Перевод:

**Выполнение программы**

1 Семантическое описание в данном Международном стандарте определяет параметризованную недетерминированную абстрактную машину. Данный Международный стандарт не накладывает ограничений к структуре соответствующих реализаций. В частности, они не обязаны копировать или эмулировать структуру абстрактной машины. Скорее всего, соответствующим реализациям эмуляция потребуется (только) для наблюдения за поведением абстрактной машины, как описано ниже в пункте 6.

2 Ряд частей и операций абстрактной машины описаны в данном Международном стандарте как определенные реализацией (например, sizeof(int)). Они составляют параметры абстрактной машины. Каждая реализация должна включать документацию, которая описывает характеристики и поведение в таких случаях. Такая документация должна определять экземпляр абстрактной машины, который соответствует данной реализации (именуемый ниже, как “соответствующий экземпляр”).

3 Ряд других частей и операций абстрактной машины описаны в данном Международном стандарте как неопределенные (например, вычисление выражений в инициализаторе new, если функция выделения памяти не может ее выделить (8.3.4)). Там, где это возможно, данный Международный стандарт определяет набор допустимых моделей поведения. Это определяет недетерминированные части абстрактной машины. Таким образом, экземпляр абстрактной машины может иметь больше одного возможного варианта исполнения для заданных программы и входных параметров.

4 Ряд других операций описаны в данном Международном стандарте как неопределенные (например, результат модификации константного объекта). [Прим.: Данный международный стандарт не предъявляет требований к поведению программ, которые содержат неопределенное поведение. – конец прим.]

5 Соответствующая реализация, выполняя хорошо сконструированную программу, должна обеспечивать такое наблюдаемое поведение, которое наблюдается при одном из возможных вызовов соответствующего экземпляра абстрактной машины с той же программой и теми же входными параметрами. Однако, если какой-либо из таких вызовов содержит неопределенное поведение, данный Международный стандарт не накладывает ограничений на реализацию выполнения данной программы с данными входными параметрами (даже в отношении операций, которые предшествуют операции с неопределенным поведением).

6 Экземпляр каждого объекта с автоматическим временем жизни (6.7.3) связан с каждой точкой входа в его блок [Прим. пер.: Блок – часть кода исполняемой программы заключённая в фигурные скобки]. Такой объект существует и хранит последнее сохраненное значение пока блок выполняется или приостановлен (посредством вызова функции или получении сигнала).

7 Список требований к соответствующим реализациям:

(7.1) – Обращения через изменяемые универсальные левосторонние вычисляются строго по правилам абстрактной машины.

(7.2) – При завершении программы все данные, записанные в файлы, должны быть идентичны одному из возможных результатов, который был бы получен в результате выполнении программы в соответствии с абстрактной семантикой.

(7.3) – Ввод и вывод в интерактивных устройствах должны быть реализованы таким образом, чтобы вывод запроса фактически осуществлялся до того, как программа начнет ожидание ввода. Что представляет собой интерактивное устройства определяется реализацией. В совокупности это называется наблюдаемым поведение программы. [Прим.: Более строгие соответствия между абстрактной и фактической семантиками определяются каждой из реализаций самостоятельно. – конец прим.]

8 [Прим.: Операторы могут быть разгруппированы в соответствии с обычными математическими правилами только там, где эти операторы действительно проявляют ассоциативные или коммутативные свойства. Например, в приведенном ниже фрагменте

int a, b;

/\* ... \*/

a = a + 32760 + b + 5;

выражение ведет себя также, как

a = (((a + 32760) + b) + 5);

из-за свойств ассоциативности и старшинства операторов. Таким образом, результат суммы (a + 32760) складывается с b, полученный результат складывается с 5, и, далее, значение результата выражения присваивается переменной a. На машине, где переполнения вызывают исключение и в которой диапазон значений для типа int равен [-32768, +32767], реализация не может изменить данное выражение на

a = ((a + b) + 32765);

так как если бы значения a и b были соответственно равны -32754 и -15, сумма a + b могла бы вызвать исключение, хотя оригинальное выражение исключений не вызывает; также выражение не может быть переписано как

a = ((a + 32765) + b);

или

a = (a + (b + 32765));

так как значения a и b могут быть соответственно 4 и -8 или -17 и 12. Однако на машинах, где переполнение не вызывает исключение и на которых результат переполнения обратим, приведенное выше выражение может быть заменено реализацией на любое из приведенных, потому что будет получен один и тот же результат. – конец прим.]

