Типизация данных

Объекты данных

Любая программа содержит набор операций, которые применяются к определенным данным в определенной последовательности.

Каждый язык программирования задает три категории характеристик:

- допустимые значения данных и способы размещения этих значений в памяти компьютера;
- допустимые операции (встроенные в язык и создаваемые программистом);
- операторы, управляющие последовательностью применения операций к данным.

Объекты данных

Данные хранятся в памяти компьютера в виде последовательности битов, которые группируются в байты или слова.

Элементы данных, рассматриваемые как единое целое в некий момент времени выполнения программы, принято называть объектами данных.

Примеры:

- относительно простые конструкции – числа, символы;
- более сложные конструкции (стеки, массивы, символьные строки).

Объекты данных

В ходе вычислений существует великое множество различных объектов данных.

Более того, объекты данных и отношения между ними динамически меняются в разные моменты вычислительного процесса.

Две категории объектов данных:

- создаются и явно управляются программистом – переменные, константы, файлы;
- создаются операционной системой (программист не имеет к ним непосредственного доступа) - стеки, записи активации функций, файловые буферы, списки свободной памяти.

Атрибуты объектов данных

Объект данных представляет собой достаточно сложную структуру (некий контейнер), имеющую многочисленный набор атрибутов.

Основные атрибуты:

- тип данных – задает возможные значения, применимые операции и формат хранения данных;
- имя – перечень имен, по которым к объекту данных обращаются в процессе вычислений. Пример: у объекта «целое число 5» может быть несколько имен: int a = 5, b = 5, c = 5;
- местоположение (адрес) – хранит координаты области памяти (адрес), отведенной под объект данных (определяется системными программными средствами, т.е. программист, как правило, не имеет доступа к этим средствам).

Атрибуты объектов данных

Основные атрибуты:

- значение – текущая величина (или набор величин) объекта данных;
- время жизни – период существования объекта данных в программе;
- область видимости – часть программы, в которой существует доступ к объекту данных. Существует две разновидности: локальная и глобальная.

Тип данных

Тип данных - это механизм классификации объектов данных.

Тип данных определяет:

- возможные значения объектов;
- операции, применимые к значениям объектов данного типа;
- размещение значений объектов в памяти компьютера (данные разного типа хранятся в разных областях памяти).

В каждом языке программирования имеется некий набор примитивных встроенных типов данных (int, float, char).

В большинстве современных языков, как правило, предусматриваются средства для создания новых собственных типов данных и операций для работы с ними (объектно-ориентированные языки).

Тип данных

Каждая переменная (константа) считается экземпляром конкретного типа данных.

Тип данных является важнейшим элементом системы контроля языка.

Знание типа переменной позволяет ответить на вопросы:

- можно или нельзя присвоить переменной конкретное значение;
- можно или нельзя применить к переменной конкретную операцию.

Примеры:

- нельзя занести в переменную целого типа строковую переменную;
- нельзя применить операцию деления к двум строковым переменным.

Объявления

В ходе создания программы программист задает имя и тип каждого используемого объекта данных.

Существует два вида объявлений – явное и неявное.

Явное объявление – это оператор программы, сообщающий компилятору об имени и типе объекта данных, который будет использоваться в программе.

Место размещения объявления в программе определяет время жизни объекта данных.

Возможны два варианта:

- локальные переменные – объявляются в начале блока, время жизни ограничено этим блоком;
- глобальные переменные – объявляются вне всех блоков и функций, время жизни ограничено временем работы программы.

Объявления

В некоторых языках программирования допускаются неявные объявления, или объявления по умолчанию.

Неявное объявление – это такое объявления, в котором тип данных объекта данных не указан.

Пример. В языке FORTRAN все переменные, начинающиеся с букв I, J, K, L, M, N по умолчанию считаются целочисленными. Если нужно переменную NUMBER сделать вещественной, то это надо прописать явно: REAL NUMBER.

Замечание. Неявные объявления могут быть пагубными для надежности ПО, поскольку препятствуют выявлению на этапе компиляции различных опечаток или ошибок программиста.

