ВВЕДЕНИЕ В ЯЗЫК C

Трансляционная модель языка, поведение программ, ISO стандарт

К. Владимиров, Yadro, 2024
mail-to: konstantin.vladimirov@gmail.com

Обсуждение

- К этому моменту мы уже около года изучаем программирование на C
- Но что такое язык программирования вообще?
- Как вы по некоему исходному тексту определите является ли он вообще программой на C?

http://uguu.org/src_fuuko_c.html

```
                                #define r(R) R"()"
                              /*[*/#include  /**/<stdio.h>
                            #include<math.h>/*!![crc=0f527cd2]*/
                          float I,bu,k,i,F,u,U,K,O;char o[5200];int
                       #define R(U) (sizeof('U')==1||sizeof(U"1"[0])==1)
                      h=0,t=-1,m=80,n=26,d,g,p=0,q=0,v=0,y=112,x=40;  float
                     N(float/*x*/_){g=1<<30;d=-~d*1103515245&--g;return  d*_
                    /g;}void/**/w(int/**/_){if(t<0){for(g=0;g<5200;o[g++   ]=
                    0);for(;g;o[g+79]=10)g-=80;for(t=37;g<62;o[80+g++]=32)   ;
                   }if(m&&o[h*80+m-1]==10){for(g=0;g<79;o[t*80+g++]=0){}o[t
                   ++*80+g]=10;t%=64;n+=2;I=N(70)+5;if(n>30&&(I-x)*(I-x)+n*
                  n>1600&&R()){O=0;F=(x=0x1!=sizeof(' '))?k=1+N(2),i=12-k+N(
                  8),N(4):(k=17+N(5),i=0,r()[0]?O=.1:  0);for(u=U=-.05;u<32;
                  U=k+i+i*.5*sin((u+=.05)+F))for( K=0   ;K< U;K+=.1)if((bu=K*
                 sin(u/5),g=I+cos( u/5) *K)>=0&&g  <     79  )o[g+(int)(t+44+
                 bu*(.5-(bu>0?3*O:  O)   ) )%64*  80      ]  =32;x*=02//* */2
                -1;n=O+x?n=I+(x?0   :N     (k)-   k           /2),g=(t+42  )%
                64,m=-~g%64,x?g=m          =-~        m%64:0  ,n>5?o[g*80   +
               n-3]=o[m*80+n-3]=       0:   0             ,n <75?o[g*80+n
               +2]=o[m*80+n+2]=0   :0:0;                     x=I;}h=-~h%64
              ;m=0;}putchar((g=o [h*                         80+m++])?g:_);
             if(g){w(_);}}void W                              (const char*_
            ){for(;*_;w(*_++));}                              int main(int a
            ,char**_){while(a--)d              +=_[a          ]-(char*)0;W( \
           "#include<stdio.h>typed"              "e"         "f\40int\40O;v"
           "oid o(O _){putchar(_);}O"                     "\40main(){O"  ""
          "*_[512],**p=_,**d,b,q;for(b=0;b"        "++<512;p=_+q)_[q"    \
          "=(p-_+1)*9%512]=(O*)p;") ;      for(;(g= getchar())-EOF;p=
          q){q=p;for(v=512;p-q-g&&q-p-               g;  v--)q=-~q*9%512
          ;W("o(");if(p>q)w(y),w(45);w(                      40);w(y^=20
          );w(075);for(a=0;a<v;a++)w(42);                      for(W("(O**"
           );a--;w(42)){}w(41);w(y^024);w(                      41);if(p<=q)w(
            45),w(y^20);W(");");}for(a=7;a-6                       ;W(a<6?"{;}":""
               ))for(a  =0;a  <6 &&   !o[h*80+m                        +a];a++){}W("r"
                  "etu"  /*J   */       "rn+0;}\n"                             );return
                      /*                   "#*/0                                    ;}
```

Небольшой опыт стандартизации

- Язык INC содержит четыре переменных: a, b, c, d, символы +, post ++, = и ; а также константа 0. Каждую можно инкрементировать и складывать с другими. В каждую можно писать результат
- Каждая запись в переменную d это вывод на экран

```
a = 0; b = a++; c = a + b++ + a; d = c; // на экране 4
```

- Как вы охарактеризуете следующие программы?

```
(1) x = 0;
(2) a = b++c;
(3) 0 = a;
```

```
(4) a = a++;
(5) a = 0; c = b = a++; c = a; d = c;
(6) a = 0; b = a++ + a++ + a++; d = a + b;
```

Задача которую решает компилятор

- Есть операционная семантика языка.

```
a = b++; // запиши в a значение b, увеличь b на один
```

- Есть возможности реальной аппаратуры.

```
mov r1, r0; // запиши в r1 значение r0
inc r0;     // увеличь r0 на 1
```

- Они не всегда точно совпадают.
- Компилятор должен пересказать программу в терминах вычислительного устройства, сохранив ожидаемое поведение (вспомните as if rule) и выполнив оптимизации.

