Правила написания программ на языке Си

13 февраля 2024 г.

1. Введение

Настоящий документ содержит требования, предъявляемые к программам на языке Си, сдаваемым студентами II курса кафедры ИУ-1 МВТУ в рамках курсов «Информатика» и «Практикум на ЭВМ». Требования предъявляются к трём аспектам программ: переносимости, структуре и исходному тексту. С точки зрения сдачи преподавателям (и, следовательно, получения оценки за курс) программы, не удовлетворяющие настоящему документу, приравниваются к ненаписанным.

В рамках этого документа принимаются следующие обозначения.

1. Некоторые требования содержат фразу «только при наличии веского обоснования». Это означает, что использование того или иной приёма считается допустимым только если при сдаче программы будет представлено удовлетворительное объяснение (неформальное), почему без этого приёма в данном конкретном случае обойтись нельзя.
2. В примерах фрагменты кода, не противоречащие требованиям, называются *допустимыми*.
3. В примерах кода допустимый код помещен в чёрную рамку, недопустимый — в красную.
4. В примерах выражение `/* ... */` означает пропуск нерелевантной части кода.

В качестве дальнейшего чтения по вопросам правил и подходов, помогающих в написании кода, можно порекомендовать работы [1], [2, главы 12–18], [3, главы 7,10–13,17,19,31,32].

Наконец, напомним важное правило:

Исходный текст программ предназначен
в первую очередь для понимания человеком.

2. Правила написания программ

2.1. Требования переносимости

1. Программа должна удовлетворять стандарту C99 [4].
2. Программа не должна использовать языковые расширения, предоставляемые конкретным компилятором.
3. Программа не должна полагаться на механизмы, предоставляемые каким-либо конкретным семействам ОС.
4. Программа не должна полагаться на особенности реализации языка конкретным компилятором. В частности:
   (a) Числовые типы используемых переменных должны выбираться так, чтобы гарантированно вмещать параметры задачи. Для явного отражения вместимости рекомендуется использовать типы фиксированного размера (определённые в `stdint.h`).
   Пример. Если по условию задачи некоторая величина может принимать значения от 0 до 100 000, то для её хранения недопустимо использовать тип `int`, поскольку

его вместимость гарантируется стандартом лишь до $2^{15} - 1 = 32\,767$. Здесь стоит использовать тип `uint32_t`.

2.2. Требования к тексту

2.2.1. Символы

1. Файл исходного кода должен быть в кодировке UTF8.
2. Символы, отличные то базового набора ASCII (коды 0–127), допускаются только в комментариях.
3. Ширина строки кода не должна превышать 80 символов.

2.2.2. Выражения и строки

1. На каждой строке кода может располагаться не более одного оператора. (Это требование можно понимать так: не более одного действия на строку.)
2. Строки с закрывающими операторными скобками (`}`) не могут содержать никаких других операторов.
3. Допускается объявление нескольких переменных в одной строчке, если они одного типа.
4. Инициализация составных типов должна быть поэлементной.
   Пример. Фрагмент слева является допустимым, справа — **недопустимым**.

```
int matrix[3][3] = {
  {1, 0, 0},
  {0, 1, 0},
  {0, 0, 1}
};
```

```
int matrix[3][3] =
   {1,0,0,0,1,0,0,0,1};
```

2.2.3. Отступы и пробелы

1. Допускается использование в роли отступов двух пробелов, четырёх пробелов или символа табуляции.
2. В программе допускается использование только одного вида отступов.
3. Строка с открывающей операторной скобкой должны быть на том же уровне вложенности, что и конструкция, порождающая блок (объявление функции, цикла или типа).
4. Строка с закрывающей операторной скобкой должна быть на том же уровне вложенности, что и строка с открывающей.
   Пример. Следующие два фрагмента кода являются допустимыми.

```
int parse(const char * str){
  /* ... */
}
```

```
int parse(const char * str)
{
  /* ... */
}
```

5. Программа должна использовать только один способ расположения открывающих скобок: либо на той же строке, что и порождающая блок конструкция, либо на следующей.
6. Открывающая и закрывающая скобки не могут находиться на одной строке.
7. Все строчки между двумя парными операторными скобками должны быть на один уровень вложенности выше, чем эти скобки.
8. Бинарные операторы (+, *, ...) должны быть отделены пробелами от аргументов.
   Примечание. Операции доступа к полю (., ->) бинарными операциями не являются.

