Sem 1.01.3 Простые типы данных

Презентация к семинару в рамках курса
«Программирование на Си»
Подготовил Кострицкий А. С.

Кафедра ИУ-7
МГТУ имени Н. Э. Баумана
Москва, Россия

Москва — 2023 — TS2302162127

Определение

*Будем называть **типом данных** пару из множества значений и операций над ними.*

Определение

*Будем называть тип данных **простым**, если никакая сущность данного типа не может включать в себя сущность простого типа.*

Тип `short` используется для хранения коротких целых.
Так как мы опираемся на фактическое равенство машинного слова четырём или восьми байтам в большинстве ситуаций, то тип `short` использовать будем редко — все значения всё равно будут приводиться к машинному слову.
По стандарту *ограничен снизу* — в стандарте указано лишь, что диапазона `short` достаточно для хранения определённой константы, из чего можно сделать вывод:

$$\text{sizeof}(\textit{short}) \geqslant 2b.$$

Множество операций: сложение, вычитание, умножение, деление нацело, остаток от деления, и так далее. Распространённый спецификатор ввода-вывода: `%sd`

Тип `long` используется, для хранения целых, которые на момент создания были *длинными* по сравнению с остальными — от четырёх байт.
По стандарту *ограничен снизу* — в стандарте указано лишь, что диапазона `long` достаточно для хранения определённой константы, из чего можно сделать вывод:

$$\text{sizeof}(long) \geqslant 4b.$$

Множество операций: сложение, вычитание, умножение, деление нацело, остаток от деления, и так далее. Распространённый спецификатор ввода-вывода: `%ld`

Тип `int` является наиболее приближенным к конкретной машине. Размер в стандарте не указан. Единственный вывод о размере:

$$\text{sizeof}(\textit{short}) \leqslant \text{sizeof}(\textit{int}) \leqslant \text{sizeof}(\textit{long}).$$

Множество операций: сложение, вычитание, умножение, деление нацело, остаток от деления, и так далее. Распространённый спецификатор ввода-вывода: %d

Тип `long long` используется для хранения сверхдлинных целых. По стандарту *ограничен снизу* — в стандарте указано лишь, что диапазона `long long` достаточно для хранения определённой константы, из чего можно сделать вывод:

$$\text{sizeof}(long) \geqslant 8b.$$

Множество операций: сложение, вычитание, умножение, деление нацело, остаток от деления, и так далее. Распространённый спецификатор ввода-вывода: `%lld`

Тип `char` включает в себя символы и будет изучаться Вами перед изучением строк. Сейчас мы обращаем на него внимание, потому что это минимальный целый тип в языке Си — он всегда весит ровно один байт и часто используется для буферных переменных.
Множество операций: сложение, вычитание, умножение, деление нацело, остаток от деления, плюс все операции над символами...
Распространённый спецификатор ввода-вывода: %c

Все *беззнаковые* (`unsigned`) варианты вышеперечисленных типов. На данный момент они становятся нишевыми — на обычных, наших ПК выигрыш в увеличении диапазона в два раза спереди не покрывает проигрыш от снижения поддерживаемости программ и библиотек с беззнаковыми из-за тонкостей приведения знаковых к беззнаковым и обратно. Например, из Java беззнаковые вырезаны изначально.

Если и правда в задаче понадобится диапазон побольше — будем использовать `long long`, 2^{63} хватит на всё.

Все целые типы замкнуты относительно операций деления нацело и взятия остатка.
Относительно умножения, сложения и вычитания замкнуты лишь беззнаковые — переполнение беззнаковых происходит по кольцу (*одновременно делая сложение, например, уже не совсем тем самым сложением*), переполнение знаковых является в Си ситуацией с неопределённым поведением (undefined behaviour).

Договоримся, что пока не сказано обратное, мы считаем, что переполнения в программе быть не может (обеспечено входными данными). Если бы мы поступали формально, пришлось бы проверять на возможное переполнение каждое целочисленное сложение.

Тип `size_t` (*беззнаковый*) используется для адресных переменных, потому что он никогда не может превышать размер адреса, достаточного для адресации доступной оперативной памяти на машине с учётом архитектуры ОС
Фактически не существует — является макросом, поэтому при попытке принять с клавиатуры значение со спецификатором из стандарта возможны ложные предупреждения, что *написанный спецификатор не подходит для типов long long int*, etc. В этих случаях нужно принуждать транслятор соблюдать стандарт.

Типы целых фиксированного размера, знаковые и беззнаковые.
Названия по шаблону `[u]int[size in bites]_t`: int8_t, int32_t, uint32_t, etc.
Множество операций: сложение, вычитание, умножение, деление нацело, остаток от деления, и так далее. Макросы для спецификаторов описаны в стандарте.
Используются в ситуациях, когда переносимость программы важнее производительности.

Тут мы видим пример ***обязательности в реализации*** — по стандарту целые фиксированные типы могут не поддерживаться транслятором, но если поддерживаются, то только в том виде (с теми же названиями и спецификаторами), как описано:
These types are optional. However, if an implementation provides integer types with widths of 8, 16, 32, or 64 bits, no padding bits, and (for the signed types) that have a two's complement representation, it shall define the corresponding typedef names.

Типы `float` и `double` для обработки чисел с плавающей точкой одинарной и двойной точности соответственно. Спецификаторы: на ввод `%f` `%lf` соответственно, на вывод для значений обоих типов — `%lf`.

Множество операций: сложение, вычитание, умножение, деление, и так далее.

Типы замыкаются относительно операций за счёт введения специальных констант: `NaN`, `Inf`, `-Inf`.

Определение

*Будем называть **приведением типа** изменение типа выражения в соответствии с заранее заданным набором правил транслятора.*

Си — ***язык со статической типизацией***, тип переменной менять нельзя, но можно менять тип выражения.

Определение

***Приведение типа** по запросу программиста будем называть **явным**.*

В Си запрос явного приведения типа оформляется как желаемый тип в скобках перед выражением.

Определение

***Приведение типа**, вызванное невозможностью осуществить действие без приведения, будем называть **неявным**.*

В общем и целом неявное приведение в большинстве случаев работает прозрачно. В приоритете у транслятора минимизация потерь информации, поэтому, например, целые чаще приводятся к ЧПТ. Выражение «$4 + 5.0$» для транслятора выполнимым не является — сложение целых и действительных в Си не определено. Поэтому транслятор приведёт тип выражения «4» к `double`.

op: +	short	int	long	lli	float	double
short						
int						
long						
lli						
float						
double						

Заполните такую таблицу для четырёх операций.

Вопросы для самопроверки I

1. Внутри диапазона длинных целых целиком находится диапазон коротких целых. Почему тогда длинные целые тоже считаются простым типом?
2. В стандартной библиотеке Си нет рациональных. Как бы Вы реализовали рациональную арифметику? Являлись бы Ваши рациональные простым типом данных? Были бы Ваши рациональные замкнуты относительно каких-то операций?
3. В стандартной библиотеке Си нет чисел с фиксированной точкой. Как бы Вы реализовали их? Являлись бы Ваши ЧФТ простым типом данных? Были бы Ваши рациональные замкнуты относительно каких-то операций?

Вопросы для самопроверки II

4. В языке Си есть комплексные числа. Являются ли комплексные числа в Си простым типом данных?
5. Сохраняется ли информация при приведении типа по схеме:
   1. `long` to `int`
   2. `int` to `long`
   3. (`int` to `long`) to `int`
   4. `float` to `long`
   5. `double` to `float`
   6. (`int` to `double`) to `int`
   7. (`long` to `double`) to `long`