разширението му обикновено е “.exe”. Той съдържа целия машинен код, необходим за изпълнението на програмата. Този файл може да се изпълни и извън средата за програмиране на езика C++. Фиг. 2.4 илюстрира стъпките на изпълнение на програма на C++.



Фиг. 2.4 Изпълнение на C++ програма

Програмистката дейност, свързана с изпълнението на програма на C++, преминава през три стъпки като реализира цикъла “редактиране-

компилиране-настройка". Започва се с редактора като се пише изходният файл. Компилира се програмата и ако има синтактични грешки, чрез редактора се поправят грешките. Когато програмата е "изчистена" от синтактичните грешки, започва изпълнението й. Ако възникнат грешки по време на изпълнението, осъществява се връщане отново в редактора и се поправят предполагаемите грешки. После пак се компилира и стартира програмата. Цикълът "редактиране-компилиране-настройка" е илюстриран на Фиг. 2.5.



Фиг. 2.5 Цикъл редактиране-компилиране-настройка

2.2. Основни означения

Всяка програма на C++ е записана като редица от знаци, които принадлежат на **азбуката на езика.**

Азбука на C++

Азбуката на езика включва:

- главните и малки букви на латинската азбука;
- цифрите;
- специалните символи

  + - * / = ( ) [ ] { } | : ; “ ‘ < > , . _
  ! @ # $ % ^ ~

Някой от тези знаци, по определени правила, са групирани в думи (лексеми) на езика.

Думи на езика

Думите на езика са идентификатори, запазени и стандартни думи, константи, оператори и препинателни знаци.

Идентификатори

Редица от букви, цифри и знака за подчертаване (долна черта), започваща с буква или знака за подчертаване, се нарича **идентификатор**.

Примери:

Редиците от знаци

A12 help Help double int15_12 rat_number INT1213 Int15_12

са идентификатори, а редиците

1ba ab+1 a(1) a’b

не са идентификатори. В първия случай редицата започва със цифра, а в останалите – редиците съдържат недопустими за идентификатор знаци.

Идентификаторите могат да са с произволна дължина. В съвременните компилатори максималният брой знаци на идентификаторите може да се задава, като подразбиращата се стойност е 32.

Забележка: При идентификаторите **се прави разлика** между малки и главни букви, така help, Help, HELP, HeLp и HElp са различни идентификатори.

Идентификаторите се използват за означаване на имена на променливи, константи, типове, функции, класове, обекти и други компоненти на програмите.

Препоръка: Не започвайте вашите идентификатори със знака за подчертаване. Такива идентификатори се използват от компилатора на C++ за вътрешно предназначение.

Допълнение: Чрез метаезика на Бекус-Наур, синтаксисът на променливите се определя по следния начин:

<променлива> ::= <идентификатор>

Някои идентификатори са резервирани в езика.

Запазени думи

Това са такива идентификатори, които се използват в програмите по стандартен, по предварително определен начин и които не могат да бъдат използвани по друг начин. Чрез тях се означават декларации, дефиниции, оператори, модификатори и други конструкции. Реализацията Visual C++ 6.0 съдържа около 70 такива думи.

В програмата Zad1.cpp са използвани запазените думи int, double, return.

Стандартни думи

Това са такива идентификатори, които се използват в програмите по стандартен, по предварително определен начин. Тези идентификатори **могат** да се използват и по други начини, например като обикновени идентификатори.

В програмата Zad1.cpp е използвана стандартната дума cout.

Например,

```
#include <iostream.h>
int main()
```

```
{int cout = 21;
 return 0;
}
```

е допустима програма на C++. В нея идентификаторът cout е използван като име на променлива. Правенето на опит за използване на cout по стандартния начин води до грешка. Така фрагментът

```
#include <iostream.h>
  int main()
  {int cout = 21;
   cout << cout << "\n";
   return 0;
   }
```

е недопустим.

Препоръка: Стандартните думи да се използват само по стандартния начин.

Константи

Информационна единица, която не може да бъде променяна, се нарича **константа**. Има числови, знакови, низови и др. типове константи.

Целите и реалните числа са **числови константи**. Целите числа се записват както в математиката и могат да бъдат задавани в десетична, шестнадесетична или осмична бройна система. Реалните числа се записват по два начина: във формат с *фиксирана точка* (например, 2.34 -12345.098) и в *експоненциален формат* (например, 5.23e-3 или 5.23E-3 означават 5.23 умножено с 10^{-3}).