https://godbolt.org/z/9axcj4Pso

Стандарт языка

- Язык программирования это соглашение между программистом и разработчиком компилятора.
- Как таковое, оно задокументировано в стандарте языка.
- Именно стандарт, а не конкретная реализация является последним и решающим аргументом в вопросе о том, что является программой, что нет и какая программа работает верно, какая нет.
- Действующий стандарт C это ISO/IEC 9899-2018 принятый в 2018 году International Organization for Standardization (ISO).
- Подробная информация доступна на http://www.iso-9899.info/wiki/The_Standard

История языка это история стандартов

- С 1972 по 1989 год – до-стандартный период. Стандартом де-факто была книга $[K\&R]$, поэтому иногда говорят о "K&R C language"
- принятие ISO/IEC 9899-1990, также обозначаемого C90 поддержка которого сейчас реализована во всех компиляторах всех платформ
- принятие ISO/IEC 9899-1999, также обозначаемого C99 поддержка которого широко распространена, но не везде он поддержан полностью
- принятие ISO/IEC 9899-2011, также обозначаемого C11 он полностью поддержан в gcc и без опциональных частей в clang/llvm
- принятие ISO/IEC 9899-2018, по сути технические правки в C11

Некоторые особенности C90

- Код откомментирован через /* */ так как // комментария в C90 не было.

```
#include <stdio.h>

static x; /* → static int x; */

main(void) { /* → int main(void) { */
  auto i; /* → auto int i; */

  for (i = 0; i < 5; i += 2) /* can not: for(int i = 0 ....) */
    x += i;
  printf("%d\n", x);
  return 0; /* mandatory in C90 */
}
```

Обсуждение

- По умолчанию в компиляторе gcc:
  - До gcc `5.1` был `C90`
  - Начиная с gcc `5.1` установлен `C11`
  - Начиная с gcc `8` установлен `C18`
- Везде далее я буду предполагать `--std=c11`.
- Дополнительно опция `-pedantic` позволяет чуть более педантично контролировать соответствие стандарту.
- Соответствующий стандарту код называется conforming и имеет однозначную семантику исполнения в абстрактной машине языка.

Синтаксис и семантика

- Стандарт языка это список синтаксических и семантических правил.
- Синтаксические правила можно проверить в грамматике языка.

```
int a = +; // syntax violation
```

- Семантические иногда сложно проверить.

```
int foo(int); // foo должна быть где-то определена

int bar(int x) {
  return foo(x); // и вызвана с правильным числом аргументов
}
```

- Реализация в идеале должна транслировать программу либо выдать диагностику.

Поведение программы

- Синтаксически некорректные
- Синтаксически корректные
  - strictly conforming behavior [C11 4.5]
  - implementation-defined behavior
  - unspecified behavior
  - undefined behavior
- Implementation limits (пределы гарантированной поддержки).
- Hosted и Freestanding environment (может ли компилятор догадываться о функциях стандартной библиотеки).

https://godbolt.org/z/17WY8Khf7

Implementation limits

- 127 nesting levels of blocks.
- 63 nesting levels of conditional inclusion.
- 12 pointer, array, and function declarators (in any combinations) modifying an arithmetic, structure, union, or **void** type in a declaration.
- 63 nesting levels of parenthesized declarators within a full declarator.
- 63 nesting levels of parenthesized expressions within a full expression.
- 63 significant initial characters in an internal identifier or a macro name.
- 31 significant initial characters in an external identifier.
- И так далее.

https://godbolt.org/z/KdPdzY6Yx

Implementation defined

- Простейший пример это размер типа int.

```
int main(void) {
  printf("%d\n", sizeof(int)); // → ?
}
```

- Попробуйте это на своём любимом компиляторе.
- Скорее всего тут будет 4.
- Но формально тут может быть что-то другое и этот код уже не strictly conforming.

Unspecified

- Порядок вызова функций без установленных отношений последования.

```
#include <stdio.h>
int foo() { printf("%s\n", "foo"); return 0; }
int bar() { printf("%s\n", "bar"); return 1; }
void buz(int x, int y) { printf("%s %d %d\n", "buz", x, y); }
int main() { buz(foo(), bar()); }
```

- Попробуйте это под разными уровнями оптимизации и на разных компиляторах. Здесь может быть абсолютно разный порядок печати bar и foo.

https://godbolt.org/z/sd6s9vGed

Undefined

- NULL pointer dereference.

```
int deref(int* a) {
  return *a; // UB if NULL pointer dereference
}
```

- Более забавный случай -- целочисленное переполнение.

```
int mult(int a, int b) {
  return a * b; // UB if integer overflow
}
```

- Звучит странно, но если здесь результат не влезает в целочисленный тип, компилятор имеет право сделать его каким угодно.

Свобода бездействия

- На всех известных мне компиляторах эта программа выведет 42

```
#include <stdio.h>

int foo(int c) {
  int x, y;
  y = c ? x : 42; // при c == true операция y = x undefined
                  // поэтому проверка c может не проводиться
  return y;
}

int main() {
  printf("%d\n", foo(1));
}
```


https://godbolt.org/z/YovPaz87r

Undefined: tricky case

- Чуть более сложный пример.

```
int k, satd = 0, dd, d[16];
/* .... more code here .... */
for (dd = d[k = 0]; k < 16; dd = d[++k])
  satd += (dd < 0 ? -dd : dd);
```

- Как вы думаете, сколько итераций может исполняться этот цикл?

