9

Функции от по-висок ред

Функция, някои формални параметри на която са функции, се нарича **функция от по-висок ред.**

В езика C++ е възможно формален параметър на функция да е указател към функция, а също е възможно резултатът от изпълнението на функция да е указател към функция. Това позволява да се реализират функции от по-висок ред, а също и такива, които връщат функция.

9.1 Указател към функция

Името на функция е константен указател, сочещ към първата машинна инструкция от изпълнимия й машинен код. В езика C++ е възможно да се дефинират променливи, които са указатели към функции (Фиг. 9.1).

Дефиниция на указател към функция

<дефиниция_на_променлива_указател_към_функция> ::=
<тип_на_функция>(*<указател_към_функция>)(<формални_параметри>)
[= <име_на_функция>]$_{опц}$;

където

- <указател_към_функция> е идентификатор;
- <име_на_функция> е идентификатор, означаващ име на функция от тип <тип_на_функция> и параметри - <формални_параметри>;
- <тип_на_функция> и <формални_параметри> се дефинират аналогично на съответните от заглавието на дефиниция функция. Имената на параметрите могат да се пропуснат.

Фиг. 9.1 Дефиниция на указател към функция

Забележка: Скобите, ограждащи *<указател_към_функция>, са задължителни. В противен случай дефиницията ще се изтълкува от компилатора като декларация на функция с име < указател_към_функция>, с параметри - <формални_параметри> и тип – указател към <тип_на_функция>.

В резултат на дефиницията на променлива от тип указател към функция, за променливата се отделят 4В ОП, която е с неопределена стойност, ако дефиницията е без инициализация, и съдържа адреса на първата машинна команда от изпълнимия код на функцията, чрез която е направена инициализацията, ако дефиницията е с инициализация.

Примери:

1. double (*p)(double, double);

е дефиниция на променлива p от тип указател към функция от тип double с два аргумента също от тип double. В резултат за p се отделят 4В ОП, които са с неопределена стойност.

2. int (*q)(int, int*);

дефинира променлива q от тип указател към функция от тип int, с два аргумента, единият от които цял, а другият – указател към int. За q се отделят 4В ОП, които са с неопределена стойност.

Нека са дефинирани следните функции за сортиране на числови редици:

```
void bubblesort(int, int*);   // метод на мехурчето
void mergesort(int, int*);    // сортиране чрез сливане
void heapsort(int, int*);   // пирамидално сортиране
```

Променливата r може да е указател към тези функции ако е дефинирана по следния начин:

```
void (*r)(int, int*);
```

r не може да е указател към функциите:

```
int f1(int, int*);
int f2(int, int*);
```

Указател към последните може да е променливата s, където:

```
int (*s)(int, int*);
```

Горните дефиниции на p, q, r и s са без инициализации.

Дефинициите на променливите x и y

```
void (*x)(int, int*) = bubbesort;
int (*y)(int, int*) = f2;
```

са с инициализация. За всяка от тях се отделят 4В ОП, в която памет се записват адресите на първите команди на изпълнимите кодове на bubblesort и f2 съответно.

На променлива от тип указател към функция може да се присвои името на функция от същия тип. Присвояването се извършва по общоприетия начин.

Пример: Допустими са присвояванията:

```
r = mergesort;
x = heapsort;
```

Обръщението към функция освен директно може да се осъществява и индиректно – чрез указател към нея. След инициализация на променлива от тип указател към функция, чрез променливата може да се осъществи обръщение към конкретна функция. Така се предоставя ефективен способ за предаване на управлението към потребителски и библиотечни функции.

Пример:

```
void (*r)(int, int*) = bubblesort;
bubblesort(n, a);     // директно обръщение
(*r)(n, a);           // индиректно обръщение (чрез r).
```

Забележка: Някой компилатори, в това число и на Visual C++ 6.0, допускат извикването на функция чрез указател да се осъществява и само чрез името на указателя.

Пример: Ако

```
void (*r)(int, int*) = bubblesort;
```

Обръщението към bubblesort

```
(*r)(n, a);
```

може да се запише и по следния начин: `r(n, a);`

9.2 Функциите като формални параметри

Указател към функция може да е формален параметър на функция. Ще илюстрираме тази възможност с няколко примери.

Задача 80. Да се напише функция, която реализира математическата абстракция:

$$\sum_{\substack{i=a \\ i->next(i)}}^{b} f(i).$$

където a и b са дадени реални числа (a ≤ b), f е реална едноаргументна функция, задаваща терма, а next е реална едноаргументна функция, задаваща стъпката за промяна на управляващия параметър на сумата.

