Работа с целыми числами

Коллоквиум №1

Кострицкий А. С.

TS1904301500

Задачи

1) Реализовать функцию, возвращающую ответ на вопрос, является ли целый положительный аргумент простым числом.

2) Реализовать функцию, возвращающую ответ на вопрос, является ли целый положительный аргумент числом Фибоначчи.

3) Реализовать функцию, возвращающую ответ на вопрос, является ли целый положительный аргумент полным квадратом.

4) Реализовать функцию, возвращающую ответ на вопрос, является ли целый положительный аргумент числом-палиндромом.

Ошибки

1) **В функции не учтены критические значения аргумента.**

   Что особенно удручает на фоне изучения именно этой темы в курсе ОПИ сейчас. При использовании функции:

```
int is_sqr(int n)
{
    int i = 0;
    while (i * i < n)
        i++;
    return i * i == n;
}
```

   с определённым аргументом, очевидно, произойдёт переполнение.

2) **Производится попытка перевернуть число целиком.**

   Что, в свою очередь, является частным случаем предыдущей ошибки. Число 1 000 000 003 умещается в `long`, его реверс не умещается.

3) **Привязка к архитектуре. Предположение, которое не продиктовано спецификацией.**

   Особенно это касается типа `int`. Размер этого типа спецификацией не определён. С чего Вы взяли, что в нём *всегда* 4 байта на любой машине, под любой ОС? Чем продиктовано предположение, что максимально возможное число в нём — 2 147 483 647? Что последним числом Фиббоначи, которое умещается в `int`, будет число 45?

4) **Предположение, которое не продиктовано спецификацией. Переполнение через кольцо.**

   Вы уверены, что переполнение числа `int` происходит по типу кольца? С любым ли знаковым целым числом это работает так? С любым ли беззнаковым?

Замечания по поводу стиля программирования

В программировании, как и в любом другом ремесле, теория рождалась из практики. Любая попытка описать практический опыт и создать свод догматов либо приводит к созданию новой парадигмы[1], либо к созданию нового языка[2], либо к созданию новой спецификации[3].

Обычно эти догматы представляют собой некоторые запреты на использование языковых конструкций: например, запрет использования оператора `goto`, единственность точки выхода, запрет использования слов `break, continue`, и так далее; или, наоборот, какие-либо общие советы по построению программы: гарантированное копирование сущности при изменении, передача всех аргументов в качестве констант, и так далее.

[1] Как, например, парадигма *структурного программирования*.

[2] Если хочется примеров, можно почитать истории создания языков Java, Haskell, Pascal, Ada.

[3] Причём тут полезно будет почитать про SEI CERT C Coding Standard. Я хотел бы назвать ещё один стандарт C, в котором запрещены все целые, кроме целых с явным указанием размера, но, к сожалению, вспомнить не могу.

И, конечно, *из каждого правила можно найти исключение*,

если уметь аргументировать свою позицию.

То есть, все описанные ниже ситуации возможны, совершенное студентом в каждом примере действие не являлось бы ошибочным, если бы студент сумел показать невозможность иного подхода. Сразу стоит сказать, что следующие фразы аргументами не являются:

1) Меня так учили.
2) Меня так научили.
3) Я слышал.
4) Я видел.
5) Я пробовал.
6) Я читал.
7) Оно же работает!
8) Оно же транслируется!
9) В другой ситуации...
10) На Хабре...
11) В C++...
12) В задании не написано!
13) Мне так удобнее.
14) Я проверил: *на этой машине* всё работает.
15) Я проверял: *на всех моих машинах* всё работает.
16) Всегда так делали!

Ну а теперь сами примеры:

1) **Магические числа.**

   В программе не должно быть констант, о смысле которых приходится догадываться. Крайне желательно использовать именованные константы; если какое-то значение привязано к архитектуре — использовать макроопределения.

2) **В имени функции указана информация, ясная из сигнатуры.**

   Совершенно правильной кажется мысль, что при первом взгляде на функцию должно быть понятно, что она делает. Подавляющее большинство сред разработки сейчас уже могут предоставлять информацию о сигнатуре при наведении на любую точку вызова, поэтому нет смысла выносить в имя функции данные о аргументах.

   Сравните:

```
long is_int_number_prime(long n); // not good naming
long is_prime(long n); // good naming
```

Иногда студента не останавливают увещевания преподавателя, и привычка так писать входит в терминальную стадию:

```
int try_create_subarray_from_collec_primed(array_int__onedim_t array, int arr_lengt
```

Надо признать, что тут могут быть исключения. С относится к языкам без перегрузки функций[4], поэтому при написании, например, двух наборов функций для двух разных структур данных придётся что-то выносить в имя:

```
int scan_matrix(matrix_t m, size_t crow, size_t ccol);
int print_matrix(matrix_t m, size_t crow, size_t ccol);
int scan_array(array_t a, size_t len);
int print_array(array_t a, size_t len);
```

3) **В имени функции есть транслитерация.**

Не думаю, что Вы станете доверять автору справочника, в котором есть функции:

```
int sortirovochka_left(array_t arr, size_t dlina);
int vvod_massiva(array_t arr, size_t dlina);
int vivod_massiva(array_t arr, size_t dlina);
```

4) **Функция не возвращает уже полученные данные.**

Рассмотрим одну функцию:

```
int is_sqr(int n)
{
    int i = 0;
    int j = 1;
    while (n > 0)
    {
        n -= j;
        i++;
        j += 2;
    }
    return n == 0;
}
```

Внутри уже вычисляется целый корень из числа, причём он входит в выходное множество, так почему бы его и не возвращать? Например:

[4]Хотя со стандартом C11 были введены *дженерики* (type-generic expressions) — одна из форм перегрузки функций, в данном случае их мы не рассматриваем.

```
int my_sqrti(int n)
{// Функция возвращает целый корень, если аргумент
 // является полным квадратом, и нуль otherwise
    int i = 0;
    int j = 1;
    while (n > 0)
    {
        n -= j;
        i++;
        j += 2;
     }
    if (!n)
        return i;
     else
        return 0;
}
```

5) **Внутри вычислительной функции есть операции ввода-вывода.**

   Всё очень просто — *внутри «вычислительных» функций не должно быть операций ввода-вывода.* Ваша подпрограмма может работать как часть программы на сервере, Ваша подпрограмма может стать частью чего-то большего, где нет привычного Вам терминала, ввод-вывод на целевой машине может быть организован не совсем тривиальным образом, etc. Просто старайтесь ввод-вывод целиком реализовать либо внутри главной программы, либо внутри подпрограмм ввода-вывода.

6) **Внутри функции, возвращающей ответ на вопрос «Обладает ли целый аргумент свойством %СВОЙСТВО%?», использован переход в другой тип данных, зачастую `double`.**

   Это очень нехорошая практика. Во-первых, привычка переходить в другой тип данных потом может сыграть с Вами злую шутку, без должной аккуратности можно начать делать преобразования между в принципе различными сущностями. Во-вторых, не на всех машинах ***аппаратно*** реализована арифметика чисел с точкой — транслятор будет транслировать текст программы в абсолютно правильный код в силу спецификации, который будет исполняться очень и очень неоптимально. В-третьих, Вы уверены, что Ваше преобразование «`int to double to int`» не теряет точность? Чем обусловлена Ваша уверенность? А если «`long long to float to long long`»? А если вычислять число Фибоначчи с помощью формулы Бине?

   Поэтому постарайтесь *не использовать внутри подпрограмм для работы с целыми числами явные или неявные приведения к числам с плавающей точкой.*