Лабораторная работа № 2

Итерации и обработка исключений

В программировании под *итерацией* понимают такую организацию обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя. В работе рассматриваются две группы итерационных методов, нашедших широкое распространение на практике: приближенное решение алгебраических и трансцендентных уравнений и приближенное интегрирование функций. При реализации итерационных методов могут возникать исключительные ситуации, которые приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма. С этой целью в работе рассматривается *обработка исключений*, применяемая в языке С.

Замечание. Обработка исключений, встроенная в язык С++ и предназначенная для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы, в работе не рассматривается.

Теоретическая часть

1. Приближенное решение алгебраических и трансцендентных уравнений

Для сложного алгебраического или трансцендентного уравнения его корни редко удается найти точно. В некоторых случаях уравнение содержит коэффициенты, известные лишь приблизительно, и, следовательно, задача о точном определении корней уравнения теряет смысл. Во всех этих случаях используют способы приближенного нахождения корней уравнения и оценки степени их точности. В работе рассматриваются три приближенного нахождения корней уравнения: *метод половинного деления*, *метод хорд* и *метод касательных*.

Пусть дано уравнение

$$f(x) = 0, \qquad (1)$$

где функция f(x) определена и непрерывна в некотором конечном или бесконечном интервале $a < x < b$.
В некоторых случаях требуется существование и непрерывность первой производной f'(x) или даже второй производной f''(x).

Всякое значение ξ, обращающее функцию f(x) в нуль, т. е. такое, что

$$f(\xi) = 0,$$

называется *корнем уравнения* (1) или *нулем* функции f(x).

Будем предполагать, что уравнение (1) имеет лишь *изолированные корни*, т. е, для каждого корня уравнения (1) существует окрестность, не содержащая других корней этого уравнения.

Приближенное нахождение изолированных действительных корней уравнения (1) обычно складывается из двух этапов:

1) отделение корней, т. е. установление возможно тесных промежутков $[\alpha, \beta]$, в которых содержится один и только один корень уравнения (1);
2) уточнение приближенных корней, т. е. доведение их до заданной степени точности.

1.1. Отделение корней

Для отделения корней полезна известная теорема из математического анализа.

Теорема. Если непрерывная функция f(x) принимает значения разных знаков на концах отрезка $[\alpha, \beta]$, т. е. $f(\alpha)f(\beta) < 0$, то внутри этого отрезка содержится по меньшей мере один корень уравнения $f(x) = 0$, т. е. найдется хотя бы одно число $\xi \in (\alpha, \beta)$ такое, что $f(\xi) = 0$. □

Корень ξ заведомо будет единственным, если производная f'(x) существует и сохраняет постоянный знак внутри интервала (α, β), т. е. если $f'(x) > 0$ (или $f'(x) < 0$) при $\alpha < x < \beta$.

Процесс отделения корней начинается с установления знаков функции f(x) в граничных точках x = a и x = b области ее существования. Затем определяются знаки функции f(x) в ряде промежуточных точек $x = \alpha_1, \alpha_2, ...$, выбор которых учитывает особенности функции f(x). Если окажется, что $f(\alpha_k)f(\alpha_{k+1}) < 0$, то в силу теоремы 1 в интервале (α_k, α_{k+1}) имеется корень уравнения f(x) = 0. Нужно тем или иным способом убедиться, является ли этот корень единственным. Для отделения корней практически часто бывает достаточно провести процесс половинного деления, приближенно деля данный интервал (α, β) на две, четыре, восемь и т. д. равных частей (до некоторого шага) и определяя знаки функции f(x) в точках делений. Полезно помнить, что алгебраическое уравнение n-й степени

$$a_0x^n + a_1x^{n-1} + ... + a_n = 0 \qquad (a_0 \neq 0)$$

имеет не более n действительных корней. Поэтому если для такого уравнения мы получили n+1 перемену знаков, то все корни его отделены.

