Mapping of programs in C language to programs in EO language

1 ВАРИАНТЫ ТРАНСФОРМАЦИИ КОНСТРУКЦИЙ ЯЗЫКА С В ЕО

1.1 Константы

Целочисленные константы

C:

```
const int constantName = 10;
```

EO:

```
10 > constantName
```

Константы с плавающей точкой

C:

```
const double constantName = 3.14;
```

EO:

```
3.14 > constantName
```

Булевские константы

C:

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

EO:

```
true > constantName
```

Символьные константы

C:

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

EO:

```
"a" > constantName
```

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

1.2 Трансформация переменных

16-разрядные целочисленные переменные

C:

```
short int varName;
```

EO:

Author's address:

```
+package c2eo.ctypes
+alias org.eolang.txt.sprintf
[] > c_int16
  memory > @
  [value] > write
    -32768 > min
    32767 > max
    65536 > range
    if. > @
      value.greater max
      ^.@.write (((value.add min).mod range).add min)
      if.
        value.less min
        ^.@.write (((value.add 32769).mod range).add max)
        ^.@.write value
  sprintf "%d" $ > toString
  "int16" > type
  4 > bytes
  32767 > max
  -32768 > min

c_int16 > varName
```

32-разрядные целочисленные переменные

C:

```
int varName;
```

EO:

```
+package c2eo.ctypes
+alias org.eolang.txt.sprintf
[] > c_int32
  memory > @
  [value] > write
    -2147483648 > min
    2147483647 > max
    4294967296 > range
    if. > @
      value.greater max
      ^.@.write (((value.add min).mod range).add min)
      if.
        value.less min
        ^.@.write (((value.add 2147483649).mod range).add max)
        ^.@.write value
  sprintf "%d" $ > toString
```

```
  "int32" > type
  4 > bytes
  -2147483648 > min
  2147483647 > max

c_int32 > varName
```

64-разрядные целочисленные переменные
C:

```
long long int varName;
```

EO:

```
+package c2eo.ctypes
+alias org.eolang.txt.sprintf
[] > c_int64
  memory > @
  sprintf "%d" $ > toString
  "int64" > type
  8 > bytes
  -9223372036854775808 > min
  9223372036854775807 > max

c_int64 > varName
```

64-разрядные переменные с плавающей точкой
C:

```
double varName;
```

EO:

```
+package c2eo.ctypes
+alias org.eolang.txt.sprintf
[] > c_float64
  memory > @
  sprintf "%f" $ > toString
  "float64" > type
  8 > bytes

c_float64 > varName
```

Булевские переменные
C:

```
_Bool varName;
```

EO:

```
+package c2eo.ctypes
+alias org.eolang.txt.sprintf
```

```
[] > c_bool
  memory > @
  sprintf "%b" $ > toString
  "bool" > type

c_bool > varName
```

Символьные переменные
C:

```
char varName;
```

EO:

```
+package c2eo.ctypes
+alias org.eolang.txt.sprintf
[] > c_char
  memory > @
  sprintf "%c" $ > toString
  "char" > type

c_char > varName
```

1.3 Трансформация глобальных переменных

Объявление инициализированной глобальной переменной
C:

```
int varName = 10;
```

EO:

```
+package c2eo.types
[arg] > global
  c_int32 > varName
  ...
  seq > @
    varName.write 10
    ...
```

Объявление инициализированной статической переменной
C:

```
static int varName = 10;
```

EO:

```
+package c2eo.types
[] > uniqueNameUsingFileName
  cInt > varName+UniqueId
  ...
  seq > @
```

```
    varName.write 10
    ...
```

1.4 Выполнение операций обработки для базовых типов данных

C:

```
float a = 0;
float b = 1;
b = a;
b = a + b;
b = a - b;
b = a * b;
b = a / b;
...
```

EO:

```
cfloat > a
cfloat > b
...

seq
  a.write 0
  b.write 1
  b.write a
  b.write (a.add b)
  b.write (a.sub b)
  b.write (a.mul b)
  b.write (a.div b)
  ...
```

1.5 Трансформация указателей

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

C:

```
int *x;
float *y;
```

EO:

```
[] > cPtr
  memptr > @
  "ptr" > type
```

```
  sprintf > toString
    "pointer to "

cPtr > x
cPtr > y
```