Низ, **знаков низ** или **символен низ** е крайна редица от знаци, оградени в кавички. Например, редиците: "Това е низ.", "1+23-34", "Hello\n" са низове.

Забележка: Операторът

```
cout << "Hello\n";
```

извежда върху екрана поздрава Hello и премества курсора на нов ред.

Оператори

В C++ има три групи оператори: аритметично-логически, управляващи и оператори за управление на динамичната памет.

- *аритметично-логически оператори*

Реализират основните аритметични и логически операции като: събиране (+), изваждане (-), умножение (*), деление (/), логическо И (&&, and), логическо ИЛИ (||, or) и др. В програмата Zad1.cpp използвахме аритметичните оператори * и +.

- *управляващи оператори*

Това са конструкции, които управляват изчислителния процес. Такива са условният оператор, операторът за цикъл, за безусловен преход и др.

- *операторите за управление на динамичната памет*

Те позволяват по време на изпълнение на програмата да бъде заделяна и съответно освобождавана динамична памет.

Препинателни знаци

Използват се ; < > { } ( ) и др. знаци.

Разделяне на думите

В C++ разделителите на думите са интервалът, вертикалната и хоризонталната табулации и знакът за нов ред.

Коментари

Коментарите са текстове, които не се обработват от компилатора, а служат само като пояснения за програмистите. В C++ има два начина за означаване на коментари. Единият начин е, текстът да се огради с /* и */. Тези коментари не могат да бъдат влагани. Другият начин са коментарите, които започват с // и завършват с края на текущия ред.

Коментарите са допустими навсякъде, където е допустим разделител.

Забележка: Не се препоръчва използването на коментари от вида // в редовете на директивите на компилатора.

2.3. Вход и изход

Програма Zad1.cpp намира периметъра и лицето само на правоъгълник със страни 2.3 и 3.7. Нека решим тази задача в общия случай.

Задача 2. Да се напише програма, която въвежда размерите на правоъгълник и намира периметъра и лицето му.

Програмата Zad2.cpp решава задачата.

```
// Program Zad2.cpp
#include <iostream.h>
int main()
{// въвеждане на едната страна
  cout << "a= ";
  double a;
  cin >> a;
 // въвеждане на другата страна
 cout << "b= ";
 double b;
 cin >> b;
 // намиране на периметъра
 double p;
 p = 2*(a+b);
 // намиране на лицето
 double s;
 s = a*b;
  // извеждане на периметъра
 cout << "p= " << p << "\n";
  // извеждане на лицето
 cout << "s= " << s << "\n";
 return 0;
}
```

Когато програмата бъде стартирана, върху екрана ще се появи подсещането

```
a=
```

което е покана за въвеждане размерите на едната страна на правоъгълника. Курсорът стои след знака =. Очаква се да бъде въведено число (цяло или реално), след което да бъде натиснат клавишът ENTER.

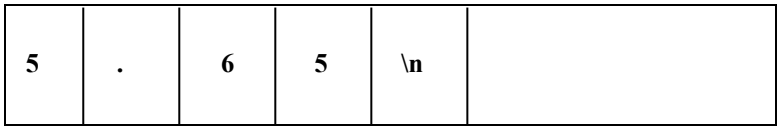
Следва покана за въвеждане на стойност за другата страна на правоъгълника, след което програмата ще изведе резултата и ще завърши изпълнението си.

Въвеждането на стойността на променливата a се осъществява с оператора за вход

```
cin >> a;
```

Обектът cin е името на стандартния входен поток, обикновено клавиатурата на компютъра. Изпълнението му води до пауза до въвеждане на число и натискане на клавиша ENTER. Нека за стойност на a е въведено 5.65, следвано от ENTER. В буфера на клавиатурата се записва

cin



След изпълнението на

```
cin >> a;
```

променливата a се свързва с 5.65, а в буфера на клавиатурата остава знакът \n, т.е.