9 Составное выражение определяется следующим образом:

(9.1) – Составным выражением выражения является это выражение.

(9.2) – Составными выражениями списка инициализации в фигурных скобках или (возможно заключенного в скобки) списка выражений являются составные выражения элементов соответствующего списка.

(9.3) – Составными выражениями инициализатора фигурные скобки или равно как формы оператора-инициализатора вида равно, является составные выражения оператора-инициализатора.

[Пример:

struct A { int x; };

struct B { int y; struct A a; };

B b = { 5, { 1+1 } };

Составное выражение инициализатора использует для инициализации b и 5 и 1 + 1. – конец примера].

10 Непосредственными подвыражениями выражения e являются:

(10.1) – Составное выражение операндов e (Раздел 8).

(10.2) – Любой вызов функции в котором неявно вызывается e

(10.3) – Если e – лямбда-выражение (8.1.5), то подвыражениями e являются инициализация внутренних переменных, захваченных посредством копирования и составные выражения инициализатора при инициализирующем-захвате

(10.4) – Если е – вызов функции или неявный вызов функции (8.2.2), то подвыражениями е являются составные выражения каждого аргумента по умолчанию (11.3.6), которые используются в вызове или

(10.5) – Если е – создает агрегированный объект (11.6.1), подвыражениями е являются составные выражения для инициализаторов каждого из членов такого объекта, определенных по умолчанию.

11 Подвыражение выражения е является непосредственным подвыражением е или подвыражением непосредственного подвыражения е. [Прим.: Выражения, появляющиеся в составном операторе, лямбда-выражений не являются подвыражениями лямбда-выражений. – конец прим.]

12 Полное выражение это:

(12.1) – Выраженный числом операнд (Раздел 8).

(12.2) – Константное выражение (8.20).

(12.3) – Декларатор инициализации (Раздел 11) или инициализатор памяти (15.6.2), включая составное выражение инициализатора.

(12.4) – Вызов деструктора, сгенерированный в конце жизни объекта отличного от временного объекта (15.2) или

(12.5) – Выражение, которое не является подвыражением другого выражения и которое в ином случае не является частью другого полного выражения. Если конструкция языка определяется для неявного вызова функции, использование такой конструкции считается выражением для этого определения. Преобразования, применяемые к результату выражения для удовлетворения требований конструкции языка в которой выражение появляется, также считается частью полного выражения. Для инициализатора, выполняющего инициализацию сущности (включая расчет стандартных инициализаторов членов совокупности) также является частью полного выражения.

[Пример:

struct S {

S(int i): I(i) { } // полное выражение - инициализация I

int& v() { return I; }

~S() noexcept(false) { }

private:

int I;

};

S s1(1); // полное выражение – вызов S::S(int)

void f() {

S s2 = 2; // полное выражение – вызов of S::S(int)

if (S(3).v()) // полное выражение включает в себя lvalue-to-rvalue и

// int-to-bool преобразования и выполняется до

// удаления временного объекта в конце полного выражения

{ }

bool b = noexcept(S()); // спецификация исключения деструктора S

// рассматривается для noexept

// полное выражение – удаление s2 в конце блока

}

struct B {

B(S = S(0));

};

B b[2] = { B(), B() }; // полное выражение – внутренняя инициализация

// включая удаление временных объектов

- конец примера]

13 [Прим.: Вычисление полного выражения может включать вычисление подвыражений, которые не являются лексическими частями полного выражения. Например, подвыражения, участвующие в расчете аргументов по умолчанию (11.3.6) считаются созданными, в выражении, которое вызывает функцию, а не в выражении, которое определяет аргумент по умолчанию. – конец прим.]