Логический вывод типа

Еще одна разновидность неявного объявления основана на контексте. Иногда его называют логическим выводом типа.

В простейшем случае контекст задает тип начального значения, которое присваивается переменной в операторе объявления.

Пример. В языке C# объявление переменной var включает в себя начальное значение, тип которого задает тип переменной:

var sum = 0; - тип переменной sum = int;

var total = 0.0; - тип переменной total = float;

var name = “Olga”; - тип переменной name = string.

Контроль типов

На аппаратном уровне компьютера контроль типов не осуществляется.

Например, аппаратная реализация операции целочисленного сложения не способна проверить, являются ли переданные ей два операнда целыми числами, для нее это просто последовательность битов.

Вывод: на аппаратном уровне обычные компьютеры очень ненадёжны ! Поскольку они не реагируют на ошибки в типах данных.

Поэтому контроль типов осуществляет компилятор. Именно он проверяет факт получения каждой операцией нужного количества операндов нужного типа.

Пример. Перед выполнением операции y = a * b + c компилятор должен проверить наличие у каждой операции (в данном случае – умножение, сложение и присваивание) операндов правильного типа. Т.е. если переменная b – это строка, то компилятор выдаст ошибку «несоответствие типов операндов».

Виды контроля типов

Различают два вида контроля типов:

- статический контроль типов – осуществляется на этапе компиляции программы; необходимые сведения о типах данных поступают компилятору из явных объявлений;
- динамический контроль типов – осуществляется в процессе выполнения программы; явные объявления типов данных отсутствуют.

Статический контроль типов

Исходные данные, необходимые для организации статического контроля типов:

- для каждой операции определены количество и типы данных, (как для операндов, так и для результата);
- тип каждого объекта данных (переменной или экземпляра типа) известен и не меняется в ходе выполнения программы;
- типы всех констант тоже понятны: тип литеральной константы определяется из её синтаксиса (10 – целая константа, 254.53 – вещественная константа), а тип именованной константы зафиксирован в её определении.

Статический контроль типов

- Вся исходная информация собирается компилятором при анализе текста программы и заносится в специальную таблицу символов, которая накапливает все сведения о типах переменных и операций.
- После завершения сбора информации компилятор проверяет все операции на предмет правильности их операндов.
- Поскольку статический контроль типов охватывает все операции программы, то проверке подвергаются все варианты вычислений, и, следовательно, отпадает необходимость в дальнейшем контроле.

Достоинства: значительный выигрыш в скорости вычислений и эффективности использования памяти.

Динамический контроль типов

- явные объявления типов данных отсутствуют (как для операндов, так и для результата), т.е. все объекты считаются объектами, не имеющими типа;
- тип каждого объекта данных (переменной или экземпляра типа) может изменяться в ходе выполнения программы;
- контроль типов (возможность осуществления той или иной операции) производится на каждом шаге по ходу выполнения программы.

Основное преимущество – гибкость программы. Объявлений типов нет => объект данных может изменить свой тип в любой момент вычислений. Программист может создавать настраиваемые программы, работающие с данными любого типа.

Динамический контроль типов

Основные недостатки:

- понижение надежности вычислений – работа над возможными ошибками типизации переносится с подготовительного этапа (этап компиляции) на исполнительный этап (этап вычислений); проверке подвергаются только отдельные объекты данных, задействованные в текущей операции, т.е. многие из объектов остаются вне контроля;
- снижение скорости вычислений – динамический контроль типов основан на информации, которая хранится в дескрипторах объектов данных, доступ к которым должен быть реализован на программном уровне, т.к. аппаратура таких действий не обеспечивает. Это приводит к существенному замедлению скорости вычислений;
- возрастание накладных расходов памяти - дескриптор объекта данных должен храниться в течении всего вычислительного процесса, что предъявляет повышенные запросы к требуемой памяти.

Системы типизации данных

Система типизации данных – вся сумма правил, связанных с типами данных.

Системы типизации данных включают в себя несколько категорий правил:

- определение атрибутов объектов данных и их связывание;
- определение типов данных;
- определение типов выражений;
- правила преобразования и проверки данных.