cin

ОП

a

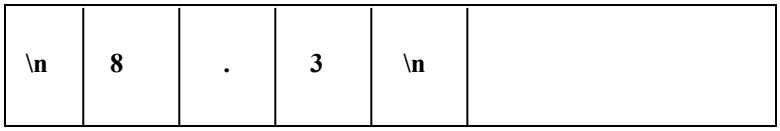
5.65

Въвеждането на стойността на променливата b се осъществява с оператора за вход

```
cin >> b;
```

Изпълнението му води до пауза до въвеждане на число и натискане на клавиша ENTER. Нека е въведено 8.3, следвано от ENTER. В буфера на клавиатурата имаме:

cin



Изпълнението на оператора

```
cin >> b;
```

прескача знака \n, свързва 8.3 с променливата b, а в буфера на клавиатурата отново остава знакът \n, т.е.

cin

```
ОП
a         b
5.65      8.3
```

Чрез оператора за вход могат да се въвеждат стойности на повече от една променлива. Фиг. 2.6 съдържа по-пълно негово описание.

Входът от клавиатурата е буфериран. Това означава, че всяка редица от натиснати клавиши се разглежда като пакет, който се обработва чак след като се натисне клавишът ENTER.

Оператор за вход >>

Синтаксис

cin >> <променлива>;

където

- cin е обект (променлива) от клас (тип) istream, свързан с клавиатурата,
- <променлива> е идентификатор, дефиниран, като променлива от "допустим тип", преди оператора за въвеждане. (Типовете int, long, double са допустими).

Семантика

Извлича (въвежда) от cin (клавиатурата) поредната дума и я прехвърля в аргумента-приемник <променлива>. Конструкцията

cin >> <променлива>

е израз от тип istream със стойност левия му аргумент, т.е. резултатът от изпълнението на оператора >> е cin. Това позволява няколко думи да бъдат извличани чрез верига от оператори >>.

Следователно, допустим е следният по-общ синтаксис на >>:

cin >> <променлива> {>> <променлива>}$_{опц}$;

Операторът >> се изпълнява отляво надясно. Такива оператори се наричат лявоасоциативни. Така операторът

cin >> променлива$_1$ >> променлива$_2$ >> … >> променлива$_n$;

е еквивалентен на редицата от оператори:

cin >> променлива$_1$;
cin >> променлива$_2$;
…
cin >> променлива$_n$;

Освен това, ако операцията въвеждане е завършила успешно, състоянието на cin е true, в противен случай състоянието на cin е false.

Пример:

```
cin >> a >> b;
```

Фиг. 2.6 Оператор за вход >>

В случая

cin >> променлива$_1$ >> променлива$_2$ >> … >> променлива$_n$;

от Фиг. 2.6, настъпва пауза. Компилаторът очаква да бъдат въведени n *стойности* - за променлива$_1$, променлива$_2$,…, променлива$_n$, съответно и да бъде натиснат клавишът ENTER. Тези стойности трябва да бъдат въведени по подходящ начин (на един ред, на отделни редове или по няколко данни на последователни редове, слепени или разделени с интервали, табулации или знака за нов ред), като *стойност$_i$* трябва да бъде от тип, съвместим с типа на *променлива$_i$* (i = 1, 2, …, n).

Пример: Да разгледаме програмния фрагмент:

```
double a, b, c;
cin >> a >> b >> c;
```

Операторът за вход изисква да бъдат въведени три реални числа за a, b и c съответно. Ако се въведат

```
1.1   2.2    3.3 ENTER
```

променливата a ще се свърже с 1.1, b – с 2.2 и c – с 3.3. Същият резултат ще се получи, ако се въведе

```
1.1   2.2 ENTER
3.3 ENTER
```

или

```
1.1 ENTER
2.2   3.3 ENTER
```

или

```
1.1 ENTER
2.2 ENTER
3.7 ENTER
```

или даже ако се въведе

```
1.1   2.2   3.3   4.4 ENTER
```

В последния случай, стойността 4.4 ще остане необработена в буфера на клавиатурата и ще обслужи следващо четене, ако има такова. Този

начин на действие съвсем не е приемлив. Още по-лошо ще стане когато се въведат данни от неподходящ тип.