14 Считывание объекта, помеченного как изменяемое универсальное левостороннее значение (6.10), модификация объекта, вызов библиотечной функции ввода/вывода или вызов функции, которая выполняет какую-либо из этих операций вызывает побочный эффект, который является изменением в состоянии среды выполнения. Вычисление выражения (или подвыражения) в общем случае включает и расчет значения (включая определение идентичности объекта для вычисления универсального левостороннего значения и извлечение значения, которое было присвоено ему ранее для вычисления чистого правостороннего значения) и создание побочных эффектов. Когда возвращается вызов библиотечной функции ввода/вывода или рассчитывается доступ через изменяемое универсальное левостороннее значение, побочный эффект считается законченным, даже когда какие-либо внешние действия, которые подразумеваются в вызове (такие как сам ввод/вывод) или изменяемый доступ еще не завершены.

15 Отношение “упорядочено до” (предшествует) представляет собой асимметричную, переходную, попарную связь между вычислениями, выполняемыми в одном потоке (4.7), которая включает частичную упорядоченность этих вычислений. Для двух любых вычислений А и В, если А предшествует В (или, что тоже самое, B в отношении “упорядочено после” с А (следует за А)), то выполнение А должно предшествовать выполнению В. Если А не предшествует В и В не предшествует А, тогда А и В неупорядочены [Прим.: Выполнение неупорядоченных вычислений может вызвать бесконечный цикл - конец прим.] Вычисления А и В неопределенно упорядочены, когда и А предшествует В или В предшествует А, но это не указано явно. [Прим.: Неопределенная упорядоченность вычислений не может привести к бесконечному циклу, и любой из расчётов может быть выполнен первым - конец прим.] Выражение X предшествует выражению Y, если все расчеты значений и все побочные эффекты, которые вызывает выражение X, предшествуют всем вычислениям значений и всем побочным эффектам, которые вызывает выражение Y.

16 Каждый расчет значения и побочный эффект, ассоциированный с полным выражением, предшествует каждому расчету и побочному эффекту, ассоциированному со следующим полным выражением, которое будет вычислено.

17 За исключением случаем, когда это явно указано, вычисление операндов индивидуальных операторов и подвыражений индивидуальных выражений неупорядоченно [Прим.: В выражениях, которые вычисляются более одного раза в течении вызова программы, неупорядоченные вычисления и вычисления с неопределенной упорядоченностью в их подвыражениях не обязательно должны выполняться в разных вычислениях - конец прим.] Расчеты значений операндов оператора предшествуют расчету значения результата выполнения оператора. Если побочный эффект на участке памяти (4.4) имеет неупорядоченное отношение к другому побочному эффекту на том же участке памяти или расчет значения использует значение какого-либо объекта из того же участка памяти, и они потенциально не выполняются параллельно – такое поведение не определено [Прим.: Следующий раздел налагает аналогичные, но более общие ограничения на потенциальные параллельные расчеты - конец прим.]

[Пример:

void g(int i) {

i = 7, i++, i++; // i стало равно 9

i = i++ + 1; // значение i не определенно

i = i++ + i; // неопределенное поведение

i = i + 1; // значение i не определено

}

- конец примера]