Пример 1.Отделить корни уравнения

$$f(x) = x^4 – 4x – 1 = 0.$$

Решение. Здесь $f'(x) = 4(x^3 – 1)$, и поэтому $f'(x) = 0$ при $x = 1$. Имеем $f(–\infty) > 0$ (+); $f(1) < 0$ (–); $f(+\infty) > 0$ (+). Следовательно, уравнение имеет только два действительных корня, из которых один лежит в интервале $(–\infty, 1)$, а другой – в интервале $(1, +\infty)$. □

1.2. Метод половинного деления

Пусть дано уравнение

$$f(x) = 0, \qquad (1)$$

где функция f(x) непрерывна на [a, b] и $f(a)f(b) < 0$.

Для нахождения корня уравнения (1), принадлежащего отрезку [a, b], делим этот отрезок пополам. Если $f((a+b)/2) = 0$, то $\xi = (a+b)/2$ является корнем уравнения. Если $f((a+b)/2) \neq 0$, то выбираем ту из половин $[a, (a+b)/2]$ или $[(a+b)/2, b]$, на концах которой функция f(x) имеет противоположные знаки. Новый суженный отрезок $[a_1, b_1]$ снова делим пополам и проводим то же рассмотрение и т. д. В результате получаем на каком-то этапе или точный корень уравнения (1), или же бесконечную последовательность вложенных друг в друга отрезков $[a_1, b_1]$, $[a_2, b_2]$, ..., $[a_n, b_n]$, ... таких, что

$$f(a_n)f(b_n) < 0 \ (n = 1, 2, ...) \qquad (2)$$

и

$$b_n – a_n = (b – a)/2^n. \qquad (3)$$

Так как левые концы $a_1, a_2, ..., a_n, ...$ образуют монотонную неубывающую ограниченную последовательность, а правые концы $b_1, b_2, ..., b_n, ...$ – монотонную невозрастающую ограниченную последовательность, то в силу равенства (3) существует общий предел

$$\xi = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$$

Переходя к пределу при $n\to\infty$ в неравенстве (2), в силу непрерывности функции f(x) получим $[f(\xi)]^2 \le 0$. Отсюда $f(\xi) = 0$, т. е. ξ является корнем уравнения (1), причем, очевидно,

$$0 \le \xi - a_n \le (b - a)/2^n. \qquad (4)$$

Если корни уравнения (1) не отделены на отрезке [a, b], то таким способом можно найти один из корней уравнения (1).

Метод половинного деления практически удобно применять для грубого нахождения корня данного уравнения, так как при увеличении точности значительно возрастает объем вычислительной работы.

Пример 2. Методом половинного деления уточнить корень уравнения

$$f(x) = x^4 + 2x^3 - x - 1 = 0,$$

лежащий на отрезке [0, 1].

Решение. Последовательно имеем:

f(0) = –1; f(1) = 1;
f(0,5) = 0,06 + 0,25 – 0,5 – 1 = –1,19;
f(0,75) = 0,32 + 0,84 – 0,75 – 1 = –0,59;
f(0,875) = 0,59 + 1,34 – 0,88 – 1 = +0,05;
f(0,8125) = 0,436 + 1,072 – 0,812 – 1 = –0,304;
f(0,8438) = 0,507 + 1,202 – 0,844 – 1 = –0,135;
f(0,8594) = 0,546 + 1,270 – 0,859 – 1 = –0,043 и т. д.

Можно принять

$\xi = (0{,}859 + 0{,}875)/2 = 0{,}867$. □

1.3. Способ пропорциональных частей (метод хорд)

Пусть для определенности $f(a) < 0$ и $f(b) > 0$. Тогда, вместо того чтобы делить отрезок [a, b] пополам, более естественно разделить его в отношении – f(a):f(b). Это дает нам приближенное значение корня

$$x_1 = a + h_1, \qquad (1)$$

где

$$h_1 = -f(a)/(-f(a) + f(b))*(b - a) = -f(a)/(f(b) - f(a))*(b - a). \qquad (2)$$

Далее, применяя этот прием к тому из отрезков $[a, x_1]$ или $[x_1, b]$, на концах которого функция f(x) имеет противоположные знаки, получим второе приближение корня x_2 и т. д.