Пример: Да разгледаме фрагмента:

```
int a;
cin >> a;
```

Той дефинира целочислена променлива a (a е променлива от тип int), след което настъпва пауза в очакване да бъде въведено цяло число. Нека сме въвели 1.25, следвано от ENTER. Състоянието на буфера на клавиатурата е:

cin

1	.	2	5	\n	

Операторът

```
cin >> a;
```

свързва a с 1, но не прескача останалата информация от буфера и тя ще обслужи следващо четене, което води до непредсказуем резултат. Още по-неприятна е ситуацията, когато вместо цяло число за стойност на a се въведе някакъв низ, например one, следван от ENTER. В този случай, изпълнението на

```
cin >> a;
```

ще доведе до

cin

o	n	e	\n	**състояние** fail

и

ОП

a

-

т.е. стойността на променливата a не се променя (остава неопределена), а буферът на клавиатурата изпада в състояние fail. За съжаление системата не извежда съобщение за грешка, което да уведоми за възникналия проблем.

Засега препоръчваме въвеждането на коректни входни данни. Преодоляването на недостатъците, илюстрирани по-горе, ще разгледаме в следващите части на книгата.

Вече използвахме оператора за изход. Фиг. 2.7 описва неговите синтаксис и семантика.

Оператор за изход <<

Синтаксис

cout << <израз>;

където

- cout е обект (променлива) от клас (тип) ostream, предварително е свързан с екрана на компютъра;
- <израз> е израз от допустим тип. Представата за израз продължава да бъде тази от математиката. Допустими типове са bool, int, short, long, double, float и др.

Семантика

Операторът << изпраща (извежда) към (върху) cout (екрана на компютъра) стойността на <израз>. Конструкцията

cout << <израз>

е израз от тип ostream и има за стойност първия му аргумент, т.е. резултатът от изпълнението на оператора << в горния случай е cout. Това позвалява чрез верига от оператори << да бъдат изведени стойностите на повече от един <израз>, т.е. допустим е следният по-общ синтаксис:

cout << <израз> {<< <израз>}$_{опц}$;

Операторът << се изпълнява отляво надясно (лявоасоциативен е). Така операторът

cout << израз$_1$ << израз$_2$ << … << израз$_n$;

е еквивалентен на редицата от оператори:

cout << израз$_1$;
cout << израз$_2$;
…
cout << израз$_n$;

Пример:

cout << “a= “ << a << “\n”;

Фиг. 2.7 Оператор за изход

2.4. Структура на програмата на C++

Когато програмата е малка, естествено е целият й код да бъде записан в един файл. Когато програмите са по-големи или когато се работи в колектив, ситуацията е по-различна. Налага се да се раздели кодът в отделни изходни (source) файлове. Причините, поради които се налага разделянето, са следните. Компилирането на файл отнема време и е глупаво да се чака компилаторът да превежда отново и отново код, който не е бил променян. Трябва да се компилират само файловете, които са били променяни.

Друга причина е работата в колектив. Би било трудно много програмисти да редактират едновременно един файл. Затова кодът на програмата се разделя така, че всеки програмист да отговаря за един или няколко файлове.

Ако програмата се състои от няколко файла, трябва да се каже на компилатора как да компилира и изгради цялата програма. Това ще направим в следващите раздели. Сега ще дадем най-обща представа за структурата на изходните файлове. Ще ги наричаме още модули.

Изходните файлове се организират по следния начин:

```
<изходен_файл> ::=   <заглавен_блок_с_коментари>
                     <заглавни_файлове>
                     <константи>
                     <класове>
                     <глобални_променливи>
                     <функции>
```

Заглавен блок с коментари

Всеки модул започва със заглавен блок с коментари, даващи информация за целта му, за използваните компилатор и операционна среда, за името на програмиста и датата на създаването на модула. Заглавният блок с коментари може да съдържа забележки, свързани с описания на структури от данни, аргументи, формат на файлове, правила, уговорки и друга информация.

Заглавни файлове

В тази част на модула са изброени всички необходими заглавни файлове. Например

```
#include <iostream.h>
```

#include <cmath.h>

Забелязваме, че за разделител е използван знакът за нов ред, а не ;.

Константи

В тази част се описват константите, необходими за модула. Вече имаме някаква минимална представа за тях. По-подробно описание на синтаксиса и семантиката им е дадена на Фиг. 2.8. За да бъде програмата по-лесна за четене и модифициране, е полезно да се дават символични имена не само на променливите, а и на константите. Това става чрез дефинирането на константи.