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

1.6 Примеры прямого использования указателей

Pointers Shifting

C:

```
#include <stdio.h>
#include <stdlib.h>

struct Book {
    int price;
    char* title;
}

int main(int argc, char** argv) {
    struct Book b = {10, "title"};
    printf("Book.price = %d \n", b.price);
    printf("Book.title = %s \n", b.title);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_book

[args] > structC

  c_book > b

  seq > main
    b.price.write 10
    b.title.write "title"
    stdout (sprintf "Book.price = %d \n" (b.price))
    stdout (sprintf "Book.title = %s \n" (b.title))
```

Type Up-casting

C:

```
#include <stdio.h>
#include <stdlib.h>

struct Book { int price; char* title; };

void f(void* b) {
    char* title = ((struct Book*) b)->title;
    printf("The title: %s", title);
}

int main() {
    struct Book b = {10, "some"};
    f(&b);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_book

[b] > f

  memory > title

  seq > @
    title.write (b.title)
    stdout (sprintf "The title: %s" title)

[args] > typeCastingC

  c_book > b

  seq > main
    c_book.price.write 10
    c_book.title.write "some"
    f c_book
```

Memory Sizing

C:

```c
#include <stdlib.h>
#include <stdio.h>
struct Book { int price; char* title; };
void f() {
  Book* b = (Book*) malloc(sizeof(Book));
  char title[] = "Object Thinking";
  b->title = title;
  printf("The title: %s", b->title);
  free(b);
}
```

Function Pointers

C:

```c
typedef struct { int price; } Book;
typedef int (*read) (Book* b);
void f(read r) {
  Book* b = 0;
  (*r)(b);
}
```

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:

```c
struct rectangle {
    int x, y;
};
```

EO:

```
[] > rectangle
  cInt > x
  cInt > y
```

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

Использование указателей на изменяемые поля

C:

```c
struct person {
    int age;
    char* name;
};
```

EO:

```
[] > person
  cInt > age
  memptr > name
```

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

Пример использования структуры

C:

```
#include <stdio.h>
#include <stdlib.h>

struct Book {
    int price;
    char* title;
}

int main(int argc, char** argv) {
    struct Book b = {10, "title"};
    printf("Book.price = %d \n", b.price);
    printf("Book.title = %s \n", b.title);
}
```

EO:

```
+package c2eo.examples
+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf
+alias c2eo.ctypes.c_book

[args] > structC
  c_book > b
  seq > main
    b.price.write 10
    b.title.write "title"
    stdout (sprintf "Book.price = %d \n" (b.price))
    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 кореллируются друг с другом;
(2) записанное значение используется в качестве различных альтернативных трактовок.

Непосредственная трансформация объединений

C:

```
union figure {
    rectangle r;
    triangle  t;
};
```

EO:

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

```
struct figure {
    rectangle r;
    triangle  t;
};
```

EO:

```
[] > figure
  rectangle > r
  triangle > t
```

Использование объединений с доступом через указатели

C:

```
union figure {
    rectangle *r;
    triangle  *t;
};
```

EO:

```
[] > figure
  memptr > rt
```

Пример анализа размещения целочисленных переменных в памяти

C:

```
#include <stdio.h>

union {
    unsigned int i;
    char c[sizeof(unsigned int)];
} foo;
int main() {
    foo.i = 1;
    if (foo.c[0] == 1) {
        printf("Little endian\n");
    } else {
        printf("Big endian\n");
    }
    foo.i = 0x01020304;
```

```
    for(int i = 0; i < sizeof(unsigned int); i++) {
        printf("%p: %X\n", &foo.c[i], (unsigned int)foo.c[i]);
    }
    return 0;
}
```

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:

```
#include <stdio.h>
#include <stdlib.h>


enum week { Mon, Tue, Wed, Thur, Fri, Sat, Sun};

enum week currentDay = Sun;

enum week getSomeDay() {
    return currentDay;
}

int main(int argc, char** argv) {
  enum week someday, anotherDay;
  someday = Wed;
  printf("enum someday = %d\n", someday);
  anotherDay = getSomeDay();
  printf("enum another day = %d\n", anotherDay);
  return 0;
}
```

EO ver. 1:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32