Геометрически способ пропорциональных частей эквивалентен замене кривой y = f(x) хордой, проходящей через точки A[a, f(a)] и B[b, f(b)]. В самом деле, уравнение хорды AB есть

$$(x - a)/(b - a) = (y - f(a))/(f(b) - f(a)).$$

Отсюда, полагая x = x1 и y = 0, получим

$$x_1 = a - f(a)/(f(b) - f(a))*(b - a) \qquad (1')$$

Формула (1') полностью эквивалентна формулам (1) и (2).

Пример 3. Найти положительный корень уравнения

$$f(x) = x^3 – 0,2x^2 – 0,2x – 1,2 = 0$$

с точностью до 0,002.

Решение. Прежде всего отделяем корень. Так как

$$f(1) = –0,6 < 0 \text{ и } f(2) = 5,6 > 0,$$

то искомый корень ξ лежит в интервале (1, 2). Полученный интервал велик, поэтому разделим его пополам. Так как

$$f(1,5) = 1,425, \text{ то } 1 < \xi < 1,5.$$

последовательно применяя формулы (1) и (2), будем иметь:

$$x_1 = 1 + 0,6/(1,425 + 0,6)*(1,5 – 1) = 1 + 0,15 = 1,15;$$

$$f(x1) = –0,173;$$

$$x_2 = 1,15 + 0,173/(1,425 + 0,073)*(1,5 – 1,15) =1,15 + 0,040 = 1,190;$$

$$f(x2) = –0,036;$$

$$x_3 = 1,190 + 0,036/* (1,5 – 1,190) = 1,190 + 0,008 = 1,198;$$

$$f(x_3) = –0,0072.$$

Так как $f'(x) = 3x2 – 0,4x – 0,2$ и при $x_3 < x < 1,5$ имеем

$$f'(x) \geq 3*1,1982 – 0,4*1,5 – 0,2 = 3*1,43 – 0,8 = 3,49,$$

то можно принять:

$$0 < \xi – x_3 < 0,0072/3,49 \approx 0,002.$$

Таким образом, $\xi = 1,198 + 0,002\theta$, где $0 < \theta \leq 1$.

Заметим, что точный корень уравнения есть $\xi = 1,2$. □

1.4. Метод Ньютона (метод касательных)

Пусть корень ξ уравнения

$$f(x) = 0 \qquad (1)$$

отделен на отрезке [a, b], причем $f'(x)$ и $f''(x)$ непрерывны и сохраняют определенные знаки при $a \leq x \leq b$. Найдя какое-нибудь n-е приближенное значение корня $x_n \approx \xi$ ($a \leq x_n \leq b$), мы можем уточнить его по методу ньютона следующим образом. Положим

$$\xi = x_n + h_n \qquad (2)$$

где h_n считаем малой величиной. Отсюда, применяя формулу Тейлора, получим:

$$0 = f(x_n + h_n) \approx f(x_n) + h_n f'(x_n).$$

Следовательно,

$$h_n = - f(x_n)/f'(x_n).$$

Внеся эту поправку в формулу (2), найдем следующее (по порядку) приближение корня .

$$x_{n+1} = x_n - f(x_n)/f'(x_n) \ (n = 0, 1, 2, \ldots). \qquad (3)$$

Геометрически метод Ньютона эквивалентен замене небольшой дуги кривой y = f(x) касательной, проведенной в некоторой точке кривой.

В общем случае «хорошим» начальным приближением x_0 является то, для которого выполнено неравенство

$$f(x_0)f''(x_0) > 0. \qquad (4)$$

Это доказывается с помощью следующей теоремы.

