

Mapping of programs in C language to programs in EO language

1 ВАРИАНТЫ ТРАНСФОРМАЦИИ КОНСТРУКЦИЙ ЯЗЫКА C В EO

1.1 Константы

Целочисленные константы

C:

```
1| const int constantName = 10;
```

EO:

```
1| 10 > constantName
```

Константы с плавающей точкой

C:

```
1| const double constantName = 3.14;
```

EO:

```
1| 3.14 > constantName
```

Булевские константы

C:

```
1| const _Bool constantName = 1;
```

EO:

```
1| TRUE > constantName
```

Символьные константы

C:

```
1| const char constantName = 'a';
```

EO:

```
1| "a" > constantName
```

Следует также отметить, что практически во всех случаях вместо специальных обозначений констант можно подставлять в формируемый код на EO непосредственные значения в виде литералов, что может оказаться целесообразнее.

1.2 Трансформация переменных

16-рядные целочисленные переменные

C:

```
1| short int varName;
```

EO:

```

1 +package c2eo.ctypes
2 +alias org.eolang.txt.sprintf
3 [] > c_int16
4   memory > @
5   [value] > write
6     -32768 > min
7     32767 > max
8     65536 > range
9     if. > @
10      value.greater max
11      ^.write (((value.add min).mod range).add min)
12      if.
13      value.less min
14      ^.write (((value.add 32769).mod range).add max)
15      ^.write value
16   sprintf "%d" $ > toString
17   "int16" > type
18   4 > bytes
19   32767 > max
20   -32768 > min
21
22 c_int16 > varName

```

32-разрядные целочисленные переменные

C:

```

1 int varName;
   EO:

1 +package c2eo.ctypes
2 +alias org.eolang.txt.sprintf
3 [] > c_int32
4   memory > @
5   [value] > write
6     -2147483648 > min
7     2147483647 > max
8     4294967296 > range
9     if. > @
10      value.greater max
11      ^.write (((value.add min).mod range).add min)
12      if.
13      value.less min
14      ^.write (((value.add 2147483649).mod range).add max)
15      ^.write value
16   sprintf "%d" $ > toString

```

```

17 | "int32" > type
18 | 4 > bytes
19 | -2147483648 > min
20 | 2147483647 > max
21 |
22 | c_int32 > varName
    | 64-разрядные целочисленные переменные
    | C:
1 | long long int varName;
    | EO:
1 | +package c2eo.ctypes
2 | +alias org.eolang.txt.sprintf
3 | [] > c_int64
4 |   memory > @
5 |   sprintf "%d" $ > toString
6 |   "int64" > type
7 |   8 > bytes
8 |   -9223372036854775808 > min
9 |   9223372036854775807 > max
10 |
11 | c_int64 > varName
    | 64-разрядные переменные с плавающей точкой
    | C:
1 | double varName;
    | EO:
1 | +package c2eo.ctypes
2 | +alias org.eolang.txt.sprintf
3 | [] > c_float64
4 |   memory > @
5 |   sprintf "%f" $ > toString
6 |   "float64" > type
7 |   8 > bytes
8 |
9 | c_float64 > varName
    | Булевские переменные
    | C:
1 | _Bool varName;
    | EO:
1 | +package c2eo.ctypes
2 | +alias org.eolang.txt.sprintf

```

4 .

```
3 | [] > c_bool
4 |   memory > @
5 |   sprintf "%b" $ > toString
6 |   "bool" > type
7 |
8 | c_bool > varName
```

Символьные переменные

C:

```
1 | char varName;
   EO:
1 | +package c2eo.ctypes
2 | +alias org.eolang.txt.sprintf
3 | [] > c_char
4 |   memory > @
5 |   sprintf "%c" $ > toString
6 |   "char" > type
7 |
8 | c_char > varName
```

1.3 Трансформация глобальных переменных

Объявление инициализированной глобальной переменной

C:

```
1 | int varName = 10;
   EO Ver. 1:
1 | +package c2eo.types
2 | [arg] > global
3 |   c_int32 10 > varName
   EO Ver. 2:
```

```
1 | +package c2eo.types
2 | [arg] > global
3 |   c_int32 > varName
4 |   seq > @
5 |     varName.write 10
```

Объявление инициализированной статической переменной

C:

```
1 | static int varName = 10;
   EO Ver 1:
1 | +package c2eo.types
2 | [] > uniqueNameUsingFileName
3 |   c_int32 10 > varName+UniqueId
```

EO Ver 2:

```
1 | +package c2eo.types
2 | [] > uniqueNameUsingFileName
3 |   c_int32 > varName+UniqueId
```

1.4 Выполнение операций обработки для базовых типов данных

C:

```
1 | float a = 0;
2 | float b = 1;
3 | b = a;
4 | b = a + b;
5 | b = a - b;
6 | b = a * b;
7 | b = a / b;
8 | ...
```

EO:

```
1 | cfloat > a
2 | cfloat > b
3 | ...
4 |
5 | seq
6 |   a.write 0
7 |   b.write 1
8 |   b.write a
9 |   b.write (a.add b)
10 |  b.write (a.sub b)
11 |  b.write (a.mul b)
12 |  b.write (a.div b)
13 |  ...
```

1.5 Трансформация указателей

Объекты для представления изменяемых указателей в ЕО отсутствуют. Предлагается ввести изменяемый объект, например, **memptr** который бы обеспечивал подключение разнообразных объектов языка ЕО. Используя его можно было бы переподключать любые объекты. В этом случае для многих простых ситуаций можно обеспечить трансформацию кода языка С во взаимосвязанные объекты ЕО.