```

```
0 > mon
1 > tue
2 > wed
3 > thur
4 > fri
5 > sat
6 > sun

[args] > enum1C

  sun > currentDay
  c_int32 > someday
  c_int32 > anotherDay

  currentDay > getSomeDay

  seq > main
    someday.write 2
    stdout (sprintf "enum1 someday = %d\n" someday)
    anotherDay.write getSomeDay
    stdout (sprintf "enum1 another day = %d\n" anotherDay)
```

EO ver. 2:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32

[args] > enum2C

  6 > currentDay
  c_int32 > someday
  c_int32 > anotherDay

  currentDay > getSomeDay

  seq > main
    someday.write 2
    stdout (sprintf "enum2 someday = %d\n" someday)
    anotherDay.write getSomeDay
    stdout (sprintf "enum2 another day = %d\n" anotherDay)
```

1.10 Трансформация typedef

Вместо typedef можно использовать исходный тип.

C:

```
typedef struct rectangle {
    int x, y;
} rect;
```

EO:

```
[] > rectangle
  cInt > x
  cInt > y

rectangle > rect
```

1.11 Использование указателей

Указатели на простые переменные

C:

```
#include <stdio.h>
int x = 5;
int *p;

int main() {
    int y;
    p = &x;
    *p = 25;
    p = &y;
    *p = x;
    printf("x = %d\ny = %d\n*p = %d\n", x, y, *p);
    return 0;
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.txt.sprintf
+alias org.eolang.io.stdout

+alias c2eo.ctypes.c_int32

[args] > pointers1C

  c_int32 5 > x

```

```
  [] > main

    c_int32 > y

    seq > @
      ^.x.write 25
      y.write (^.x)
      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:

```
#include <stdio.h>
int x = 5;
void *p;

int main() {
    int y;
    p = &x;
    *(int*)p = 25;
    p = &y;
    *(int*)p = x;
    printf("x = %d\ny = %d\n*p = %d\n", x, y, *(int*)p);
    return 0;
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.txt.sprintf
+alias org.eolang.io.stdout

+alias c2eo.ctypes.c_int32

[args] > pointers2C

  c_int32 5 > x

  [] > main

    c_int32 > y

    seq > @
```

```
      ^.x.write 25
      y.write (^.x)
      stdout (sprintf "x = %d\ny = %d\n*p = %d\n" (^.x) y y)
```

Решение практически аналогично предыдущему

Цепочка указателей, ссылающихся друг на друга

C:

```
// Работа с указателями
// Формирование значений через void-указатели

#include <stdio.h>
int x = 5;
int *p;
int **pp;

int main() {
    int y;
    p = &x;
    *p = 25;
    pp = &p;
    p = &y;
    y = x + 10;
    int ***ppp = &pp;
    printf("x = %d\ny = %d\n*p = %d\n**pp = %d\n***ppp = %d\n",
           x, y, *p, **pp, ***ppp);
    return 0;
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.txt.sprintf
+alias org.eolang.io.stdout

+alias c2eo.ctypes.c_int32

[args] > pointers3C
  c_int32 5 > x
  [] > main
    c_int32 > y
    seq > @
      ^.x.write 25
```

```
      y.write (^.x.add 10)
      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:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct { int price; } Book;

typedef int (*read) (Book* b);

int f(Book* b){
    printf("price is %d", b->price);
}

void g(read r, Book* b) {
    r(b);
}

int main(int argc, char** argv) {
    Book b = { 7 };
    Book* pb = &b;
    g(f, pb);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_book

[b] > f
  seq > @
    stdout (sprintf "price is %d" b.price)

[r b] > g
  seq > @
    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)
```

```
      c
      error "Unexpected behavior"

  [] > main
    c_int32 > a
    c_int32 > result

    seq > @
      a.write (^.args.get 0).toInt
      result.write (^.f a a)
      stdout (sprintf "simpleReturn[%d] = %d \n" a result)
```

Многократное использование оператора return

C:

```
#include <stdio.h>
#include <stdlib.h>

int f(int a) {
  if (a == 1) {
    a = a + 1;
  }
  if (a == 2) {
      a = a + 2;
      return 2;
  }
  else if (a == 3) {
      a = a + 3;
      return 3;
  }
  if (a == 4) {
    a = a + 4;
  }
  a = a + 5;
  return a;
}

int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    int result = f(a);
    printf("complexReturn[%d] = %d", a, result);
}
```