Теорема. Если f(a)f(b) < 0, причем f'(x) и f''(x) отличны от нуля и сохраняют определенные знаки при $a \le x \le b$, то, исходя из начального приближения $x_0 \in [a, b]$, удовлетворяющего неравенству (4), можно вычислить методом Ньютона (формула (3)) единственный корень ξ уравнения (1) с любой степенью точности. □

Замечание 1. Если: 1) функция f(x) определена и непрерывна при $-\infty < x < +\infty$; 2) f(a)f(b) < 0; 3) $f'(x) \ne 0$ при $a \le x \le b$; 4) f''(x) существует всюду и сохраняет постоянный знак, то при применении метода Ньютона для нахождения корня уравнения f(x) = 0, лежащего в интервале (a, b), за начальное приближение x_0 можно принять любое значение $c \in [a, b]$. В частности, можно взять $x_0 = a$ или $x_0 = b$.

Замечание 2. Из формулы (3) видно, что чем больше численное значение производной f'(x) в окрестности данного корня, тем меньше поправка, которую нужно прибавить к n-му приближению, чтобы получить (n+1)-е приближение. Поэтому метод Ньютона особенно, удобно применять тогда, когда в окрестности данного корня график функции имеет большую крутизну. Но если численное значение производной f'(x) близ корня мало, то поправки будут велики, и вычисление корня по этому методу может оказаться очень долгим, а иногда и вовсе невозможным. Следовательно, если, кривая y = f(x) вблизи точки пересечения с осью Ox почти горизонтальна, то применять метод Ньютона для решения уравнения f(x) = 0 не рекомендуется.

Пример 4. Вычислить методом Ньютона отрицательный корень уравнения $f(x) = x^4 - 3x^2 + 75x - 10000 = 0$ с пятью верными знаками.

Решение. Полагая в левой части уравнения x = 0, –10, –100, ... , получим f(0) = –10000, f(–10) = –1050, $f(-100) \approx +10^8$. Следовательно, искомый корень ξ находится в интервале $-100 < \xi < -10$. Сузим найденный интервал. Так как f(–11) = 3453, то $-11 < \xi < -10$. В этом последнем интервале f'(x) < 0 и f''(x) > 0. Так как f(–11) > 0 и f''(–11) > 0, то можем принять за начальное приближение $x_0 = -11$. Последовательные приближения x_n (n = 1, 2, ...) вычисляем по следующей схеме:

n	x_n	$f(x_n)$	$f'(x_n)$	$h_n = -f(x_n)/f''(x_n)$
0	–11	3453	–5183	0,7
1	–10,3	134,3	–4234	0,03
2	–10,27	37,8	–4196	0,009
3	–10,261	0,2	–	–

Останавливаясь на n = 3, провернем знак значения $f(x_n+0{,}001) = f(-10{,}260)$. Так как f(–10,260) < 0, то $-10{,}261 < \xi < -10{,}260$, и любое из этих чисел дает искомое приближение. □

2. Приближенное интегрирование функций

Если функция f(x) непрерывна на отрезке [a, b] и известна ее первообразная F(x), то определенный интеграл от этой функции в пределах от a до b может быть вычислен по *формуле Ньютона-Лейбница*

$$\int_a^b f(x)dx = F(b) - F(a), \qquad (1)$$

где F'(x) = f(x).

Однако во многих случаях первообразная функция F(x) не может быть найдена с помощью элементарных средств или является слишком сложной; вследствие этого вычисление определенного интеграла по формуле (1) может быть затруднительным или даже практически невыполнимым. Кроме того, на практике подынтегральная функция f(x) часто задается таблично и тогда само понятие первообразной теряет смысл. Во всех этих случаях используют приближенные и в первую очередь *численные методы* вычисления определенных интегралов.

Задача численного интегрирования функции заключается в вычислении значения определенного интеграла на основании ряда значений подынтегральной функции. В работе рассматриваются три правила численного вычисления однократных интегралов: *правило трапеций*, *правило Симпсона* и *правило трех восьмых*.