Связывание – это процесс установления связи между атрибутами.

Время связывания – момент времени, когда происходит связывание.

Когда осуществляется связывание ?

Связывание может происходить:

- при проектировании языка;
- при разработке компилятора языка;
- при компиляции программы;
- при загрузке программы в память;
- при выполнении программы.

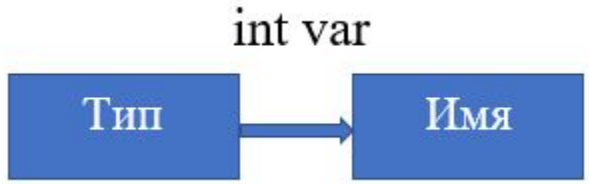
Примеры связывания

- символ «+» обычно связывается с операцией сложения на этапе проектирования языка;
- тип int в языке C++ связывается с диапазоном возможных значений во время компиляции программы (в зависимости от типа процессора – это 2 байта или 4 байта);
- в языках C++, Java и др. переменная связывается с конкретным типом данных на этапе компиляции программы;
- переменная может связываться с ячейкой памяти при загрузке программы в память;
- вызов библиотечной функции связывается с кодом этой функции на этапе редактирования связей и загрузки (сборка программы).

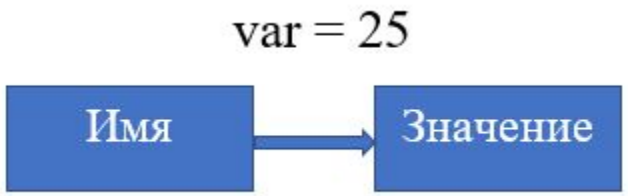
Замечание. Как правило, программист может участвовать в связывании только на этапах компиляции и выполнения программы.

Примеры связывания

- связывание типа
  (имя связывается с типом)



- связывание значения
  (имя связывается со значением)

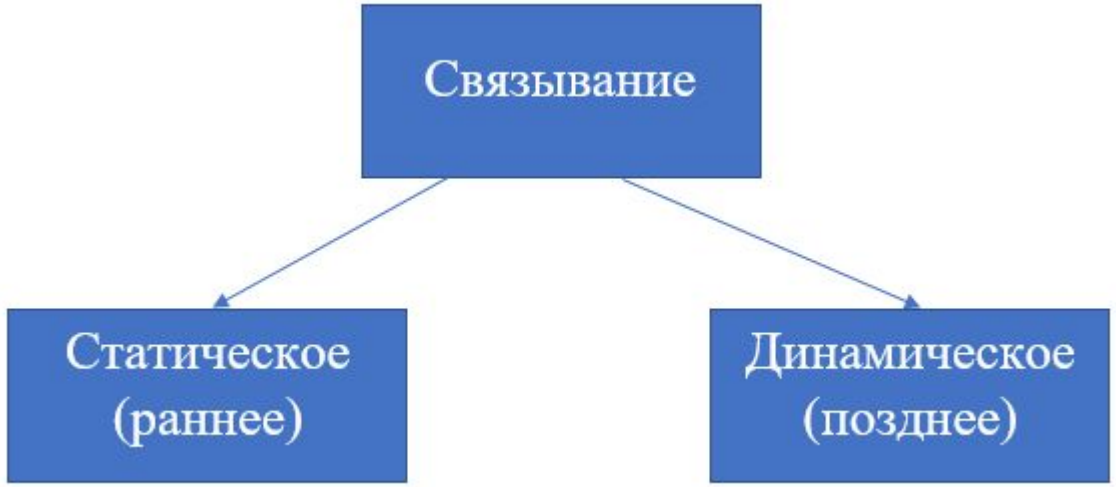


Примеры связывания

Необходимые связывания для операции: var = var + 10;

- связывание типа для переменной var – на этапе компиляции;
- связывание символа «+» после определения типов операндов – на этапе компиляции;
- внутреннее представление константы 10 – связывание на этапе компиляции;
- значение переменной var – связывание во время выполнения оператора.