9. Унарные операторы пробелами не отделяются.
10. После запятой всегда ставится пробел.
11. После точки с запятой в заголовке цикла `for` ставится пробел.
12. Метки веток (`case`, `default`}) оператора выбора должны иметь глубину отступа на единицу выше, оператора. На строчке с меткой ничего, кроме самой метки, быть не должно. Все строчки блока метки, включая оператор `break`, должны иметь глубину отступа на единицу выше, чем метка.
    Пример. Следующие фрагменты кода являются допустимыми.

```
switch(op)
{
  case '+':
    c = a + b;
    break;
  case '-':
    c = a - b;
    break;
  default:
    assert(0);
}
```

```
switch(op)
{
case '+':
  c = a + b;
  break;
case '-':
  c = a - b;
  break;
default:
  assert(0);
}
```

13. В случае, когда каждая ветка `case` состоит из одного короткого выражения с оператором `return`, то допускается упрощённая запись, в которой метка и `return` располагаются на одной строчке.
    Пример. Следующий фрагмент кода является допустимым.

```
const char* state_str(enum State state){
  switch(state)
  {
  case STATE_IDLE: return "idle";
  case STATE_WORK: return "work";
  case STATE_DONE: return "done";
  }
  assert(0);
  return 0;
}
```

14. В случае, когда последовательность вложенных операторов выбора является взаимоисключающей[1], каждый последующий `if` должен располагаться на той же строчке, что и `else`; при этом не должны использоваться операторные скобки и глубина вложенности не увеличивается.
    Пример. Следующий фрагмент кода является допустимым.

```
if (strcmp(cmd, "start") == 0)
{
  /* ... */
}
else if (strcmp(cmd, "stop") == 0)
```

[1] Т.е. когда эта последовательность играет роль конструкции `switch` с более сложной логикой в каждом `case`.

```
{
  /* ... */
}
else if (strcmp(cmd, "next") == 0)
{
  /* ... */
}
else if (strcmp(cmd, "previous") == 0)
{
  /* ... */
}
else /* unknown command */
{
  /* ... */
}
```

Следующий фрагмент кода является **недопустимым**.

```
if (strcmp(cmd, "start") == 0)
{
  /* ... */
}
else
{
  if (strcmp(cmd, "stop") == 0)
  {
    /* ... */
  }
  else
  {
    if (strcmp(cmd, "next") == 0)
    {
      /* ... */
    }
    else
    {
      if (strcmp(cmd, "previous") == 0)
      {
        /* ... */
      }
      else /* unknown command */
      {
        /* ... */
      }
    }
  }
}
```

2.2.4. Разбиение на несколько строк

15. Разбиение длинного строкового литерала на несколько строк должно осуществляться только через конкатенацию строковых литералов; при этом вторая и последующие строчки должны быть расположены на одном уровне вложенности больше, чем первая.
Пример. Пример слева является допустимым, справа — **недопустимым**.

```
const char * str = "The "
  "quick brown fox jumps "
  "over the lazy dog.";
```

```
const char * str = "The\
quick brown fox jumps\
over the lazy dog.";
```

16. При необходимости разбиения вызова функции или её объявления на несколько строк, каждый параметр должен располагаться на отдельной строке; строки, начиная со второй, должны иметь отступ на единицу выше строки вызова (объявления).
Пример. Следующие фрагменты кода являются допустимыми.

```
readCfgParams(file,
  &cfg,
  errorHandler);
```

```
readCfgParams(
  file,
  &cfg,
  errorHandler
);
```

17. При разбиении выражения на несколько строк каждая строка, начиная со второй, должна иметь глубину отступа на единицу выше первой.
18. При разбиении строки с объявлением цикла `for` на несколько каждое выражение должно быть на отдельной строке, причём вторая и последующая строки должны иметь глубины отступа на единицу выше.
Пример. Следующие фрагменты кода являются допустимым.

```
Node * node;
for(node = &list;
  node != 0;
  node = node->next)
{
  /* ... */
}
```

```
Node * node;
for(
  node = &list;
  node != 0;
  node = node->next
){
  /* ... */
}
```

2.3. Требования к структуре программы

2.3.1. Состав программы

1. Программа должна представлять собой набор взаимосвязанных функций.
2. Ответ на вопрос, что делает та или иная функция, должен состоять из одной простой фразы, причём из неё должен быть понятен смысл аргументов и возвращаемого значения функции. Если эта фраза не очевидна из сигнатуры функции, то рекомендуется поместить её в комментарий перед функцией.
3. Код каждой функции должен быть понятен без знания кода, эту функцию вызывающего.
4. Код, вызывающий функцию, должен быть понятен без знания кода вызываемой функции.
5. Не рекомендуется давать функциям более трёх параметров.

6. Использование у функции шести и более параметров допустимо только при наличии веского обоснования.
7. Не рекомендуется делать функции длиннее 30–50 строк.
8. Использование функций длинной 100 строк и более возможно только при наличии веского обоснования.

2.3.2. Выбор имён

1. Имена (идентификаторы) должны выбираться в соответствии с тем, для чего они используются.
2. Имя переменной не должно содержать указания её типа.
   Примечание. Указание единицы измерения является допустимым, например, `distance_km`, `angle_deg`.
3. Имена переменных и функций должны начинаться с строчной буквы.
4. Имена переменных и функций могут быть либо в `camelCase`, либо в `snake_case`, при этом во всей программе для этого должна использоваться только одна схема именования.
5. Имена макросов должны содержать буквы только в `ВЕРХНЕМ РЕГИСТРЕ`.
6. При использовании слова естественного языка как части имени допустимо использование только слов английского языка и сокращений от них.
   Пример. Следующие имена являются допустимыми: `my_number`, `num_elements`, `value`, `is_off`.
   Следующие имена являются **недопустимыми**: `mnu_chiselko`, `imya_faila`, `kol_elementov`, `znach`, `is_vykl`.