Обычный прием *механической квадратуры* (так называется численное вычисление однократного интеграла) состоит в том, что данную функцию f(x) на рассматриваемом отрезке [a, b] заменяют интерполирующей или аппроксимирующей функцией φ(x) простого вида (например, полиномом), а затем приближенно полагают:

$$\int_a^b f(x)dx = \int_a^b \varphi(x)dx \qquad (2)$$

Функция φ(x) должна быть такова, чтобы интеграл $\int_a^b \varphi(x)dx$ вычислялся непосредственно.

Если заменить функцию f(x) соответствующим интерполяционным полиномом Лагранжа $L_n(x)$ и вывести явно выражения для коэффициентов этого полинома, то получится следующая квадратурная формула

$$\int_a^b ydx = (b-a)\sum_{i=0}^{n} Hiyi$$

где h = (b–a)/n, $y_i = f(a+ih)$ (i=0, 1, …, n) и H_i – постоянные, называемые *коэффициентами Котеса.*

При n = 1 получаем *формулу трапеций*

$$\int_{x0}^{x1} ydx = h/2*(y_0+y_1).$$

При n = 2 получаем *формулу Симпсона*

$$\int_{x0}^{x2} ydx = h/3*(y_0+4y_1+y_2).$$

При n = 3 получаем *квадратурную формулу Ньютона*

$$\int_{x0}^{x3} ydx = 3h/8*(y_0+3y_1+3y_2+y_3).$$

(*правило трех восьмых*).

Для вычисления интеграла $\int_a^b ydx$ необходимо разделить промежуток интегрирования [a, b] на n равных частей $[x_0, x_1]$, $[x_1, x_2]$, …, $[x_{n-1}, x_n]$ и к каждому их них применить полученную формулу.

Пример 5.Пусть n=2m есть четное число и $y_i = f(x_i)$ (i=0, 1, 2, …, n) – значения функции y=f(x) для равноотстоящих точек $a=x_0, x_1, x_2, \ldots, x_n=b$ с шагом h=(b–a)/n=(b–a)/(2m). Применяя формулу Симпсона к каждому удвоенному промежутку $[x_0, x_2]$, $[x_2, x_4]$, …, $[x_{2m-2}, x_{2m}]$ длины 2h, будем иметь:

$$\int_a^b ydx = h/3*(y_0+4y_1+y_2) + h/3*(y_2+4y_3+y_4) + \ldots + h/3*(y_{2m-2}+4y_{2m-1}+y_{2m}).$$

Отсюда получаем *общую формулу Симпсона*

$$\int_a^b ydx = h/3*[(y_0+y_{2m}) + 4(y_1+y_3+ \ldots +y_{2m-1}) + 2(y_2+y_4+ \ldots +y_{2m-2})].$$

Введя обозначения

$\sigma_1 = y_1+y_3+ \ldots +y_{2m-1}$,
$\sigma_2 = y_2+y_4+ \ldots +y_{2m-2}$,

Получим общую формулу Симпсона в более простом виде:

$$\int_a^b ydx = h/3*[(y_0+y_n) + 4\sigma_1 + 2\sigma_2].$$

С помощью формулы Симпсона вычислить интеграл

$$I = \int_0^1 dx/(1+x),$$

приняв n=10.

Решение. Имеем 2m=10. Отсюда

h=(1-0)/10=0,1.

Результаты вычислений приведены в таблице.

i	**x_i**	**y_{2j-1}**	**y_j**
0	0		y_0 = 1,00000
1	0,1	0,90909	
2	0,2		0,83333
3	0,3	0,76923	
4	0,4		0,71429
5	0,5	0,66667	
6	0,6		0,62500
7	0,7	0,58824	
8	0,8		0,55556
9	0,9	0,52632	
10	1,0		0,50000 = y_n
		3,45955 (σ_1)	2,72818 (σ_2)

По общей формуле Симпсона получаем:

$I \approx h/3*(y_0+y_n+4\sigma_1+2\sigma_2) = 0,69315.$ □

Практическая часть

1. Реализация итерационных методов

Для реализации итерационных методов могут быть использованы следующие операторы цикла:

while (*выражение*) *оператор*
do *оператор* **while** (*выражение*) ;

for (*оператор-инициализации-for выражение-1*; *выражение-2*) *оператор*

Оператор «пока» (while)

В операторе **while**, входящий в него *оператор* выполняется повторно до тех пор, пока значение *выражения* не станет равным нулю. Проверка производится перед каждым выполнением *оператора*.