Разновидности связывания



Статическое связывание выполняется на этапе компиляции.

Динамическое связывание выполняется в процессе выполнения программы.

Основной постулат динамического связывания – любой переменной может быть присвоено значение любого типа !

Динамическое связывание типов

Основные постулаты динамического связывания:

- любой переменной может быть присвоено значение любого типа;
- во время выполнения программы тип одной и той же переменной может меняться многократно.

Таким образом:

- при динамическом связывании оператор объявления переменной не содержит имени типа;
- тип переменной определяется после присвоения ей значения оператором присваивания;
- при выполнении оператора присваивания переменная в его левой части получает тип переменной, выражения или значения, находящегося в его правой части.

Примеры.

var = 10; - переменной var присвоился тип int;

fvar = 6.5; - переменной fvar присвоился тип float;

fvar = var; - переменной fvar присвоился тип int.

Динамическое связывание типов

Примеры языков со статическим связыванием: C++, Java

Примеры языков с динамическим связыванием: Python, Ruby, JavaScript, PHP

Примеры на языке Python.

list = [4.5, 6.8, 7.4] – list – это список (массив) вещественных чисел;

list = 25; - list - это целочисленная переменная.

В языках с динамической типизацией все переменные являются фактически ссылками на объекты данных, т.е. любая переменная может ссылаться на любой объект.

Динамическое связывание типов

Основные недостатки:

- снижение надежности программ – неправильные типы на правой стороне оператора присваивания (=) не будут распознаны как ошибки; вместо этого тип переменной на левой стороне изменится на неправильный тип;
- повышенные затраты памяти – каждая переменная несет в себе сложный дескриптор; для хранения любой переменной задействуется область памяти переменного размера, поскольку формат сохраняемого значения меняется от типа к типу;
- снижение скорости вычислений – чистая интерпретация выполняется, как правило, в 10 раз медленнее, чем эквивалентный машинный код (компиляция).

Явные приведения типа в языке C++

Современная версия языка C++ предлагает четыре специальных оператора для выполнения явного приведения типа:

- static_cast;
- const_cast;
- reinterpret_cast;
- dynamic_cast.

Оператор static_cast

Оператор static_cast обеспечивает обычные статические приведения типов, а также преобразования, которые компилятор не способен выполнить автоматически.

Пример.

```
float fvar = 25.0;
int ivar = static_cast<int>(fvar);
```

Приведение такого типа обычно используется при присвоении значения большего формата переменной меньшего формата.

В этом случае при использовании неявного преобразования не исключена потеря точности результата, о чем компилятор выдаст соответствующее предупреждение.

При использовании явного преобразования предупреждения компилятора не будет.

Оператор static_cast

Оператор static_cast также обеспечивает преобразования, которые компилятор не способен выполнить автоматически.

Пример. Возращение значения переменной, адрес которой был сохранен в указателе типа void.

```
double dvar = 254.0;
void* ptr = &dvar; // сохраняем в указателе на void адрес переменной типа double
double dd = *ptr; // указатель на void разыменовывать нельзя
double* dptr = static_cast<double*>(ptr); //преобразуем указатель на void в указатель на double
double ddd = *dptr; // указатель на double разыменовывать можно
```

Оператор const_cast

Оператор const_cast преобразует тип операнда с ключевым словом const в аналогичный тип, но без ключевого слова const. При этом все остальные атрибуты «типа-источника» и «типа-приёмника» должны совпадать.

Пример. Переменная типа Key (Key key) хранит пароль системы. В целях безопасности пользователю выдается указатель на ключ как на константу. Но пользователь хочет получить возможность изменять пароль. Для этого ему необходимо создать свой указатель (неконстантный).

```
const Key* ptr; // указатель на константу типа Key
Key* my_ptr = const_cast<Key*>(ptr); // преобразуем указатель на константу в обычный указатель
```

Теперь пользователь может делать с паролем всё, что угодно. Использование любой другой формы приведения типа в данном случае привело бы к ошибке при компиляции.