C:

```
1 | int *x;
2 | float *y;
```

EO:

```

1 [] > cPtr
2   memptr > @
3   "ptr" > type
4   sprintf > toString
5   "pointer to "
6
7 cPtr > x
8 cPtr > y

```

Информация о типе в указателе не сохраняется и определяется исходя из того, какой объект будет подключен на текущий момент к объекту-указателю. Учитывая косвенное связывание объектов в ЕО, в ряде случаев (при отсутствии изменчивости) их можно подключать напрямую.

1.6 Примеры прямого использования указателей

Pointers Shifting

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Book {
5     int price;
6     char* title;
7 }
8
9 int main(int argc, char** argv) {
10     struct Book b = {10, "title"};
11     printf("Book.price = %d \n", b.price);
12     printf("Book.title = %s \n", b.title);
13 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_book
7
8 [args] > structC
9
10   c_book > b
11
12   seq > main
13     b.price.write 10

```

```

14     b.title.write "title"
15     stdout (sprintf "Book.price = %d \n" (b.price))
16     stdout (sprintf "Book.title = %s \n" (b.title))

```

Type Up-casting

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Book { int price; char* title; };
5
6  void f(void* b) {
7      char* title = ((struct Book*) b)->title;
8      printf("The title: %s", title);
9  }
10
11 int main() {
12     struct Book b = {10, "some"};
13     f(&b);
14 }

```

EO:

```

1  +package c2eo.examples
2
3  +alias org.eolang.io.stdout
4  +alias org.eolang.txt.sprintf
5
6  +alias c2eo.ctypes.c_book
7
8  [b] > f
9
10     memory > title
11
12     seq > @
13         title.write (b.title)
14         stdout (sprintf "The title: %s" title)
15
16 [args] > typeCastingC
17
18     c_book > b
19
20     seq > main
21         c_book.price.write 10

```

```

22 |     c_book.title.write "some"
23 |     f c_book

```

Memory Sizing

C:

```

1 | #include <stdlib.h>
2 | #include <stdio.h>
3 | struct Book { int price; char* title; };
4 | void f() {
5 |     Book* b = (Book*) malloc(sizeof(Book));
6 |     char title[] = "Object Thinking";
7 |     b->title = title;
8 |     printf("The title: %s", b->title);
9 |     free(b);
10 | }

```

Function Pointers

C:

```

1 | typedef struct { int price; } Book;
2 | typedef int (*read) (Book* b);
3 | void f(read r) {
4 |     Book* b = 0;
5 |     (*r)(b);
6 | }

```

1.7 Трансформация структур

Если внутри структуры непосредственно присутствуют другие артефакты, они могут трансформироваться в соответствующие объекты, также непосредственно включаемые во внешний объект.

Прямое создание структуры

C:

```

1 | struct rectangle {
2 |     int x, y;
3 | };

```

EO:

```

1 | [] > rectangle
2 |   cInt > x
3 |   cInt > y

```

При использовании в структуре языка С указателей предлагаемые указатели можно вставить и в объекте на EO, устанавливая и изменяя их в ходе вычислений.

Использование указателей на изменяемые поля

C:


```

1 struct person {
2     int age;
3     char* name;
4 };

```

EO:

```

1 [] > person
2   cInt > age
3   memptr > name

```

Предполагается, учитывая синтаксическую корректность исходной программы на C, данный указатель будет инициализирован корректным значением.

Пример использования структуры

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Book {
5     int price;
6     char* title;
7 }
8
9 int main(int argc, char** argv) {
10     struct Book b = {10, "title"};
11     printf("Book.price = %d \n", b.price);
12     printf("Book.title = %s \n", b.title);
13 }

```

EO:

```

1 +package c2eo.examples
2 +alias org.eolang.io.stdout
3 +alias org.eolang.txt.sprintf
4 +alias c2eo.ctypes.c_book
5
6 [args] > structC
7   c_book > b
8   seq > main
9     b.price.write 10
10    b.title.write "title"
11    stdout (sprintf "Book.price = %d \n" (b.price))
12    stdout (sprintf "Book.title = %s \n" (b.title))

```

1.8 Трансформация объединений

Варианты:

- (1) одна и та же область памяти заполняется различными по контексту данными, которые не коррелируются друг с другом;
- (2) записанное значение используется в качестве различных альтернативных трактовок.

Непосредственная трансформация объединений

C:

```
1 union figure {
2     rectangle r;
3     triangle t;
4 };
```

EO:

В общем случае невозможно напрямую трансформировать этот код, связанный с наложением областей памяти двух объектов. Как вариант, можно использовать в ряде случаев неоднозначное отображение, эквивалентное структуре на C:

```
1 struct figure {
2     rectangle r;
3     triangle t;
4 };
```

EO:

```
1 [] > figure
2     rectangle > r
3     triangle > t
```

Использование объединений с доступом через указатели

C:

```
1 union figure {
2     rectangle *r;
3     triangle *t;
4 };
```

EO:

```
1 [] > figure
2     memptr > rt
```

Пример анализа размещения целочисленных переменных в памяти

C:

```
1 #include <stdio.h>
2
3 union {
4     unsigned int i;
5     char c[sizeof(unsigned int)];
6 } foo;
7 int main() {
8     foo.i = 1;
```

```

9     if (foo.c[0] == 1) {
10         printf("Little endian\n");
11     } else {
12         printf("Big endian\n");
13     }
14     foo.i = 0x01020304;
15     for(int i = 0; i < sizeof(unsigned int); i++) {
16         printf("%p: %X\n", &foo.c[i], (unsigned int)foo.c[i]);
17     }
18     return 0;
19 }

```

EO:

В случае, когда размер данных в объединении соответствует размеру одного машинного слова, для реализации можно использовать объект **memory** (пример в процессе разработки...).

1.9 Трансформация enum

В принципе целочисленные значения перечислимого типа можно непосредственно подставлять в местах использования. Вместе с тем, их можно также определять в виде соответствующих констант.

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  enum week { Mon, Tue, Wed, Thur, Fri, Sat, Sun};
6
7  enum week currentDay = Sun;
8
9  enum week getSomeDay() {
10     return currentDay;
11 }
12
13 int main(int argc, char** argv) {
14     enum week someday, anotherDay;
15     someday = Wed;
16     printf("enum someday = %d\n", someday);
17     anotherDay = getSomeDay();
18     printf("enum another day = %d\n", anotherDay);
19     return 0;
20 }

```

EO ver. 1:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7
8 0 > mon
9 1 > tue
10 2 > wed
11 3 > thur
12 4 > fri
13 5 > sat
14 6 > sun
15
16 [args] > enum1C
17
18     sun > currentDay
19     c_int32 > someday
20     c_int32 > anotherDay
21
22     currentDay > getSomeDay
23
24     seq > main
25         someday.write 2
26         stdout (sprintf "enum1 someday = %d\n" someday)
27         anotherDay.write getSomeDay
28         stdout (sprintf "enum1 another day = %d\n" anotherDay)

```

EO ver. 2:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > enum2C
9
10 6 > currentDay
11 c_int32 > someday
12 c_int32 > anotherDay
13

```

```

14 |   currentDay > getSomeDay
15 |
16 |   seq > main
17 |     someday.write 2
18 |     stdout (sprintf "enum2 someday = %d\n" someday)
19 |     anotherDay.write getSomeDay
20 |     stdout (sprintf "enum2 another day = %d\n" anotherDay)

```

1.10 Трансформация typedef

Вместо typedef можно использовать исходный тип.

C:

```

1 | typedef struct rectangle {
2 |     int x, y;
3 | } rect;

```

EO:

```

1 | [] > rectangle
2 |   cInt > x
3 |   cInt > y
4 |
5 | rectangle > rect

```

1.11 Использование указателей

Указатели на простые переменные

C:

```

1 | #include <stdio.h>
2 | int x = 5;
3 | int *p;
4 |
5 | int main() {
6 |     int y;
7 |     p = &x;
8 |     *p = 25;
9 |     p = &y;
10 |    *p = x;
11 |    printf("x = %d\n y = %d\n *p = %d\n", x, y, *p);
12 |    return 0;
13 | }

```

EO:

```

1 | +package c2eo.examples
2 |
3 | +alias org.eolang.txt.sprintf

```

```

4 +alias org.eolang.io.stdout
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > pointers1C
9
10   c_int32 5 > x
11
12   [] > main
13
14     c_int32 > y
15
16     seq > @
17       ^.x.write 25
18       y.write (^x)
19       stdout (sprintf "x = %d\ny = %d\n*p = %d\n" (^x) y y)

```

Это нельзя назвать обращением через указатель. Идет передача значений. Для реальной трансформации нужен соответствующий объект-указатель

Разыменованный (бестиповой) указатель

C:

```

1 #include <stdio.h>
2 int x = 5;
3 void *p;
4
5 int main() {
6     int y;
7     p = &x;
8     *(int*)p = 25;
9     p = &y;
10    *(int*)p = x;
11    printf("x = %d\ny = %d\n*p = %d\n", x, y, *(int*)p);
12    return 0;
13 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.txt.sprintf
4 +alias org.eolang.io.stdout
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > pointers2C

```

```

9
10 c_int32 5 > x
11
12 [] > main
13
14 c_int32 > y
15
16 seq > @
17   ^.x.write 25
18   y.write (^x)
19   stdout (sprintf "x = %d\ny = %d\n*p = %d\n" (^x) y y)