EO:

```
+package c2eo.examples
+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf
+alias c2eo.ctypes.c_int32
+alias c2eo.ctypes.c_bool

[args] > complexReturnC
  [a] > f
    c_int32 > result
    c_bool > isReturn
    if. > @
      seq
        isReturn.write false
        if.
          a.eq 1
          seq
            a.write (a.add 1)
          seq
        if.
          a.eq 2
          seq
            a.write (a.add 2)
            isReturn.write true
            result.write 2
          if.
            a.eq 3
            seq
              a.write (a.add 3)
              isReturn.write true
              result.write 3
            seq
        if.
          isReturn
          seq
          if.
            seq
              if.
                a.eq 4
                seq
                  a.write (a.add 4)
                seq
            seq
              a.write (a.add 5)
```

```
              isReturn.write true
              result.write a
            error "Unexpected behavior"
      result
      error "Unexpected behavior"

  [] > main
    c_int32 > a
    c_int32 > result
    seq > @
      a.write (^.args.get 0).toInt
      result.write (^.f a)
      stdout (sprintf "complexReturn[%d] = %d\n" a result)
```

Цикл While-do

C:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    int i = 0;
    while (i < a)
    {
        printf("while[%d] ", i);
        i++;
    }
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32

[args] > whileC

  c_int32 > a
  c_int32 > i

  seq > main
    a.write (args.get 0).toInt
```

```
    i.write 0
    while.
      i.less a
      [x]
        seq > @
          stdout (sprintf "while[%d] " (^.i))
          ^.i.write (^.i.add 1)
```

Цикл do-while

C:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    int i = 0;
    do
    {
        printf("do while[%d] ", i);
        i++;
    } while (i < a);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32

[args] > dowhileC

  c_int32 > a
  c_int32 > i

  [] > cycle_body
    seq > @
      stdout (sprintf "dowhile[%d] " (^.i))
      ^.i.write (^.i.add 1)

  seq > main
    a.write (args.get 0).toInt
    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:

```c
#include <stdio.h>
#include <stdlib.h>

int collatzProblem(int number)
{
    while (number > 1)
    {
        if ((number % 2) == 0 )
        {
            number = number / 2;
            continue;
        }

        number = (number * 3) + 1;
    }

    return number;
}


int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    int result = collatzProblem(a);
    printf("collatzProblem(%d) = %d", a, result);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_bool
+alias c2eo.ctypes.c_int32

[args] > continueC

  [number] > collatzProblem

    c_int32 number > number1
    c_bool FALSE > isContinue

    if. > @
```

```
      seq
        while.
          number1.greater 1
          [x]
            seq > @
              if.
                (^.number1.mod 2).eq 0
                seq
                  ^.number1.write (^.number1.mod 2)
                  ^.isContinue.write TRUE
                seq
              if.
                ^.isContinue
                seq
                ^.number1.write ((^.number1.mul 3).add 1)
      number1
      error "Unexpected behavior"

  [] > main

    c_int32 > a
    c_int32 > result

    seq > @
      a.write (^.args.get 0).as-int
      result.write (^.collatzProblem a)
      stdout (sprintf "collatzProblem(%d) = %d" a result)
```

Оператор break в цикле

C:

```
#include <stdio.h>
#include <stdlib.h>

int breakOrReturn(int a)
{
    while (1)
    {
        if ((a % 5) == 0)
        {
            break;
        }

        if ((a % 3) == 0)
```

```
        {
            return 1;
        }

        a++;
    }

    printf("We broke out of the loop\n");
    return 0;
}


int main(int argc, char** argv) {
    int a = atoi(argv[1]);
    int result = breakOrReturn(a);
    printf("breakOrReturn = %d", result);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32
+alias c2eo.ctypes.c_bool

[args] > breakC

  [a] > breakOrReturn

    c_int32 > a1
    c_int32 > result
    c_bool > isBreak
    c_bool > isReturn

    if. > @
      seq
        a1.write a
        isBreak.write FALSE
        isReturn.write FALSE
        while.
          TRUE.and (isBreak.not) (isReturn.not)
```

```
          [i]
            seq > @
              if.
                (^.a1.mod 5).eq 0
                seq
                  ^.isBreak.write TRUE
                seq
              if.
                (^.a1.mod 3).eq 0
                seq
                  ^.isReturn.write TRUE
                  ^.result.write 1
                seq
              if.
                ^.isReturn
                seq
                  ^.a1.write (^.a1.add 1)
                seq
        if.
          isReturn
          seq
          if.
            seq
              stdout "We broke out of the loop\n"
            result.write 0
            seq

      result
      error "Unexpected behavior"