Пример. Вычисление суммы бесконечного ряда с заданной точностью.

```
double row(double x, double eps)
{
	int k = 1;
	double term = x, sum = x;
	while (fabs(term) > eps*fabs(sum))
	{
		k += 2;
		term *= -(x*x)/(k*(k-1));
		sum += term;
	}
	return sum;
}
```

Оператор «повторить» do-while

В операторе **do-while**, входящий в него *оператор* выполняется повторно до тех пор, пока значение *выражения* не станет равным нулю. Проверка производится после каждого выполнения *оператора*.

Пример. Уточнение корня уравнения с заданной точностью по методу Ньютона.

```
double root(double a, double b, double eps)
{
	double xold, xnew = (b+a)/2;
	do
	{
		xold = xnew;
		xnew = xold-f(xold)/g(xold);
	} while (fabs(xnew-xold) > eps);
	return xnew;
}
```

Оператор итерации (for)

Оператор итерации (оператор цикла **for**) эквивалентен

оператор-инициализации-for
while (*выражение-1*) {
	оператор
	выражение-2;
}

за исключением того, что оператор **continue** в *операторе* приводит к выполнению *выражения-2* и, далее, очередному вычислению *выражения-1*. Оператор инициализации может быть как оператором выражением, так и оператором объявлением.

Примеры. Вычисление приближенного значения определенного интеграла по формуле трапеций.

```
double area(double a, double b, int n)
{
	double s=0, x=a, h=(b-a)/n;
	if (n > 1)
```

```
            for(int i=1; i<n; x+=h, s+=f(x), i++);
        return h*(f(a)+f(b)+2*s)/2;
}
```

2. Обработка исключительных ситуаций

В C исключение реализуется с помощью макроса assert(), который может использоваться для проверки сделанных программой вычислений. Макрос assert() объявляется в заголовочном файле assert.h.

Макрос assert() добавляет к программе процедуру диагностики. После выполнения, если выражение ложно (то есть, результат сравнения 0), assert() пишет информацию о вызове в поток stderr и вызывает функцию abort(). Информация, которая пишется в stderr включает в себя:

- имя файла с исходным кодом (предопределённый макрос __FILE__)
- номер строки файла с исходным кодом (предопределённый макрос __LINE__)
- функция в исходном коде (предопределённый макрос __func__) (добавлено в стандарте C99)
- текст выражения, значение которого равно нулю 0

Для того, чтобы отключить проверку, не обязательно исключать её из кода или комментировать объявление макроса, достаточно лишь объявить ещё один макрос — NDEBUG в программе перед #include <assert.h>, тогда объявление макроса assert() будет иметь следующий вид:

```
#define assert(ignore)((void) 0)
```

и поэтому никак не будет влиять на работу программы.

Пример.

```
#include <stdio.h>
#include <assert.h>

int test_assert ( int x )
{
   assert( x <= 4 );
   return x;
}

int main ( void )
{
  int i;

    for (i=0; i<=9; i++){
        test_assert( i );
        printf("i = %d\n", i);
    }

  return 0;
}
```

Результат.

```
i = 0
i = 1
i = 2
i = 3
i = 4
assert: assert.c:6: test_assert: Assertion 'x <= 4' failed.
Aborted
```

Задание

Составить программу вычисления всех действительных корней уравнения с заданной точностью (каждым из трех методов) и вычисления определенного интеграла с заданной точностью (каждым из трех методов).
При составлении программы использовать средства обработки исключений.

При вычислении интеграла с точностью ε использовать правило Рунге, суть которого состоит в том, что выполняют вычисление интеграла с двумя разными шагами изменения переменной x, а затем сравнивают результаты и получают оценку точности. Наиболее часто используемое правило связано с вычислением интеграла дважды: с шагом Δx и шагом Δx/2.