```

Решение практически аналогично предыдущему

Цепочка указателей, ссылающихся друг на друга

C:

```

1 // Работа с указателями
2 // Формирование значений через void-указатели
3
4 #include <stdio.h>
5 int x = 5;
6 int *p;
7 int **pp;
8
9 int main() {
10     int y;
11     p = &x;
12     *p = 25;
13     pp = &p;
14     p = &y;
15     y = x + 10;
16     int ***ppp = &pp;
17     printf("x = %d\ny = %d\n*p = %d\n**pp = %d\n***ppp = %d\n",
18           x, y, *p, **pp, ***ppp);
19     return 0;
20 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.txt.sprintf
4 +alias org.eolang.io.stdout
5
6 +alias c2eo.ctypes.c_int32
7

```

```

8 [args] > pointers3C
9   c_int32 5 > x
10  [] > main
11    c_int32 > y
12    seq > @
13      ^.x.write 25
14      y.write (^x.add 10)
15      stdout (sprintf "x = %d\ny = %d\n*p = %d\n**pp = %d\n***ppp = %d\n"
        ↪ (^x) y y y y)

```

Решение невоспроизводимо.

1.12 Трансформация указателей на функции

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct { int price; } Book;
5
6 typedef int (*read) (Book* b);
7
8 int f(Book* b){
9     printf("price is %d", b->price);
10 }
11
12 void g(read r, Book* b) {
13     r(b);
14 }
15
16 int main(int argc, char** argv) {
17     Book b = { 7 };
18     Book* pb = &b;
19     g(f, pb);
20 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_book
7
8 [b] > f

```



```

9   seq > @
10   stdout (sprintf "price is %d" b.price)
11
12 [r b] > g
13   seq > @
14   r b
15
16 [args] > functionPointersC
17
18   c_book > b
19
20   seq > main
21   b.price.write 7
22   g f b

```

1.13 Трансформация операторов и операций

Оператор **return** в конце функции

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int a, int b) {
5      int c = a + b;
6      return c;
7  }
8
9  int main(int argc, char** argv) {
10     int a = atoi(argv[1]);
11     int result = f(a, a);
12     printf("simpleReturn[%d] = %d \n", a, result);
13 }

```

EO:

```

1  +package c2eo.examples
2
3  +alias org.eolang.io.stdout
4  +alias org.eolang.txt.sprintf
5
6  +alias c2eo.ctypes.c_int32
7
8  [args] > simpleReturnC
9
10 [a b] > f

```

```

11
12     c_int32 > c
13
14     if. > @
15         seq
16             c.write (a.add b)
17         c
18         error "Unexpected behavior"
19
20     [] > main
21         c_int32 > a
22         c_int32 > result
23
24     seq > @
25         a.write (~.args.get 0).toInt
26         result.write (~.f a a)
27         stdout (sprintf "simpleReturn[%d] = %d \n" a result)

```

Многократное использование оператора return

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f(int a) {
5      if (a == 1) {
6          a = a + 1;
7      }
8      if (a == 2) {
9          a = a + 2;
10         return 2;
11     }
12     else if (a == 3) {
13         a = a + 3;
14         return 3;
15     }
16     if (a == 4) {
17         a = a + 4;
18     }
19     a = a + 5;
20     return a;
21 }
22
23 int main(int argc, char** argv) {

```

```

24     int a = atoi(argv[1]);
25     int result = f(a);
26     printf("complexReturn[%d] = %d", a, result);
27 }

```

EO:

```

1  +package c2eo.examples
2  +alias org.eolang.io.stdout
3  +alias org.eolang.txt.sprintf
4  +alias c2eo.ctypes.c_int32
5  +alias c2eo.ctypes.c_bool
6
7  [args] > complexReturnC
8  [a] > f
9      c_int32 > result
10     c_bool > isReturn
11     if. > @
12         seq
13             isReturn.write FALSE
14             if.
15                 a.eq 1
16                 seq
17                     a.write (a.add 1)
18                 seq
19             if.
20                 a.eq 2
21                 seq
22                     a.write (a.add 2)
23                     isReturn.write TRUE
24                     result.write 2
25             if.
26                 a.eq 3
27                 seq
28                     a.write (a.add 3)
29                     isReturn.write TRUE
30                     result.write 3
31             seq
32         if.
33             isReturn
34             seq
35             if.
36                 seq
37                 if.

```

```

38         a.eq 4
39         seq
40         a.write (a.add 4)
41         seq
42     seq
43     a.write (a.add 5)
44     isReturn.write TRUE
45     result.write a
46     error "Unexpected behavior"
47 result
48 error "Unexpected behavior"
49
50 [] > main
51 c_int32 > a
52 c_int32 > result
53 seq > @
54     a.write (^args.get 0).toInt
55     result.write (^f a)
56     stdout (sprintf "complexReturn[%d] = %d\n" a result)

```

Цикл While-do

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int a = atoi(argv[1]);
6     int i = 0;
7     while (i < a)
8     {
9         printf("while[%d] ", i);
10        i++;
11    }
12 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > whileC

```

```

9
10 c_int32 > a
11 c_int32 > i
12
13 seq > main
14   a.write (args.get 0).toInt
15   i.write 0
16   while.
17     i.less a
18     [x]
19     seq > @
20     stdout (sprintf "while[%d] " (^i))
21     ^.i.write (^i.add 1)