  [] > main

    c_int32 > a
    c_int32 > result

    seq > @
      a.write (^.args.get 0).as-int
      result.write (^.breakOrReturn a)
      stdout (sprintf "breakOrReturn = %d" result)
```

Операторы if
C:

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    int a = atoi(argv[1]);

    if (a == 5) {
      printf("IF work\n");
    }
    if (a == 5) {
      printf("IF-else work\n");
    }
    else {
      printf("if-ELSE work\n");
    }
    if (a == 5) {
      printf("IF-else_if work\n");
    }
    else if(a == 6) {
      printf("if-ELSE_IF work\n");
    }
    if (a == 5) {
      printf("IF-else_if-else work\n");
    }
    else if (a == 6) {
      printf("if-ELSE_IF-else work\n");
    }
    else {
      printf("if-else_if_ELSE work\n");
    }
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32

[args] > ifC

  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
```

```
int main(int argc, char** argv) {
    a = atoi(argv[1]);
    printf("switch[%d] = ", a);
    switch (a) {
     case 0:
       printf("0");
       break;
     case 1:
       printf("1");
       break;
     case 5:
       printf("5");
       break;
     default:
       printf("default");
       break;
   }
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.io.stdout
+alias org.eolang.txt.sprintf

+alias c2eo.ctypes.c_int32

[args] > switchC

  c_int32 > a

  seq > main
    a.write (args.get 0).toInt
    stdout (sprintf "switch[%d] = " a)
    if.
      a.eq 0
      seq
        stdout "0"
      if.
        a.eq 1
        seq
          stdout "1"
        if.
```

```
          a.eq 5
          seq
            stdout "5"
          seq
            stdout "default"
```

1.14 Трансформация функций

Рекусивное вычисление факториала с глобальными переменными

C:

```
#include <stdio.h>
#include <stdlib.h>

int n;
int result = 1;

void factorial() {
    if(n < 2) {
        return;
    } else {
        result = result * n;
        n = n - 1;
        factorial();
    }
}

int main(int argc, char** argv) {
    n = atoi(argv[1]);
    printf("%d! = ", n);
    factorial();
    printf("%d\n", result);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.txt.sprintf
+alias org.eolang.io.stdout
+alias c2eo.ctypes.c_int32

[args] > factorialC
  c_int32 > n
  c_int32 > result
  if. > factorial
```

```
    n.less 2
    seq
    seq
      result.write (n.mul result)
      n.write (n.sub 1)
      factorial
  seq > main
    result.write 1
    n.write (args.get 0).toInt
    stdout (sprintf "%d! = " n)
    factorial
    stdout (sprintf "%d\n" result)
```

Рекусивное вычисление функции Фибоначчи с глобальными переменными
C:

```
#include <stdio.h>
#include <stdlib.h>

int n;
int result = 1;
int lastResult = 0;
int tmp;

void fibonacci() {
    if(n < 1) {
        return;
    } else {
        tmp = result;
        result = result + lastResult;
        lastResult = tmp;
        n = n - 1;
        fibonacci();
    }
}

int main(int argc, char** argv) {
    n = atoi(argv[1]);
    printf("fibonacci[%d] = ", n);
    fibonacci();
    printf("%d\n", lastResult);
}
```

C:

```
+package c2eo.examples

+alias org.eolang.txt.sprintf
+alias org.eolang.io.stdout
+alias c2eo.ctypes.c_int32

[args] > fibonacciC
  c_int32 > n
  c_int32 > result
  c_int32 > lastResult
  c_int32 > tmp

  if. > fibonacci
    n.less 1
    seq
    seq
      tmp.write result
      result.write (result.add lastResult)
      lastResult.write tmp
      n.write (n.sub 1)
      fibonacci

  seq > main
    result.write 1
    lastResult.write 0
    n.write (args.get 0).toInt
    stdout (sprintf "fibonacci[%d] = " n)
    fibonacci
    stdout (sprintf "%d\n" result)
```