Задача 3. Да се напише програма, която въвежда радиуса на окръжност и намира и извежда дължината на окръжността и лицето на кръга с дадения радиус.

Една програма, която решава задачата е следната:

```
// Program Zad3.cpp
#include <iostream.h>
const double PI = 3.14159265;
int main()
{ double r;
  cout << “r= “;
  cin >> r;
  double p = 2 * PI * r;
  double s = PI * r * r;
  cout << “p=” << p << “\n”;
  cout << “s=” << s << “\n”;
  return 0;
}
```

В тази програма е дефинирана реална константа с име PI и стойност 3. 14159265, след което е използвано името PI.

Дефиниране на константи

Синтаксис

const <име_на_тип> <име_на_константа> = <израз>;

където

const е запазена дума (съкращение от constant);
<име_на_тип> е идентификатор, означаващ име на тип;
<име_на_константа> е идентификатор, обикновено състоящ се от главни букви, за да се различава визуално от променливите.
<израз> е израз от тип, съвместим с <име_на_тип>.
Семантика
Свързва <име_на_константа> със стойността на <израз>. Правенето на опит да бъде променяна стойността на константата предизвиква грешка.
Примери:

```
const int MAXINT = 32767;
const double RI = 2.5 * MAXINT;
```

Фиг. 2.8 Дефиниране на константи

Предимства на дефинирането на константите:

- Програмите стават по-ясни и четливи.
- Лесно (само на едно място) се променят стойностите им (ако се налага).
- Вероятността за грешки, възможни при многократното изписване на стойността на константата, намалява.

Забележка: Тъй като в програмата Zad3.cpp е използвана само една функция (main), дефиницията на константата PI може да се постави във функцията main, преди първото нейно използване.

Класове

Тази част съдържа дефинициите на класовете, използвани в модула.

В езика C++ има стандартен набор от типове данни като int, double, float, char и др. Този набор може да бъде разширен чрез дефинирането на класове.

Дефинирането на клас въвежда нов тип, който може да бъде интегриран в езика. Класовете са в основата на обектно-ориетираното програмиране, за което е предназначен езикът C++.

Дефинирането и използването на класове ще бъде разгледано във втората част на курса.

Глобални променливи

Езикът поддържа глобални променливи. Те са променливи, които се дефинират извън функциите и които са “видими” за всички функции, дефинирани след тях. Дефинират се както се дефинират другите (локалните) променливи. Използването на много глобални променливи е лош стил за програмиране и не се препоръчва. Всяка глобална променлива трябва да е съпроводена с коментар, обясняващ предназначението й.

Функции

Всеки модул задължително съдържа функция main. Възможно е да съдържа и други функции. Тогава те се изброяват в тази част на модула. Ако функциите са подредени така, че всяка от тях е дефинирана преди да бъде извикана, тогава main трябва да бъде последна. В противен случай, в началото на тази част на модула, трябва да се **декларират** всички функции.

Задачи

Задача 1. Кои от следните редици от знаци са идентификатори, кои не и защо?

а) a	б) x1	в) x_1	г) x’	д) x1x2
е) sin	ж) sin x	з) cos(x)	и) x-1	к) 2a
л) min 1	м) Beta	н) a1+a2	о) k”m	п) sin’x

Задача 2. Намерете синтактичните грешки в следващата програма:

```
include <iostream>
int Main()
{ cout >> “a, b =  “;
  cin << a, b;
  cout << “The product of “ << a << “and” << b << “is: “
       << a*b < “\n”
  return 0;
}
```

Задача 3. Напишете програма, която разменя стойностите на две числови променливи.

Задача 4. Напишете програма, която намира минималното (максималното) от две цели числа.

Задача 5. Напишете програма, която изписва с главни букви текста Hellow World.

Упътване: Голямата буква Н може да се представи така:

```
o   o
o   o
o   o
ooooo
o   o
o   o
o   o
```

и да се реализира по следния начин:

char* let_H = “o o\no o\no o\nooooo\no o\no o\no o\n”;

където char* означава тип низ.

Допълнителна литература

1. Г. Симов, Програмиране на C++, София, СИМ, 1993.
2. К. Хорстман, Принципи на програмирането със C++, София, СОФТЕХ, 2000.
3. П. Лукас, Наръчник на програмиста, София, Техника, 1994.