```

Цикл do-while

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int a = atoi(argv[1]);
6     int i = 0;
7     do
8     {
9         printf("do while[%d] ", i);
10        i++;
11    } while (i < a);
12 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > dowhileC
9
10 c_int32 > a
11 c_int32 > i
12
13 [] > cycle_body
14   seq > @

```

```

15     stdout (sprintf "dowhile[%d] " (^i))
16     ^.i.write (^i.add 1)
17
18 seq > main
19     a.write (args.get 0).toInt
20     i.write 0
21
22     cycle_body
23     while.
24         i.less a
25         [x]
26         ^.cycle_body > @

```

Оператор цикла for

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int a = atoi(argv[1]);
6     for(int i = 0; i < a; i++) {
7         printf("for[%d] ", i);
8     }
9 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > forC
9
10 c_int32 > a
11 c_int32 > i
12
13 seq > main
14     a.write (args.get 0).toInt
15     i.write 0
16     while.
17         i.less a
18         [x]

```

```

19         seq > @
20         stdout (sprintf "for[%d] " (^i))
21         ^i.write (^i.add 1)

```

Оператор continue в цикле

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int collatzProblem(int number)
5  {
6      while (number > 1)
7      {
8          if ((number % 2) == 0 )
9          {
10             number = number / 2;
11             continue;
12          }
13
14             number = (number * 3) + 1;
15     }
16
17     return number;
18 }
19
20
21 int main(int argc, char** argv) {
22     int a = atoi(argv[1]);
23     int result = collatzProblem(a);
24     printf("collatzProblem(%d) = %d", a, result);
25 }

```

EO:

```

1  +package c2eo.examples
2
3  +alias org.eolang.io.stdout
4  +alias org.eolang.txt.sprintf
5
6  +alias c2eo.ctypes.c_bool
7  +alias c2eo.ctypes.c_int32
8
9  [args] > continueC
10
11  [number] > collatzProblem

```

```

12
13     c_int32 number > number1
14     c_bool FALSE > isContinue
15
16     if. > @
17         seq
18         while.
19             number1.greater 1
20             [x]
21             seq > @
22             if.
23                 (^number1.mod 2).eq 0
24                 seq
25                     ^number1.write (^number1.mod 2)
26                     ^isContinue.write TRUE
27                 seq
28                 if.
29                     ^isContinue
30                     seq
31                     ^number1.write ((^number1.mul 3).add 1)
32             number1
33             error "Unexpected behavior"
34
35     [] > main
36
37     c_int32 > a
38     c_int32 > result
39
40     seq > @
41     a.write (^args.get 0).as-int
42     result.write (^collatzProblem a)
43     stdout (sprintf "collatzProblem(%d) = %d" a result)

```

Оператор break в цикле

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int breakOrReturn(int a)
5 {
6     while (1)
7     {
8         if ((a % 5) == 0)

```



```

9      {
10         break;
11     }
12
13     if ((a % 3) == 0)
14     {
15         return 1;
16     }
17
18     a++;
19 }
20
21 printf("We broke out of the loop\n");
22 return 0;
23 }
24
25
26 int main(int argc, char** argv) {
27     int a = atoi(argv[1]);
28     int result = breakOrReturn(a);
29     printf("breakOrReturn = %d", result);
30 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7 +alias c2eo.ctypes.c_bool
8
9 [args] > breakC
10
11 [a] > breakOrReturn
12
13     c_int32 > a1
14     c_int32 > result
15     c_bool > isBreak
16     c_bool > isReturn
17
18     if. > @
19         seq

```

```

20     a1.write a
21     isBreak.write FALSE
22     isReturn.write FALSE
23     while.
24         TRUE.and (isBreak.not) (isReturn.not)
25         [i]
26             seq > @
27             if.
28                 (^a1.mod 5).eq 0
29                 seq
30                 ^isBreak.write TRUE
31                 seq
32             if.
33                 (^a1.mod 3).eq 0
34                 seq
35                 ^isReturn.write TRUE
36                 ^result.write 1
37                 seq
38             if.
39                 ^isReturn
40                 seq
41                 ^a1.write (^a1.add 1)
42                 seq
43         if.
44             isReturn
45             seq
46         if.
47             seq
48             stdout "We broke out of the loop\n"
49             result.write 0
50             seq
51
52     result
53     error "Unexpected behavior"
54
55 [] > main
56
57 c_int32 > a
58 c_int32 > result
59
60 seq > @
61     a.write (^args.get 0).as-int

```

```

62     result.write (^..breakOrReturn a)
63     stdout (sprintf "breakOrReturn = %d" result)

```

Операторы if

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv) {
5      int a = atoi(argv[1]);
6
7      if (a == 5) {
8          printf("IF work\n");
9      }
10     if (a == 5) {
11         printf("IF-else work\n");
12     }
13     else {
14         printf("if-ELSE work\n");
15     }
16     if (a == 5) {
17         printf("IF-else_if work\n");
18     }
19     else if(a == 6) {
20         printf("if-ELSE_IF work\n");
21     }
22     if (a == 5) {
23         printf("IF-else_if-else work\n");
24     }
25     else if (a == 6) {
26         printf("if-ELSE_IF-else work\n");
27     }
28     else {
29         printf("if-else_if_ELSE work\n");
30     }
31 }

```

EO:

```

1  +package c2eo.examples
2
3  +alias org.eolang.io.stdout
4  +alias org.eolang.txt.sprintf
5
6  +alias c2eo.ctypes.c_int32

```

```

7
8 [args] > ifC
9
10 c_int32 > a
11
12 seq > main
13   a.write (args.get 0).toInt
14
15   if.
16     a.eq 5
17     seq
18       stdout "IF work\n"
19     seq
20
21   if.
22     a.eq 5
23     seq
24       stdout "IF-else work\n"
25     seq
26       stdout "if-ELSE work\n"
27
28   if.
29     a.eq 5
30     seq
31       stdout "IF-else_if work\n"
32     if.
33       a.eq 6
34       seq
35         stdout "if-ELSE_IF work\n"
36       seq
37
38   if.
39     a.eq 5
40     seq
41       stdout "IF-else_if-else work\n"
42     if.
43       a.eq 6
44       seq
45         stdout "if-ELSE_IF-else work\n"
46       seq
47         stdout "if-else_if_ELSE work\n"

```

Оператор switch

C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     a = atoi(argv[1]);
6     printf("switch[%d] = ", a);
7     switch (a) {
8         case 0:
9             printf("0");
10            break;
11        case 1:
12            printf("1");
13            break;
14        case 5:
15            printf("5");
16            break;
17        default:
18            printf("default");
19            break;
20    }
21 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.io.stdout
4 +alias org.eolang.txt.sprintf
5
6 +alias c2eo.ctypes.c_int32
7
8 [args] > switchC
9
10 c_int32 > a
11
12 seq > main
13     a.write (args.get 0).toInt
14     stdout (sprintf "switch[%d] = " a)
15     if.
16         a.eq 0
17         seq
18             stdout "0"
19     if.

```

```

20     a.eq 1
21     seq
22     stdout "1"
23     if.
24     a.eq 5
25     seq
26     stdout "5"
27     seq
28     stdout "default"

```

1.14 Трансформация функций

Рекурсивное вычисление факториала с глобальными переменными

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int n;
5  int result = 1;
6
7  void factorial() {
8      if(n < 2) {
9          return;
10     } else {
11         result = result * n;
12         n = n - 1;
13         factorial();
14     }
15 }
16
17 int main(int argc, char** argv) {
18     n = atoi(argv[1]);
19     printf("%d! = ", n);
20     factorial();
21     printf("%d\n", result);
22 }

```

EO:

```

1  +package c2eo.examples
2
3  +alias org.eolang.txt.sprintf
4  +alias org.eolang.io.stdout
5  +alias c2eo.ctypes.c_int32
6

```

```

7 [args] > factorialC
8   c_int32 > n
9   c_int32 > result
10  if. > factorial
11    n.less 2
12    seq
13    seq
14      result.write (n.mul result)
15      n.write (n.sub 1)
16      factorial
17  seq > main
18    result.write 1
19    n.write (args.get 0).toInt
20    stdout (sprintf "%d! = " n)
21    factorial
22    stdout (sprintf "%d\n" result)

```

Рекурсивное вычисление функции Фибоначчи с глобальными переменными
C:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int n;
5 int result = 1;
6 int lastResult = 0;
7 int tmp;
8
9 void fibonacci() {
10     if(n < 1) {
11         return;
12     } else {
13         tmp = result;
14         result = result + lastResult;
15         lastResult = tmp;
16         n = n - 1;
17         fibonacci();
18     }
19 }
20
21 int main(int argc, char** argv) {
22     n = atoi(argv[1]);
23     printf("fibonacci[%d] = ", n);
24     fibonacci();

```

```

25     printf("%d\n", lastResult);
26 }
    C:
1  +package c2eo.examples
2
3  +alias org.eolang.txt.sprintf
4  +alias org.eolang.io.stdout
5  +alias c2eo.ctypes.c_int32
6
7  [args] > fibonacciC
8      c_int32 > n
9      c_int32 > result
10     c_int32 > lastResult
11     c_int32 > tmp
12
13     if. > fibonacci
14         n.less 1
15         seq
16         seq
17             tmp.write result
18             result.write (result.add lastResult)
19             lastResult.write tmp
20             n.write (n.sub 1)
21             fibonacci
22
23     seq > main
24         result.write 1
25         lastResult.write 0
26         n.write (args.get 0).toInt
27         stdout (sprintf "fibonacci[%d] = " n)
28         fibonacci
29         stdout (sprintf "%d\n" result)

```

Вычисление числа Пи

C:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int n;
5  int i = 0;
6  float divider = 1.0;
7  float result = 0;
8

```



```

9 void pi() {
10     if(i < n) {
11         result = result + (1.0 / ((i % 2 == 0) ? divider : -divider));
12         divider = divider + 2.0;
13         i = i + 1;
14         pi();
15     } else {
16         result = result * 4.0;
17         return;
18     }
19 }
20
21 int main(int argc, char** argv) {
22     n = atoi(argv[1]);
23     printf("pi[%d] = ", n);
24     pi();
25     printf("%f\n", result);
26 }

```

EO:

```

1 +package c2eo.examples
2
3 +alias org.eolang.txt.sprintf
4 +alias org.eolang.io.stdout
5
6 +alias c2eo.ctypes.c_int32
7 +alias c2eo.ctypes.c_float64
8
9 [args] > piC
10
11     c_int32 > n
12     c_int32 > i
13     c_float64 > divider
14     c_float64 > result
15
16     if. > pi
17         i.less n
18         seq
19             result.write
20             result.add
21             1.0.div
22             if.
23                 (i.mod 2).eq 0

```

```

24         divider
25         divider.neg
26
27         divider.write (divider.add 2.0)
28         i.write (i.add 1)
29         pi
30
31     seq
32         result.write (result.mul 4.0)
33
34 seq > main
35     n.write (args.get 0).toInt
36     i.write 0
37     divider.write 1.0
38     result.write 0.0
39     stdout (sprintf "pi[%d] = " n)
40     pi
41     stdout (sprintf "%f\n" result)

```

2 ОБЩАЯ КОНЦЕПЦИЯ ПОСТРОЕНИЯ ПРОГРАММЫ НА ЕО, ПОЛУЧЕННОЙ В ХОДЕ ТРАНСПИЛЯЦИИ С ЯЗЫКА ПРОГРАММИРОВАНИЯ C

2.1 Структура каталогов

result

result - каталог для хранения данных, используемых компилятором ЕО. В нем содержится информация о проекте на ЕО, обновляемая каждый раз при обновлении проекта этого компилятора:

- **pom.xml** - берется из текущей версии ЕО с возможной заменой имени проекта на c2eo;
- **README.md** - описание компилируемого проекта, которое формируется разработчиками c2eo и практически не меняется (может только корректироваться);
- **run.sh** - скрипт, осуществляющий запуск откомпилированного приложения;
- **eo** - каталог, содержащий используемые библиотеки, написанные на ЕО для поддержки артефактов языка C, а также исходные тексты на ЕО, порождаемые транслятором или сформированные вручную.

Внутри каталога **eo** структура сформирована подкаталогов. Непосредственно в **eo** находится каталог **c2eo**, определяющий общее название пакета. В нем располагаются:

- файл **app.eo**, отвечающий за запуск приложения (он пишется вручную и не изменяется);
- каталог **ctypes**, который по сути определяет некоторую библиотеку объектов, написанную на ЕО и предназначенную для моделирования различных артефактов языка C;

- каталог **src**, в который записывается файл **global.eo** с объектами, порожденными транслятором в ходе анализа программы на C (он содержит объект **global**, в котором собраны все артефакты в виде соответствующих объектов).

Формирование файла **global.eo** по сути является основной задачей транслятора и обеспечивает путем сборки множества единиц компиляции исходной программы на языке C.

llvm-clang

Каталог **llvm-clang** предназначен для хранения сборки проекта llvm. Предполагается, что данная сборка будет формироваться на уровне бинарных кодов под конкретную автономную реализацию (докер, VM) и впоследствии не будет меняться. Вряд ли в ходе разработки проекта стоит переходить на более свежую версию llvm без особых на то оснований. Нахождение внутри проекта позволит ее распространять вместе с результатом работы. При этом можно посмотреть и выкинуть все лишнее, что не нужно для проекта, тем самым сократив 8 гигабайт до более приемлемого значения.

project

Каталог **project** содержит все то, что является результатом разработки. В данный момент в нем просматриваются следующие каталоги:

- **assembly** - каталог, предназначенный для хранения промежуточных результатов, а также окончательного результата работы транслятора. Из отдельных промежуточных файлов в нем формируется окончательный файл **global.eo**, который затем копируется (пересылается) в соответствующий подкаталог каталога **result**.
- **bin** - каталог, в котором группируются все исполняемые файлы и скрипты, обеспечивающие работу транслятора.
- **build** - каталог предназначенный для сборки проекта. Предполагается сборка проекта с использованием **cmake**. В связи с этим используется иерархическая организация файлов в каждом из подпроектов, обеспечивающих выполнение отдельных функций, при необходимости должен размещаться свой файл **CMakeLists.txt**. Также корневой файл **CMakeLists.txt** располагается в каталоге **project**.
- **lib** - каталог, предназначенный для хранения статических и (или) динамических библиотек, формируемых в процессе создания проекта, если таковые появятся. Пока особых предпосылок для их появления не наблюдается.
- **src** - каталог с исходными текстами программ проекта, состоящий из подкаталогов, в каждом из которых формируется свой проект.
- **tests** - каталог с различными тестовыми программами и данными, проверяющими работоспособность разрабатываемого кода.

Каталог **src** является ключевым для разработки. В целом его содержание скорее всего будет меняться. Но на данном этапе просматриваются два основных проекта, размещенные в соответствующих каталогах.

- **transpiler** - транслятор, осуществляющий разбор AST одной единицы компиляции и формирующий на выходе объекты EO. Эти объекты размещаются в каталоге **assembly** и разделяются по двум файлам. В одном собираются все объекты, соответствующие глобальным артефактам, а в другом статическим. Учитывая специфику анализа AST, данный проект реализуется на C++.

- **collector** - каталог, в котором разрабатывается сборщик файла **global.eo**. В общем случае после обработки транслятором всех единиц компиляции он осуществляет компоновку из множества файлов, содержащих как статические, так и глобальные объекты, единый файл объектов на ЕО. Разработка данной программы может вестись на языке сценариев.
- **launcher** - каталог, содержащий программу, которая осуществляет многократный запуск транслятора по числу единиц компиляции, после чего передает управление сборщику полученных отдельных файлов в монолит. После завершения сборки данная программа осуществляет перенос сформированного файла **global.eo** в каталог **result/eo/c2eo/src**.

Каталог **doc** содержит документацию, формируемую в ходе работы над проектом.

collection

Каталог **collection** содержит исходные тексты программ на языках программирования С и ЕО, которые предполагается использовать как для интеграционного тестирования транслятора, так и для проверки возможных вариантов трансформации в ЕО. Программы на С размещаются в подкаталоге 'c-sources'. Они формируют наборы данных, позволяющие оценить работоспособность разрабатываемого транслятора. В подкаталоге 'eo-sources' размещаются программы на ЕО, которые используются для анализа различных вариантов кодогенерации, а также для анализа возможности трансформации программ с С в ЕО.

2.2 Размещение программы на ЕО, полученной в ходе трансляции

Представленная структура стала возможной из-за использования начальной инициализации объектов, имитирующих переменные языка С. Трансляция осуществляется отдельно для каждой единицы компиляции, которая на выходе формирует два файла:

- файл с описанием всех глобальных объектов, к которым относятся абстрактные объекты, полученные при трансформации абстрактных типов данных, глобальных переменных, глобальных описаний функций;
- файл с описанием всех статических объектов, которые трансформируются из описаний статических переменных и функций, расположенных в глобальном пространстве, статических переменных, размещенных внутри функций.

Эти два файла являются базовой заготовкой для дальнейшей сборки после трансляции всех единиц компиляции проекта. Сама сборка на текущий момент заключается в формировании общего файла на языке программирования ЕО. В нем формируется глобальный объект **global**, который содержит все объекты, полученные в результате компиляции абстрактных типов данных, внешних переменных, внешних функций, а также объектов, которые получены из файлов, описывающих статические объекты. Количество статических объектов определяется количеством файлов со статическими артефактами. Размещение в едином объекте **global** всех данных позволяет без проблем обеспечить доступ как со стороны глобальных объектов к своим статическим данным, так и со стороны статических объектов к глобальным данным. Сборщик этого файла может в принципе быть отдельной программой, реализованной на любом удобном языке программирования.

При наличии в одной из единиц компиляции функции **main**, она преобразуется в соответствующий объект глобального пространства. А сразу за его описанием следует описание ее

запуска. Функция может располагаться в любом месте глобального объекта. В целом порядок сборки файла с глобальными объектами и статическими объектами несущественен.

Представленная схема обеспечивает полную автономность формирования программы на EO. Объект, запускающий приложение содержит только датаизацию глобального объекта. Он не меняется, оставаясь постоянным независимо от транспилируемого проекта.

2.3 Пример трансформации

Рассматривается пример обмена двух переменных значениями.

Исходный текст программы на языке C:

```

1  #include <stdio.h>
2
3  int a = 0;
4  int b = 10;
5  int c = 30;
6
7  int main() {
8      printf("Start: a = %d, b = %d, c = %d\n", a, b, c);
9      a = b;
10     b = c;
11     c = a;
12     printf("Finish: a = %d, b = %d, c = %d\n", a, b, c);
13     return 0;
14 }
```

Предполагаемы файл global.eo, который будет формироваться в ходе трансформации:

```

1  +package c2eo.src
2
3  +alias stdout org.eolang.io.stdout
4  +alias sprintf org.eolang.txt.sprintf
5
6  +alias c2eo.ctypes.c_bool
7  +alias c2eo.ctypes.c_char
8  +alias c2eo.ctypes.c_float64
9  +alias c2eo.ctypes.c_int16
10 +alias c2eo.ctypes.c_int32
11 +alias c2eo.ctypes.c_int64
12
13 [arg] > global
14   c_int64 0 > a
15   c_int64 10 > b
16   c_int64 30 > c
17
18   [arg] > main
```

```

19     seq > @
20     stdout
21     sprintf
22     "Start: b = %s, c = %s\n"
23     ^.b.as-string
24     ^.c.as-string
25     ^.a.write (^b)
26     ^.b.write (^c)
27     ^.c.write (^a)
28     stdout
29     sprintf
30     "Finish: b = %s, c = %s\n"
31     ^.b.as-string
32     ^.c.as-string
33     main arg > @

```

Файл app.eo, запускающий сформированное приложение

```

1 +package c2eo
2
3 +alias stdout org.eolang.io.stdout
4 +alias sprintf org.eolang.txt.sprintf
5
6 +alias c2eo.src.global
7
8 [args...] > app
9     global args > @

```