Вычисление числа Пи

C:

```
#include <stdio.h>
#include <stdlib.h>

int n;
int i = 0;
float divider = 1.0;
float result = 0;

void pi() {
    if(i < n) {
        result = result + (1.0 / ((i % 2 == 0) ? divider : -divider));
```

```
        divider = divider + 2.0;
        i = i + 1;
        pi();
    } else {
        result = result * 4.0;
        return;
    }
}

int main(int argc, char** argv) {
    n = atoi(argv[1]);
    printf("pi[%d] = ", n);
    pi();
    printf("%f\n", result);
}
```

EO:

```
+package c2eo.examples

+alias org.eolang.txt.sprintf
+alias org.eolang.io.stdout

+alias c2eo.ctypes.c_int32
+alias c2eo.ctypes.c_float64

[args] > piC

  c_int32 > n
  c_int32 > i
  c_float64 > divider
  c_float64 > result

  if. > pi
    i.less n
    seq
      result.write
        result.add
          1.0.div
            if.
              (i.mod 2).eq 0
              divider
              divider.neg

```

```
      divider.write (divider.add 2.0)
      i.write (i.add 1)
      pi

    seq
      result.write (result.mul 4.0)

seq > main
  n.write (args.get 0).toInt
  i.write 0
  divider.write 1.0
  result.write 0.0
  stdout (sprintf "pi[%d] = " n)
  pi
  stdout (sprintf "%f\n" result)
```

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

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 С, а также исходные тексты на ЕО, порождаемые транспилятором 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 С;
- каталог **src**, в который записывается файл **global.eo** с объектами, порожденными транспилятором в ходе анализа программы на С (он содержит объект **global**, в котором собраны все артефакты в виде соответствующих объектов).

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

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, ВМ) 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 транспилятора. 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 транспилятора.
- **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, предназначеный 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 кажом из который 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** - транспилятор, 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 транспилятором всех единиц компиляции он осуществляет компоновку из множества файлов, содержащих как статические, так и глобальные объекты, единый файл объектов на EO. Разработка данной программы может вестись на языке сценариев.

- **launcher** - каталог, содержащий программу, котора осуществляет многократный запуск транспилятора 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 C и EO, которые предполагается использовать как для интеграционного тестирования транспилятора, так и для проверки возможных вариантов трансформации в EO. Программы на C размещаются в подкаталоге ‘c-sources‘. Они формируют наборы данных, позволяющие оценить работоспособность разрабатываемого транспилятора. В подкаталоге ‘eo-sources‘ размещаются программы на EO, которые используются для анализа различных вариантов кодогенерации, а также для анализа возможности трансформации программ с C в EO.

2.2 Размещение программы на EO, полученной в ходе транспиляции

Представленная структура стала возможной из-за использования начальной инициализации объектов, имитирующих переменные языка C. Транспиляция 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 транспиляции всех единиц компиляции проекта. Сама сборка на текущий момент заключается в формировании общего файла на языке программирования EO. В нем формируется глобаальный 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 стороных 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 формирование 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:

```
#include <stdio.h>

int a = 0;
int b = 10;
int c = 30;

int main() {
    printf("Start: a = %d, b = %d, c = %d\n", a, b, c);
    a = b;
    b = c;
    c = a;
    printf("Finish: a = %d, b = %d, c = %d\n", a, b, c);
    return 0;
}
```

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

```
+package c2eo.src

+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf

+alias c2eo.ctypes.c_bool
+alias c2eo.ctypes.c_char
+alias c2eo.ctypes.c_float64
+alias c2eo.ctypes.c_int16
+alias c2eo.ctypes.c_int32
+alias c2eo.ctypes.c_int64

[arg] > global
  c_int64 0 > a
  c_int64 10 > b
  c_int64 30 > c

  [arg] > main
    seq > @
      stdout
        sprintf
          "Start: b = %s, c = %s\n"
          ^.b.as-string
          ^.c.as-string
```

```
      ^.a.write (^.b)
      ^.b.write (^.c)
      ^.c.write (^.a)
      stdout
        sprintf
          "Finish: b = %s, c = %s\n"
          ^.b.as-string
          ^.c.as-string
  main arg > @
```

Файл app.eo, запускающий сформированное приложение

```
+package c2eo

+alias stdout org.eolang.io.stdout
+alias sprintf org.eolang.txt.sprintf

+alias c2eo.src.global

[args...] > app
  global args > @
```