18 При вызове функции (не важно является ли функция встроенной), любой расчет значения и побочный эффект, ассоциированный с любым выражением в аргументе функции или с постфиксным выражением, обозначающим вызов функции, предшествует выполнению любого выражения или оператора в теле вызываемой функции. Для каждого вызова функции F, для каждого вычисления A, который происходит внутри F и для каждого вычисления B, который не происходит внутри F, но выполняется в том же потоке и как часть того же обработчика сигналов (если такой имеется), A предшествует B или B предшествует A. [Прим.: Если с другой стороны A и В могут быть не упорядочены, тогда они имеют неопределенную упорядоченность, - конец прим.] В некоторых контекстах в C++ вычисления вызовов функций происходят даже если в единице трансляции нет соответствующего синтаксиса вызова функции [Например: Вычисление new-выражения вызывает одну или более функцию выделения памяти или конструктор; см. 8.3.4. Другой пример – вызов функции преобразования (15.3.2) может возникать в контексте, в котором нет синтаксиса вызова функции – конец примера] Ограничения в упорядочивании исполнения вызовов (в описании выше) являются особенностями вызовов функций как вычислений, каким бы ни был синтаксис выражений где эти функции вызываются.

19 Если обработчик сигналов исполняется как результат вызова функции std::raise, тогда исполнение обработчика происходит после вызова функции std::raise и до возврата из нее [Прим.: Когда сигнал получен по другим причинам, исполнение обработчика сигналов обычно неупорядочено по отношению к остальной части программы - конец прим.]

**Время жизни объектов**

1 Время жизни объектов или ссылок есть свойства времени выполнения объекта или ссылки. Говорят, что объекта имеет нетривиальную инициализацию, если он относится к типу класса или агрегатному типу и он сам или один из его подобъектов инициализируется конструктором, отличным от тривиального конструктора. [Прим.: Инициализация через тривиальный копирующий/перемещающий конструктор является нетривиальной - конец прим.] Время жизни объекта типа Т начинается когда:

(1.1) – выделена память с правильным выравниванием и размером, которые подходят для типа Т, и

(1.2) – если объект имеет нетривиальную инициализацию, его инициализация завершена,

исключением является случай, когда объект является членом объединения или его подобъектом, в этом случае его время жизни начинается когда он будет инициализирован как член объединения или, как наступит случай, описанный в 12.3. Время жизни объекта *о* типа Т заканчивается, когда:

(1.3) – если Т тип класса с нетривиальным деструктором (15.4), начнется вызов деструктора или

(1.4) – память, где располагается объект, освобождено или повторно использовано для объекта, который не является вложенным в *o* (4.5)

2 Время жизни ссылки начинается, когда завершена ее инициализация. Время жизни ссылки заканчивается, также, как и у скалярного объекта.

3 [Прим.: 15.6.2 описывает время жизни базового подобъекта и подобъекта-члена объекта - конец прим.]

4 Свойства, приписываемые объектам и ссылкам в данном Международном стандарте, могут быть применены для данных объекта и ссылки только на время их жизни. [Прим.: В частности, до начала и по окончанию времени жизни объекта есть существенные ограничения на использование объектов, описанные ниже в 15.6.2 и в 15.7. Также, поведение объекта во время создания или удаления может не соответствовать поведению объекта после того, как началось и до того как закончилось его время жизни. 15.6.2 и 15.7 описывают поведение объектов во время фаз создания и удаления. - конец прим.]

5 Программа может закончить время жизни любого объекта начав перераспределение памяти, с которой объекта ассоциирован или через явный вызов деструктора объекта типа класса с нетривиальным деструктором. Для объекта типа класса с нетривиальным деструктором, программе не требуется явно вызывать деструктор до того, как память с которой ассоциирован объект будет перераспределена или освобождена; однако, если нет явного вызова деструктора или если оператор-удаления не используется для освобождения памяти, деструктор не должен быть вызван неявно и любая программа, зависящая от побочных эффектов, порождаемых деструктором, будет иметь неопределенное поведение.