Преди да решим задачата в общияй вид ще предложим няколко частни решения.

а) Да се дефинира функция, която намира стойността на сумата:

sin(a) + sin(a+1) + sin(a+2) + ... + sin(b),

където a и b са дадени реални числа.

Функцията sum_sin намира тази сума.

```
double sum_sin(double a, double b)
{double s = 0;
 for (double i = a; i <= b + 1E-14; i = i + 1)
    s = s + sin(i);
 return s;
}
```

б) Да се дефинира функция, която намира стойността на сумата:

cos(a) + cos(a + 0.2) + cos(a + 0.4) + ... + sin(b)

където a и b са дадени реални числа.

Функцията sum_cos намира тази сума.

```
double sum_cos(double a, double b)
{double s = 0;
 for (double i = a; i <= b + 1e-14; i = i + 0.2)
     s = s + cos(i);
 return s;
}
```

Забелязваме, че тези две функции се “приличат”. Написани са по следния общ шаблон:

```
double <name>(double a, double b)
{double s = 0;
 for (double i = a; i <= b + 1e-14; i = <next>(i))
    s = s + <f>(i);
 return s;
}
```

Елементите, по които функциите sum_sin и sum_cos се различават са означени с <...> в шаблона. Това са две функции: f, означаваща терма и next - стъпката на сумата. Като използваме възможността формален параметър на функция да е указател към функция, можем да изнесем <f> и <next> като формални параметри на функцията и да обобщим тези частни случаи. Така стигаме до функцията sum:

```
// Function Zad80
double sum(double a, double b, double (*f)(double),
           double (*next)(double))
{double s = 0;
 for (double i = a; i <= b + 1e-14; i = next(i))
    s = s + f(i);
 return s;
}
```

Обръщенията към sum:

```
sum(a, b, sin, next1)
sum(a, b, cos, next2)
```

където

```
int next1(double x)
{return x + 1;
}
int next2(double x)
{return x + 0.2;
}
```

реализират горните частни случаи.

sum е функция от по-висок ред. В нея третият и четвъртият параметри са указатели към функции.

Като използваме sum, може да дефинираме sum_sin и sum_cos по следния начин:

```
double sum_sin(double a, double b)
{return sum(a, b, sin, next1);
}
double sum_cos(double a, double b)
{return sum(a, b, cos, next2);
}
```

Задача 81. Да се напише функция, която реализира математическата абстракция:

$$\prod_{\substack{i=a \\ i->next(i)}}^{b} f(i)$$

където a и b са реални числа, f е реална едноаргументна функция, задаваща терма, а next - реална едноаргументна функция, задаваща стъпката за промяна на управляващия параметър на произведението. Да се включи тази функция в програма и се намерят:

```
tg(1) * tg(1.5) * tg(2) * tg(2.5) * tg(3)
```

и

```
arctg(1) * arctg(1.1) * arctg(1.2) * arctg(1.3).
```

Програма Zad81.cpp решава задачата. Функцията prod от нея се реализира чрез последователно преминаване през стъпки, аналигични на тези от задача 80.

```
// Program Zad81.cpp
#include <iostream.h>
#include <math.h>
double prod(double, double, double (*)(double),
            double (*) (double));
double next1(double);
double next2(double);
int main()
{cout << prod(1, 3, tan, next1) << '\n';
 cout << prod(1, 1.3, atan, next2) << '\n';
 return 0;
}
double prod(double a, double b, double (*f)(double),
            double (*next)(double))
{double s = 1.0;
 for (double i = a; i <= b + 1e-14; i = next(i))
    s = s * f(i);
 return s;
}
double next1(double x)
```

```
{return x + 0.5;
}
double next2(double x)
{return x + 0.1;
}
```

В тази програма е дефинирана функцията от по-висок ред prod, реализираща исканата абстракция. В нея третият и четвъртият параметри са указатели към функции. В главната програма са направени две обръщения към нея

```
prod(1, 3, tan, next1)
```

и

```
prod(1, 1.3, atan, next2),
```

намиращи търсените произведения.

