

## **Правила перевода программ с МИКРОЛИСПа на C++**

- 1) В начале целевой программы записывается директива `#include "mlisp.h"` .**
- 2) Константы**
  - 2.1 Вещественные и строковые константы переписываются без изменений.**
  - 2.2 Целые константы переводятся в вещественную форму. Например, -1, 1, 003 переписываются как -1. , 1. , 003. .**
  - 2.3 Булевские константы `#t` и `#f` заменяются `true` и `false`**
- 3) Идентификаторы.**
  - 3.1 -(дефис) заменяется на `__`(два подчеркика).**
  - 3.2 ! заменяется на `_E` .**
  - 3.3 ? заменяется на `_Q` .**
  - 3.4 Буквы переписываются с сохранением регистра.**
  - 3.5 Идентификаторы, совпадающие с ключевыми словами C++, «декорируются» префиксами `__xxx__` , `xxx` – Ваши инициалы, записанные латиницей.**

Например, идентификаторы `Good-enough`, `x!a`, `equal?`, `try` заменяются `Good__enough`, `x_Ea`, `equal_Q` , `__KPG__try` .

- 4) Типы.**
  - 4.1 Объектам, чьи имена оканчиваются знаком `?` , сопоставляются объекты типа `bool`.**
  - 4.2 Всем другим именованным объектам сопоставляются объекты типа `double`.**
- 5) Выражения.**

Каждому выражению МИКРОЛИСПа соответствует выражение C++.

  - 5.1 Вызов процедуры заменяется вызовом функции с тем же количеством аргументов.**

`(f a b) -> f(a,b)`  
`(g) -> g()`
  - 5.2 Арифметические выражения.**

Арифметические выражения заменяются арифметическими выражениями. При необходимости

**эквивалентный порядок вычислений задается приоритетами операторов и скобками.**

**5.2.1 (+ a b c) -> a + b + c**

**5.2.2 (- a (+ b c)) -> a - ( b + c)**

**5.2.3 (+ a) -> + a**

**5.2.4 (- a) -> - a**

**(- (+ a b)) -> - (a + b)**

**(- -1) -> - -1.**

**5.2.5 (\* a) -> a**

**В C++ нет одноместного оператора умножения!**

**5.2.6 (/ a) -> 1./ a**

**(/ (\* a b)) -> 1./ (a \* b)**

**(/(/ a)) -> 1./(1. / a)**

### **5.3 Логические выражения.**

**Логические выражения переводятся аналогично арифметическим.**

**5.3.1 (< a b) -> a < b**

**(= a b) -> a == b**

**5.3.2 (or a? b? c?) -> a\_Q || b\_Q || c\_Q**

**5.3.3 (and a? (or b? c?)) -> a\_Q && ( b\_Q || c\_Q)**

**5.3.4 (or a?) -> a\_Q**

**5.3.5 (and a?) -> a\_Q**

**5.3.6 (not a?) -> !a\_Q**

**(not (< a b)) -> !(a < b)**

### **5.4 Форма let.**

**Форма let переводится с помощью оператора последовательности(,). Эквивалент формы ОБЯЗАТЕЛЬНО обрамляется скобками.**

**(let()a b c) -> ( a , b , c )**

### **5.5 Форма if.**

**Форма if переводится с помощью условного оператора (? :). Эквивалент формы ОБЯЗАТЕЛЬНО обрамляется скобками.**

**(if x? a b) -> ( x\_Q ? a : b )**

### **5.6 Форма cond.**

**Форма cјnd переводится с помощью вложенных условных операторов (? :). Эквивалент формы ОБЯЗАТЕЛЬНО обрамляется скобками.**

**(cond( x? a)( y? b)(else 0)) ->**

```
( x_Q ? a
  : y_Q ? b
  : 0. )
```

**5.6.1** Если форма `cond` не содержит ветви `else`, то ее эквивалент завершается константой `_infinity`, определенное в файле `mlisp.h`.

```
(cond((< a b) a)
      (#t b) ) ->
( a < b ? a
  : true ? b
  : _infinity )
```

**5.6.2** Составные ветви `cond` переводятся с помощью оператора последовательности. Эквивалент составной клаузы можно записать без скобок, но эквивалент составной ветви `else` **ОБЯЗАТЕЛЬНО** обрамляется скобками.

```
(cond((< a b) (display "a ") a)
      (else (display "b ") b) ) ->
( a < b ? display("a "),a
  : ( display("b "),b ) )
```

**5.7** Форма `set!` переводится с помощью оператора присваивания.

```
(set! a b) -> a = b
```

**6) Определения.**

**6.1** Определениям процедур соответствуют определения функций C++ с тем же числом параметров.

**6.1.1** Типы функций и параметров определяются п.4 Правил.

**6.1.2** Эквивалент каждого выражения тела процедуры формирует отдельную инструкцию. Эквивалент последнего выражения помещается в инструкцию `return`.

**6.1.3** Порядок записи определений сохраняется. Для выполнения этого правила следует поместить в начале программы предварительные **ОБЪЯВЛЕНИЯ** всех функций.

```
(define (p? x? y?) (not(and x? y?)) )
```

```
(define (s x y)(set! x (+ x y))x )
```

перевод

```
#include "mlisp.h" .
```

```
bool p_Q(bool x_Q, bool y_Q); //объявления  
double s(double x, double y);
```

```
bool p_Q(bool x_Q, bool y_Q){ //определения  
    return !(x_Q && y_Q); }  
double s(double x, double y){  
    x = x + y;  
    return x; }
```

**6.2 Определения переменных соответствуют определения переменных C++.**

**6.2.1 Типы переменных определяются п.4 Правил.**

**6.2.2 Порядок записи определений сохраняется.**

**6.2.3 Определение, записанное вне тела функции, создает глобальную переменную.**

**6.2.3.1 Для каждой глобальной переменной в начале программы записывается предварительное ОБЪЯВЛЕНИЕ.**

**6.2.3.2 Инициализатор глобальной переменной отделяется от имени знаком = .**

```
(define eps 1e-5 )
```

```
(define count 0 )
```

перевод

```
#include "mlisp.h" .
```

```
extern double eps; //объявления
```

```
extern double count;
```

```
double eps = 1e-5; //определения
```

```
double count = 0.;
```

**6.2.4 Определение, записанное в теле функции, создает локальную переменную.**

**6.2.4.1 Инициализатор локальной переменной записывается после имени в скобках .**

```
(define (f x)
```

```
    (define eps 1e-5 )
```

```
    (+ x eps)
```

)

перевод

```
double f(double x){  
    double eps ( 1e-5 );  
    return x + eps;  
}
```

**NB!! Правила 6.2.3.2 и 6.2.4.1 предполагают, что инициализаторы глобальных и локальных переменных имеют разную форму записи при переводе на C++ .**

## **7) Вычисления.**

**Вычисление – это выражение, записанное вне определения процедуры или глобальной переменной.**

**7.1 Эквиваленты вычислений помещаются в теле функции main.**

**7.2 Если интерпретатор DrRacket печатает результат, то эквивалент вычисления «оборачивается» функциями display и newline, определенными в mlisp.h .**

```
(define a 0)  
pi  
(sin e)  
(set! a 7) :не печатает результат  
a
```

перевод

```
#include "mlisp.h"  
extern double a;  
double a=0.;  
int main(){  
    display(pi); newline();  
    display(sin(e)); newline();  
    a = 7.;  
    display(a); newline();  
    return 0;  
}
```

**В файле factorial.cpp записан перевод программы для вычисления факториала.**