6 До начала времени жизни объекта и после выделения памяти, ассоциированной с объектом или после окончания времени жизни объекта и перед перераспределением или освобождением памяти, ассоциированной с объектом, любой указатель, представленный адресом в памяти и указывающий на этот объект, может быть использован ограниченным образом. Для объекта в стадии создания или удаления см. 15.7. Иначе, такой указатель указывает на распределенную память (6.7.4.2) и использование указателя, как если бы указатель имел тип void\*, также определено. Косвенное обращение через такой указатель допускается, но результирующая левосторонняя ссылка может быть использована ограниченным образом, как описано ниже. Программа имеет неопределенное поведение, если:

(6.1) – объект будет или был типом класса с нетривиальным деструктором и указатель используется как операнд оператора-удаления,

(6.2) – указатель используется для доступа к нестатическим переменным или вызова нестатических функций-членов или

(6.3) – указатель неявно преобразован в указатель виртуального базового класса или

(6.4) – указатель используется как операнд static\_cast (8.2.9), кроме случаев, когда выполняется преобразование из указателя в константный изменяемый void или из указателя в константный изменяемый void и впоследствии из указателя в константный изменяемый char, константный изменяемый unsigned char или константный изменяемый std::byte (21.2.1) или

(6.5) – указатель используется как операнд dynamic\_cast (8.2.7)

[Пример:

#include <cstdlib>

struct B {

virtual void f();

void mutate();

virtual ~B();

};

struct D1 : B { void f(); };

struct D2 : B { void f(); };

void B::mutate() {

new (this) D2; // перераспределенная память – конец времени жизни \*this

f(); // неопределенное поведение

... = this; // OK, this указывает на допустимую память

}

void g() {

void\* p = std::malloc(sizeof(D1) + sizeof(D2));

B\* pb = new (p) D1;

pb->mutate();

\*pb; // OK: pb указывает на допустимую память

void\* q = pb; // OK: pb указывает на допустимую память

pb->f(); // неопределенное поведение, время жизни \*pb закончилось

}

- конец примера]

7 Аналогично, до начала времени жизни объекта но после того, как память под этот объекта была распределена или после окончания жизни объекта, но до того как память, распределенная под этот объекта была перераспределена или освобождена, любое универсальное левостороннее значение, которое указывает на оригинальный объекта может быть использовано ограниченным образом. Для объекта в стадии создания или удаления см. 15.7. В остальных случаях такое универсальное левостороннее значение указывает на выделенную память (6.7.4.2) и использует свойства универсального левостороннего значения, которые не зависят от того четко ли определено его значение. Программа имеет неопределенное поведение, если:

(7.1) – универсальное левостороннее значение используется для доступа к объекту или

(7.2) – универсальное левостороннее значение используется для обращения к нестатическим членам или для вызова нестатических функций объекта или

(7.3) – универсальное левостороннее значение привязано к ссылке на виртуальный базовый класс или

(7.4) – универсальное левостороннее значение используется как операнд dynamic\_cast или операнд typeid.

8 Если после окончания времени жизни объекта и до того, как память под этот объекта была перераспределена или освобождена, а в памяти, ассоциированной с оригинальным объектом, создается новый объект указатель, который указывает на оригинальный объекта, ссылка, которая указывает на оригинальный объекта или имя оригинального объекта начнет автоматически указывать на новый объекта и как только начнется время жизни нового объекта, может быть использовано для управления этим объектом, если:

(8.1) – память для нового объекта в точности повторяет память, которая была распределена под оригинальный объект и

(8.2) – новый объект имеет тот же тип, что и оригинальный объекта (игнорируя высокоуровневые cv-квалификаторы) и

(8.3) – тип оригинального объекта не является константным и если тип класса не содержит каких-либо нестатических членов, типы которых имеют константные или ссылочные типы и

(8.4) – оригинальный объект был последним в иерархии наследования (4.5) от типа Т и новый объекта также является последним в иерархии наследования от типа Т (т.е. они не подобъекты базового класса)

[Пример:

struct C {

int i;

void f();

const C& operator=( const C& );

};

const C& C::operator=( const C& other) {

if ( this != &other ) {

this->~C(); // конец времени жизни \*this

new (this) C(other); // создан новый объекта типа С

f(); // четко определенный вызов

}

return \*this;

}

C c1;

C c2;

c1 = c2; // четко определенный вызов

c1.f(); // четко определенный вызов; c1 указывает на новый объект типа C

- конец примера] [Прим.: Если эти условия не встречаются, указатель на новый объект может быть получен из указателя на память вызовом sta::launder (21.6) - конец при.]