Забелязваме, че функциите sum и prod си "приличат". Написани са по следния общ шаблон.

```
double <name>(double a, double b, double (*f)(double),
              double (*next)(double))
{double s = <null_val>;
 for (double i = a; i <= b + 1e-14; i = next(i))
    s = s <op> f(i);
 return s;
}
```

И в този случай, елементите, по които sum и prod се различават са оградени с <...>. Това са операцията op и нулата на операцията - null_val. Отново ще изнесем op и null_val като формални параметри на функцията. Тъй като op е бинарна инфиксна операция, а не име на функция, ще дефинираме помощна реална функция с име op, с два реални параметъра и връщаща резултата от прилагането на операцията op към аргументите на функцията op. Така получаваме още едно обобщение на горните абстракции - функцията от по-висок ред accumulate (задача 82).

Задача 82. Да се напише програма, която реализира следната математическа абстракция:

$$f(a) \otimes f(next(a)) \otimes f(next(next(a)) \otimes \ldots \otimes f(b)$$

където с ⊗ е означена произволна бинарна целочислена операция, а f и next имат смисъла, определен в предходните две задачи.

Програма Zad82.cpp решава задачата.

```
// Program Zad82.cpp
#include <iostream.h>
#include <math.h>
double accumulate(double (*) (double, double),
                  double, double, double,
                  double (*)(double), double (*) (double));
double plus(double, double);
double mult(double, double);
double next1(double);
double next2(double);
int main()
{cout << "a, b= ";
 double a, b;
 cin >> a >> b;
 if (!cin)
 {cout << "Error! \n";
  return 1;
 }
 cout << accumulate(plus, 0, a, b, cos, next1) << '\n';
 cout << accumulate(mult, 1, a, b, sin, next2) << '\n';
 return 0;
}

double accumulate(double (*op)(double, double),
                  double null_val, double a, double b,
                  double (*f)(double), double (*next)(double))
{double s = null_val;
 for (double i = a; i <= b +1e-14; i = next(i))
    s = op(s, f(i));
 return s;
}
```

```
double next1(double x)
{return x + 1;
}
double next2(double x)
{return x + 2;
}
double plus(double x, double y)
{return x + y;
}
double mult(double x, double y)
{return x * y;
}
```

Функцията accumulate е функция от по-висок ред. Нейните първи, пети и шести формални параметри са указатели към функции, задаващи съответно операцията op, терма f и стъпката next.

В тази програма са направени две обръщения към функцията accumulate, които намират:

```
cos(a) + cos(a+1) + cos(a+2) + ... + cos(b)
```

и

```
sin(a) * sin(a+2) *  sin(a+4) * ... * sin(b)
```

съответно.

Чрез accumulate, функциите sum и prod могат да се дефинират по следния начин:

```
double sum(double a, double b, double (*f)(double),
           double (*next)(double))
{return accumulate(plus, 0, a, b, f, next);
}
double prod(double a, double b, double (*f)(double),
            double (*next)(double))
{return accumulate(mult, 1, a, b, f, next);
}
```

където plus и mult са дефинирани в програма Zad82.cpp.

Използването на променливи, които са указатели към функции, усложнява записа на дефиницията на функция. Добре би било да дадем имена на типовете указател към функция и вместо дефиницията да използваме името на типа. Задаването на имена на типове може да се

осъществи чрез оператора typedef. На Фиг. 9.2 са дадени синтаксисът и семантиката на този оператор.

Оператор typedef

Синтаксис

typedef <тип> <име>;

където

- <тип> е дефиниция на тип;
- <име> е идентификатор, определящ името на новия тип.

Семантика

Определя <име> за синоним на типа от <тип>.

Фиг. 9.2 Оператор typedef

Примери:

```
typedef unsigned char BYTE; // BYTE е синоним на unsigned char
typedef double REAL;  // REAL е синоним на double
```

Задаването на алтернативно име на тип указател към функция чрез typedef се осъществява по аналогичен начин на дефиниране на променлива от тип указател към функция като новото име на типа заема мястото на променливата.

Примери:

1. typedef double(*mytype)(double);

определя mytype като синоним на типа double (*)(double);

2. typedef double(*newtype)(double, double);

определя newtype като синоним на типа double (*)(double, double).