2.3.3. Языковые конструкции

1. Использование глобальных переменных возможно только при наличии веского обоснования.
2. Использование в выражении трёх и более побочных эффектов возможно только при наличии веского обоснования.
3. Запрещено использование более одного побочного эффекта в логических выражениях.
4. Использование вложенности блоков уровня четыре и выше возможно только при наличии веского обоснования.
   Пример. В следующем фрагменте уровень вложенности указан в комментарии в скобках.

```
int has_negative(const struct Node * list)
{
  struct Node * node = list; /* (1) */
  while (node != 0)          /* (1) */
  {
    const int * data = node->data;        /* (2) */
    int k;                                /* (2) */
    for (k = 0; k < data->length; k++)    /* (2) */
    {
      if (data->arr[k] < 0) /* (3) */
      {
        return 1;       /* (4) */
      }
    }
  }
  return 0;                   /* (1) */
```

```
}
```

5. Структуры и объединения могут передаваться в функцию по значению только при наличии веского обоснования.
6. Если в функцию передаётся указатель на переменную, значение которой функция не меняет, то у соответствующего аргумента должен быть квалификатор `const`.
7. Переменные должны объявляться (и, если возможно, инициализироваться) как можно ближе к месту своего первого использования.
   Пример. Пример слева является допустимым, справа — **недопустимым**.

```
{
  /* ... */
  int k;
  for (k = 0; k < MAX_ELEMS; k++)
  {
    /* ... */
  }
}
```

```
{
  int k = 0;
  /* ... */
  for (; k < MAX_ELEMS; k++)
  {
    /* ... */
  }
}
```

8. Внутри тела цикла `for` запрещено менять значение счётчика цикла.
9. Использование оператора безусловного перехода (`goto`) допустимо только в следующих случаях.
   (a) Для досрочного выхода из двойных, тройных и более вложенных циклов.
   Пример.

```
  int i, j, k, found = 0;
  for (i = 0; i < tensor->size[0]; i++){
    for (j = 0; j < tensor->size[1]; j++){
      for (k = 0; k < tensor->size[2]; k++){
        if (tensor.data[i][j][k] < 0){
          found = 1;
          goto done;
        }
      }
    }
  }
done:
  if (found){
    printf("%d %d %d\n", i, j, k);
  }
  else{
    printf("no negative elements\n");
  }
```

   (b) Для организации освобождения выделенных в начале блока ресурсов (памяти, файлов, ...).
   Пример.

```
int process(const char * file_name)
{
    int success = 1;
    FILE * fp = fopen(file_name);
```

```
    /* ... */
    if (bad_data)
    {
        success = 0;
        goto finalize;
    }
    /* ... */
finalize:
    fclose(fp);
    return success;
}
```

10. Не допускается использование числовых констант, отличных от 0 и 1. Любая числовая константа должна быть объявлена (например, через `#define`) и иметь имя, отражающее её применение.
11. Запрещается использовать локальные массивы переменного размера (определяемого на этапе выполнения, VLA).
12. Запрещается использовать цикл `for` для ситуаций, когда к моменту начала его выполнения не известно максимальное число итераций.
    Пример. В примере слева осуществляется обход односвязного списка. Число итераций ограничено количеством элементов списка (фактическое число может быть меньше из-за операторов `break` и `return`). Этот пример является допустимым.
    В примере справа в теле цикла осуществляется операция ввода, и максимальное число итераций в цикле может быть любым (даже бесконечным). Этот пример является **недопустимым**. В таких ситуациях надо использовать цикл `while`.

```
for (
  const Node* q = list->head;
  q != 0;
  q = q->next)
{
  /* ... */
}
```

```
for (int ch;
(ch = getchar()) != EOF;)
{
  /* ... */
}
```

2.3.4. Комментарии

1. Комментарии должны использоваться для описания вещей, не очевидных непосредственно из кода.
   Пример. При объявлении функции комментарии могут описывать следующие аспекты её аргументов:
   - Единица измерения, если она есть (пиксель, байт, килобайт,...).
   - Свойства, которые обязаны выполняться для корректной работы функции. (Например, ограничения на диапазон принимаемых значений для скалярных типов; для указателей — допустимость нулевого значения; и так далее.)
2. Комментарии не должны дублировать код.

Список литературы

[1] Столяров А.В. Оформление программного кода. Макс-Пресс, Москва, 2019.

[2] Ousterhaut J. A philosophy of software design. Yaknyam Press, Palo Alto, 2018.

[3] McConnel S. Code complete. Microsoft Press, 2004. Перевод: Макконел С. Совершенный код. Русская редакция, Москва, 2010.

[4] Стандарт ISO/IEC 9899:1999. Ссылка на PDF