9 Если программа завершила время жизни объекта типа Т со статическим, потоковым или автоматическим временем распределения памяти и, если Т имеет нетривиальный деструктор, программа должна гарантировать, что объект оригинального типа занимает тоже место в памяти, когда выполняется неявный вызов деструктора; иначе поведение программы неопределенно. Это верно даже если программа покинула блок через исключение.

[Пример:

class T { };

struct B {

~B();

};

void h() {

B b;

new (&b) T;

} // Неопределенное поведение при выходе из блока

- конец примера]

Создание нового объекта в памяти под готовый константный объект со статическим, потоковым или автоматическим временем распределения памяти или в памяти, которая была зарезервирована под такой константный объект до окончания времени его времени жизни, вызывает неопределенное поведение.

[Пример:

struct B {

B();

~B();

};

const B b;

void h() {

b.~B();

new (const\_cast(&b)) const B; // неопределенное поведение

}

- конец примера]

11 В данной секции «до» и «после» указывает на «произойдет до» отношение (4.7). [Прим.: Следовательно, неопределенное поведение возникает, если ссылаться на объект, созданный в одном потоке, из другого потока без надлежащей синхронизации - конец прим.]

**Типы**

1 [Прим.: параграф 6.9 и его подпункты налагают требования на реализацию касательно представления типов. Есть два вида типов: фундаментальные типы и составные типы. Типы описывают объекты (4.5), ссылки (11.3.2) или функции (11.3.5) – конец прим.]

2 Для любого объекта (отличного от подобъекта базового класса) или тривиально копируемого типа Т, является ли объект обладателем допустимого значения типа Т, лежащие в основе байты делают объект копируемым в массив объектов типа char или std::byte (21.2.1). Если содержимое этого массива копируется обратно в объект, объект должен впоследствии содержать его оригинальное значение.

[Пример:

#define N sizeof(T)

char buf[N];

T obj; // obj инициализирован его первоначальным значением

std::memcpy(buf, &obj, N); // между этими двумя вызовами std::memcpy,

//obj может быть изменен

std::memcpy(&obj, buf, N); // в этой точке каждый подобъект obj скалярного типа

// содержит первоначальное значение

- конец примера]

3 Для любого тривиального копируемого типа Т, если два указателя на Т указывают на различные объекты obj1 и obj2, где ни obj1, ни obj2 не являются подобъектами базового класса, если лежащие в основе байты делают obj1 копируемым в obj2, obj2 должен сохранить то же значение, что и obj1.

[Пример:

T\* t1p;

T\* t2p;

// при условии, что t2p указывает на инициализированный объект...

std::memcpy(t1p, t2p, sizeof(T));

// в данной точке, каждый подобъект тривиально копируемого типа \*t1p содержит

// то же значение, что и соответствующий тип в \*t2p

- конец примера]

4

|  |  |
| --- | --- |
| glvalue | универсальное левостороннее значение |
| prvalue | чистое правостороннее значение |
| braced-init-list | список инициализации в фигурных скобках |
| brace-or-equal-initializer | инициализатор фигурные скобки или равно |
| initializer-clause | оператор-инициализатор |
| lambda-expression | лямбда выражение |
| unevaluated | выраженный числом |
| init-declarator | декларатор инициализации |
| mem-initializer | инициализатор памяти |
| unsequenced | неупорядоченный |
| overlap | бесконечный цикл |
| inline | встроенный |
| translation unit | единица трансляции |
| sequenced before | отношение “упорядочено до” (предшествует) |
| sequenced after | отношение “упорядочено после” (следует за) |