Оператор dynamic_cast

Оператор dynamic_cast применяется для преобразования указателя на объект базового класса в указатель на объект производного класса.

Класс объекта, на который будет указывать указатель, в процессе компиляции известен не всегда, поскольку в указателе на объект базового класса может храниться адрес объекта производного класса.

В результате динамического приведения типа указатель на объект базового класса заменяется указателем на объект производного класса.

Совместимость типов

В конкретном контексте многие языки программирования не требуют эквивалентности типов операндов при осуществлении операций.

Вместо этого необходимо, чтобы значения типов были совместимы с тем контекстом, в котором они появляются. Например:

- в операторе присваивания тип в правой части должен быть совместим с типом в левой части;
- типы операндов операции сложения должны быть совместимы с целым или вещественным типом;
- в операторе вызова функции типы аргументов должны быть совместимы с типами соответствующих формальных параметров.

Замечание. В разных языках программирования определения совместимости могут быть различными.

Неявные преобразования типов

Если язык программирования позволяет использовать значение одного типа в контексте, где ожидается другой тип, то в таком случае выполняется автоматическое, или неявное преобразование типов.

Различают два типа неявных преобразований типов:

- сужающее приведение - уменьшает диапазон возможных значений (float → int, double → float);
- расширяющее приведение - увеличивает диапазон возможных значений (int → float, float → double).

Расширяющее приведение безопасно практически всегда..

При сужающем приведении возможна потеря точности.

Неявные преобразования типов

Если в программе используются смешанные выражения, в которых операнды имеют разные типы, то начинают работать соглашения языка о неявном приведении типов.

Пример.

```
int ivar = 10;
float fvar = 5.4, rez;
rez = ivar * fvar; // умножаем целое число на вещественное
```

Компилятор языка C++ или Java не воспримет такой код, как ошибочный.

Однако есть языки, которые не скомпилировали бы такой код.

Примеры.

- в языке Ada есть ограничения на неявные преобразования типов (запрещено смешивать целые и вещественные числа);
- в языках ML и F# неявные преобразования типов запрещены (все преобразования типов должны быть явными).

Неявные преобразования типов

С точки зрения проектирования языка вопрос формулируется следующим образом: кто должен находить в выражениях ошибки типизации – компилятор или программист ?

Поэтому неявное приведение типов – спорный вопрос в проектировании языка.

Основной аргумент противников неявного преобразования – снижение надежности программ.

Основной аргумент сторонников неявного преобразования – повышение гибкости ПО.

Наиболее современные языки программирования отражают тенденцию перехода к статической типизации и отказа от неявного приведения типов.

Язык C++, напротив, предлагает предельно богатый, в том числе, расширяемый программистом (с помощью всевозможных перегрузок), набор правил неявного приведения типов. Однако этот функционал традиционно относится к наиболее трудным для понимания понятиям.

Уровень типизации языка

С точки зрения безопасности выделяют два типа языков:

- сильно типизированные языки – языки, в которых каждая операция безопасна в отношении типа. Сильная система типизации разрешает только безопасные выражения, которые гарантируют вычисления без ошибок типизации;
- слабо типизированные языки – языки, система типизации которых не является сильной.

Основная проблема. Сильная система типизации будет разрешать только некоторое подмножество безопасных программ и отвергать очень многие программы.

Патологический пример – система типизации, которая будет отвергать все программы. Такая система типизации будет сильной, но бесполезной.

Поэтому большинство языков ищут разумный компромисс.

Уровень типизации языка

Сильно типизированные языки: Pascal, Ada, ML, F#

Слабо типизированные языки: C, C++

Относительно сильно типизированные языки: Java, C#

Пример слабо типизированного кода на языке C++.

```
int ivar = 10; // целочисленная переменная
char ch = '*'; // символ
bool bb = false; // булева переменная
int* ip_1 = NULL; // указатель на тип int
int* ip_2 = &ivar; // указатель на тип int
float rez; // вещественная переменная

rez = (ip_2 - ip_1 + ivar) / (ch + bb);
```

Компилятор всё пропустит !