Программирование на C++

Будут рассмотрены некоторые особенности языка C++ и даны отдельные рекомендации по его использованию.

Скотт Мейерс «Наиболее эффективное использование C++».

Указатели и массивы

Одномерный массив можно отождествить с указателем, который проинициализирован адресом выделенной под массив памяти.

```
int a[5];
int* b = a;
```

При этом запись **a[i]** полностью эквивалентна ***(a+i)**.

Аналогично, запись **a[i][j]** в языке С полностью эквивалентна записи ***(*(a+i)+j)**. Эта эквивалентность имеет место как в случае, если **a** — двумерный массив, так и в случае, когда **a** — двойной указатель (указатель на указатель). Но двумерный массив и двойной указатель несовместимы.

Пример

```
int a[5][5];
int* b = (int*) a;
int** c = (int**) a;
b[7] = 0;
c[0][0] = 0;
```

Поскольку компилятор разместит массив **a** в памяти как 25 идущих подряд переменных типа **int**, обращение к **b[7]** сработает корректно.

При этом обращение **c[0][0]** почти гарантированно приведет к исключению, поскольку фактически будет выполнено

```
**((int**) (a[0][0])) = 0;
```

то есть значение будет разыменовано как указатель.

Адресная арифметика

Адресная арифметика является одной из сильных сторон языка С.

Сложение указателя **`T* p`** и целого **`i`** — увеличивает указатель на **`i*sizeof(T)`**.

Сложение 2-х указателей — не имеет смысла.

Разность 2-х указателей (одинакового типа **`T`**) — дает количество элементов типа **`T`** между ними.

Ссылки

Ссылку обычно определяют как альтернативное имя переменной. Однако дать такое же краткое и однозначное определение имени переменной уже не так просто. В данном контексте имя — это не обязательно идентификатор.

Можно мыслить ссылку как указатель, который при использовании автоматически разыменовывается.

Физически ссылка в большинстве случаев (но не всегда) действительно реализуется с помощью указателя. Однако внутреннее представление ссылки стандартом не оговорено.

Константный указатель (`*const`), как правило, может быть заменен ссылкой.

Вопросы безопасности ссылок

Синтаксис языка не запрещает получить ссылку на несуществующий объект:

```
int &a = *((int*) 0);
```

Учитывая, что исключение произойдет не при инициализации ссылки, а только при обращение к объекту, локализация подобной ошибки может оказаться нетривиальной.

Опасность неявного каста для ссылок:

```
const A &a = b;
```

Если **b** не является объектом класса, наследующего от **A**, но определен каст **b** в **A**, то **a** будет ссылкой не на **b**, а на временный объект, сконструированный из **b**.

Модификаторы

Модификатор **`const`** означает, что объявляемый объект не должен изменять свое значение после инициализации. Настоятельно рекомендуется использовать для всех переменных, которые не должны изменять свое значение, а также (если нет контрдоводов) объявлять константными все методы класса, не изменяющие состояния объекта.

Модификатор **`volatile`** предписывает компилятору при оптимизации не делать никаких предположений о значении данной переменной. Его следует использовать, в частности, если переменная может асинхронно модифицироваться из разных потоков (хотя подобных ситуаций лучше избегать).

Конструктор и оператор копирования

Если в классе не объявлен конструктор или оператор копирования, то компилятор создает его по умолчанию. Следует хорошо представлять их функциональность и оценить, достаточно ли она для данного класса.

Если объекты класса не подлежат копированию, то необходимо только объявить конструктор и оператор копирования, но не реализовывать:

```
A(const A&);
A& operator=( const A&);
```

где `A` — имя класса.

Полиморфизм

В C++ существует два механизма полиморфизма: шаблоны (параметризованные классы и функции) и виртуальные методы классов.

Виртуальный метод можно мыслить как функцию, адрес которой хранится в объекте класса, но не копируется при присваивании объектов. При этом, если вызывать метод объекта через указатель на родительский класс, то будет вызван метод, определенный в классе, которому фактически принадлежит объект.

Фактически адреса виртуальных методов хранятся не в объекте, а в таблице виртуальных методов, указатель на которую хранится в объекте.

Приведения типов

При использовании C и C++ следует иметь в виду возможность неявного (автоматического) приведения типов, которое может приводить к потере точности (например, присваивание значения **`double`** в переменную **`char`** не считается синтаксической ошибкой).

В C++ можно определять свои методы приведения типов (только для классов). Для этого используется соответствующий оператор или конструктор.

Если нужно, чтобы каст мог происходить только явно, конструктор нужно помечать как **`explicit`**.

Стили явного приведения типов

Стиль C

```
(type) variable
```

Стиль C++

```
…_cast<type>(variable)
```

Виды:

```
static_cast<>
dynamic_cast<>
const_cast<>        (mutable)
reinterpret_cast<>
```

Использование стиля C++ более предпочтительно. Кроме того, каст в стиле C не позволяет в принципе реализовать `dynamic_cast`.

Исключения

Прообразом исключений в языке C можно считать длинный переход, реализуемый стандартными функциями: setjump, longjump.

Исключения предназначены, чтобы сигнализировать о ситуации, когда невозможно продолжить текущие действия. Альтернатива исключениям — задание специальных кодов, возвращаемых функцией в случае ошибок. Преимущество исключений по сравнению с таким решением в том, что исключения нельзя игнорировать. Вместе с тем, исключения предназначены для нештатных ситуаций, и использовать их, например, как метод выхода из цикла неоправданно.

Макрос assert

Допустимым является стиль программирования, когда исключения не используются вовсе. Однако практически обязательным можно считать использование макроса **`assert`**. Макрос проверяет истинность указанного условия и в случае, если оно ложно, прерывает программу. При этом макрос работает только при компиляции программы в отладочном режиме и не замедляет работу релиза. Помимо существенного облегчения процесса отладки программы, **`assert`** документирует код, заменяя часть комментариев.

Перегрузка операторов

Возможность переопределения операторов является весьма привлекательным инструментом, однако область оправданного применения этого инструмента достаточно ограничена (за исключением оператора копирования).

Нельзя перегружать операторы ‘**&&**’, ‘**||**’, ‘**^^**’, ‘**,**’. Стандарт C++ гарантирует, что операнды логического выражения вычисляются слева направо, причем как только результат всего выражения становится известным, дальнейшие операнды не вычисляются.

```
if (x && x->f())…
```

Здесь если **x==0**, то **f()** вызвана не будет.

Перегруженные операторы имеют другое поведение.

Ввод-вывод

Для ввода–вывода в C предусмотрена стандартная библиотека stdio.h, в C++ предпочтительной считается iostream, что в значительной мере оправдано.

Недостаток stdio.h — невозможность автоматического контроля соответствия типов.

Однако в некоторых случаях использование iostream затруднительно. Кроме того, при работе с файлами больших размеров stdio.h может давать заметный выигрыш в скорости.

При этом в stdio.h есть функции, использование которых вряд ли оправдано, например функции **`fscanf()`** и **`gets()`**, вместо которых предпочтительнее **`sscanf()`** и **`fgets()`**.