Undefined: tricky case

- Чуть более сложный пример

```
int k, satd = 0, dd, d[16];
/* .... more code here .... */
for (dd = d[k = 0]; k < 16; dd = d[++k])
  satd += (dd < 0 ? -dd : dd);
```

- Как вы думаете, сколько итераций может исполняться этот цикл?
- Правильный ответ: компилятор имеет право сделать этот цикл бесконечным.
- Важный урок здесь в том, что UB может быть неявным и компилятор может как сделать так и не сделать что угодно.

Точка зрения компилятора

- Неопределённое поведение это простор для оптимизаций

```
int elt = arr[idx]; // выход за границы массива это UB
                    // значит мы его не учитываем
```

- Видите логику?

```
y = c ? x : 42;     // возможность, что c == true это UB
                    // значит мы её не рассматриваем
```

- Удивительно, но компилятор **сознательно** слеп в отношении UB
- Мы следуем так называемому "узкому контракту".

Undefined: исправляем ситуацию

- Предположим, что вам предложили доработать следующий простой код, чтобы избежать UB.

```
int mult(int a, int b) {
  return a * b; // UB if integer overflow
}
```

- Как вы это сделаете?

Упражнение в чтении стандарта

- Внимательно прочитайте пункт 6.5.7 стандарта C11 и охарактеризуйте семантику выполнения следующих конструкций.

```
int a = -2, b = 2;

int v0 = a << 3;  // #1
int v1 = a << 38; // #2
int v2 = a >> 3;  // #3
int v3 = a >> 38; // #4
int v4 = b >> 3;  // #5
int v5 = b >> -3; // #6
int v6 = b << 3;  // #7
int v7 = b << 38; // #8
```

Undefined: чего опасаться

- Целочисленное знаковое переполнение.
- Деление на ноль.
- Некоторые случаи сдвигов (см. слайдом ранее).
- Приведение целого числа к слишком узкому типу.
- Попытка изменить константный объект приведением.
- Модификация не упорядоченная по побочным эффектам (см. далее).
- Разыменование нулевого указателя.
- Обращение за границами массива.
- Обращение к неинициализированной переменной.
- Использование указателя на истекший или удалённый объект.
- Доступ к значению через несовместимый тип.
- Использование некоторых библиотечных функций (например memcpy с пересекающимися участками памяти).

Упорядочение побочных эффектов

- Ключевым в абстрактной машине языка C является понятие побочного эффекта (C11, 5.1.2.3) и отношения следования над побочными эффектами
- Примеры побочных эффектов: вывод на экран, печать в файл, сохранение в глобальную переменную
- Побочные эффекты должны быть упорядочены (отношениями последования)

```
x = 5; x = x + 1; foo(++x);
```

- Стандарт (C11, 6.5.2) гласит, что если побочный эффект на скалярный объект не упорядочен с другим побочным эффектом или со значением скалярного объекта, то это UB

```
x = x++ + ++x; // undefined
```

Стандарт как источник знаний

- Допустим вы читаете код и видите там следующую (к счастью крайне редкую) конструкцию

```
void f(double a[restrict static 3][5]); // так вообще можно???
```

- Используйте стандарт (C11, 6.7.6.3) чтобы понять что означает это объявление
- Никогда не пишите такие конструкции (например потому что они чудовищно не совместимы ни с ранними версиями C ни с C++)
- Потренируемся:

```
enum { MAX = 100; }
int arr[MAX] = { 1, 3, 5, [MAX-3] = 8, 4, 2 }; // что в arr?
```

Странности inline

- Используя компилятор gcc попробуйте откомпилировать код.

```
inline void my_assert(int b, const char *str) {
  if (!b) return;
  fprintf (stderr, "Assertion failed: %s\n", str);
  exit (-1);
}

int main(int argc, char **argv) {
  my_assert (argc > 0, "argc <= 0");
  return 0;
}
```

- С опциями (1) --std=c90 (2) --std=gnu90 (3) --std=c99

Немного о важности стандартизации

- Конкретные формулировки в стандарте очень важны.
- Пример: знаменитая memmove/memcpy saga когда внезапно оказалось, что ядро Линукс много лет закладывалось на некорректное понимание memcpy.
- Стандарт это контракт.
- Контракты пишутся людьми и для людей, но содержание их пунктов и даже мелкий шрифт имеют значение.

https://sourceware.org/bugzilla/show_bug.cgi?id=12518

Литература

- [C90] ISO/IEC – "Information technology – Programming languages – C", 1990
- [C99] ISO/IEC – "Information technology – Programming languages – C", 1999
- [C11] ISO/IEC – "Information technology – Programming languages – C", 2011
- [$CComm$] C standard commentary
- [$K\&R$] Brian W. Kernighan, Dennis Ritchie – The C programming language, 1988
- [$Linden$] Peter van der Linden – Expert C Programming: Deep C Secrets , 1994