Для методов прямоугольников и трапеций рекомендовано начальное число разбиений выбирать согласно следующему выражению:

$$n = floor((b–a)/sqrt(\varepsilon)) + 1.$$

Для метода парабол рекомендовано начальное число разбиений выбирать согласно следующему выражению:

$$n = floor((b–a)/2*sqrt(sqrt(\varepsilon))) + 1.$$

В программе вычисления интеграла с точностью ε во внутреннем цикле находят значение определенного интеграла при заданном (фиксированном) числе разбиений интервала интегрирования. Во внешнем цикле производится сравнение значений интегралов, вычисленных при числе шагов, равных n и 2n соответственно. Если требуемая точность не достигнута, то число разбиений удваивается, а в качестве предыдущего значения интеграла берут текущее и вычисление интеграла выполняется при новом числе разбиений.

В дальнейшем в программах при вычислении значений аргумента используют формулу арифметической прогрессии $x=a+(i–1)\Delta x$, а не формулу накопления суммы $x=x+\Delta x$, так как в первом случае меньше погрешность вычислений.

Уравнения и подынтегральные функции выбрать по своему усмотрению.

Приложение А

Определение операторов цикла в стандарте ISO/IEC 9899:201x

6.8.5 Iteration statements

Syntax

1 *iteration-statement:*
while (*expression* **)** *statement*
do *statement* **while (** *expression* **) ;**
for (*expressionopt* **;** *expressionopt* **;** *expressionopt* **)** *statement*
for (*declaration expressionopt* **;** *expressionopt* **)** *statement*

Constraints

2 The controlling expression of an iteration statement shall have scalar type.

3 The declaration part of a **for** statement shall only declare identifiers for objects having storage class **auto** or **register**.

Semantics

4 An iteration statement causes a statement called the *loop body* to be executed repeatedly until the controlling expression compares equal to 0. The repetition occurs regardless of whether the loop body is entered from the iteration statement or by a jump.155)

5 An iteration statement is a block whose scope is a strict subset of the scope of its enclosing block. The loop body is also a block whose scope is a strict subset of the scope of the iteration statement.

6 An iteration statement whose controlling expression is not a constant expression,156) that performs no input/output operations, does not access volatile objects, and performs no synchronization or atomic operations in its body, controlling expression, or (in the case of a **for** statement) its *expression-3*, may be assumed by the implementation to terminate.157)

6.8.5.1 The while statement

1 The evaluation of the controlling expression takes place before each execution of the loop body.

6.8.5.2 The do statement

1 The evaluation of the controlling expression takes place after each execution of the loop body.

6.8.5.3 The for statement

1 The statement

for (*clause-1* **;** *expression-2* **;** *expression-3* **)** *statement*

behaves as follows: The expression *expression-2* is the controlling expression that is evaluated before each execution of the loop body. The expression *expression-3* is evaluated as a void expression after each execution of the loop body. If *clause-1* is a declaration, the scope of any identifiers it declares is the remainder of the declaration and the entire loop, including the other two expressions; it is reached in the order of execution before the first evaluation of the controlling expression. If *clause-1* is an expression, it is evaluated as a void expression before the first evaluation of the controlling expression.158)

2 Both *clause-1* and *expression-3* can be omitted. An omitted *expression-2* is replaced by a nonzero constant.

---

155) Code jumped over is not executed. In particular, the controlling expression of a **for** or **while** statement is not evaluated before entering the loop body, nor is *clause-1* of a **for** statement.

156) An omitted controlling expression is replaced by a nonzero constant, which is a constant expression.

157) This is intended to allow compiler transformations such as removal of empty loops even when termination cannot be proven.

158) Thus, *clause-1* specifies initialization for the loop, possibly declaring one or more variables for use in the loop; the controlling expression, *expression-2*, specifies an evaluation made before each iteration, such that execution of the loop continues until the expression compares equal to 0; and *expression-3* specifies an operation (such as incrementing) that is performed after each iteration.