Като използваме оператора typedef и дефинираме:

```
typedef double (*type1) (double, double);
typedef double (*type2) (double);
```

декларацията на функцията accumulate от Zad82.cpp:

```
double accumulate(double (*) (double, double),
                  double, double, double,
                  double (*)(double), double (*) (double));
```

може да се запише по следния начин:

```
double accumulate(type1, double, double, double, type2, type2);
```

Задача 83. Като се направи подходяща модификация на accumulate, да се напише програма, която по дадени естествено число n и реално число x, намира сумата:

$$\sum_{i=0}^{n} \frac{x^i}{i!}$$

Програма Zad83.cpp решава задачата. В нея a и b са 0 и n съответно. Термът f е:

$$f: i \longrightarrow \frac{x^i}{i!},$$

а стъпката се задава от:

$$next: i \longrightarrow i + 1.$$

Направена е модификация на функцията accumulate. Последната се налага заради промяната на типовете на a, b, на f и next.

```
// Program Zad83.cpp
#include <iostream.h>
#include <math.h>
typedef double (*type1) (double, double);
typedef double (*type2)(int);
typedef int (*type3) (int);
double x;
double accumulate(type1, double, int, int, type2, type3);
double f(int);
int next(int);
double sum(double, double);
int main()
{cout << "n= ";
 int n;
 cin >> n;
 if (!cin || n < 0)
 {cout << "Error! \n";
  return 1;
 }
 cout << "x= ";
```

```
 cin >> x;
 if (!cin)
 {cout << "Error! \n";
  return 1;
 }
 cout << accumulate(sum, 0, 0, n, f, next) << '\n';
 return 0;
}
double f(int i)
{int p = 1;
 for (int j = 1; j <= i; j++) p = p*j;
  return pow(x, i)/p;
}
int next(int x)
{return x + 1;
}
double sum(double x, double y)
{return x + y;
}
double accumulate(type1 op, double null_val,
                  int a, int b, type2 f, type3 next)
{double s = null_val;
 for (int i = a; i <= b; i = next(i))
    s = op(s, f(i));
 return s;
}
```

9.3 Функциите като върнати оценки

Функция може да върне като резултат указател към друга функция. Например, декларацията

```
int (*fun(int, int))(int*, int);
```

определя функцията fun с два цели аргумента и връщаща указател към функция от тип

```
int (*)(int*, int).
```

Ако зададем име на този тип чрез typedef, т.е.

```
   typedef int (*fun-point)(int*, int);
```
този запис може да се опрости до:
```
   fun-point fun(int, int);
```

Задача 84. Да се напише програма, която по зададено реално число x и символ (a, b, c или d) избира за изпълнение функция, определена чрез зависимостта:

$$y = \begin{cases} \sin(x) & \longrightarrow a \\ \cos(x) & \longrightarrow b \\ \exp(x) & \longrightarrow c \\ \log(x) & \longrightarrow d. \end{cases}$$

Програма Zad84.cpp решава задачата.

```
// Program Zad84.cpp
#include <iostream.h>
#include <math.h>
typedef double (*f_type)(double);
f_type table(char ch)
{switch(ch)
 {case 'a': return sin; break;
  case 'b': return cos; break;
  case 'c': return exp; break;
  case 'd': return log; break;
  default: cout << "Error! \n"; return tan;
 }
}
int main()
{char ch;
 cout << "ch= ";
 cin >> ch;
 if (ch < 'a' || ch >'d') cout << "Incorrect input! \n";
 else
 {double x;
  cout << "x= ";
  cin >> x;
```

```
    cout << table(ch)(x) << '\n';
   }
   return 0;
  }
```

Илюстрираните в тази част възможности на езика C++ показват, че данните от тип функции съществено не се отличават от другите видове данни. Това показва високата степен на унифицираност в езика и води до увеличаване на изразителната му сила.

Задачи

Задача 1. Като използвате функциите от по-висок ред sum и prod, намерете:

$$S = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{3!} + \ldots + (-1)^n \frac{x^{(2n+1)}}{(2n+1)!}$$

$$S = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$$

Задача 2. Като използвате функцията от по-висок ред prod, намерете:

а) x^n, където x е дадено реално, а n – дадено естествено число.

б) n!, където n е дадено естествено число.

в) броят на вариациите от n елемента от k-ти клас (n и k са дадени естествени числа, $0 \le k \le n$).

г) броят на комбинациите от n елемента от k-ти клас (n и k са дадени естествени числа, $0 \le k \le n$).

Допълнителна литература

1. Ст. Липман, Езикът C++ в примери, “КОЛХИДА ТРЕЙД” КООП, София, 1993.

2. Д. Луис, C/C++ бърз справочник, ИНФОДАР, София, 1998.

3. М. Тодорова, Езици за функционално и логическо програмиране. Функционално програмиране, СОФТЕХ, София, 1998.

4. B. Stroustrup, C++ Programming Language. Third Edition, Addison - Wesley, 1997.