**Лабораторная работа N1-3**

**Студент: Сорокин Д.М.**

**Группа: 8О-204Б**

**Номер по списку: 15**

**Вариант: 15/29**

**Дата: 3.03.2017**

**Тема: Знакомство с языком МИКРОЛИСП.**

**Отображение программ из МИКРОЛИСПа в С++.**

**Часть 1.**

**…………………………………………………………………………………………………………………………………….**

**Записать и выполнить в среде DrRacket программу проверки числа на простоту [1](стр 42). Программу дополнить определением составной процедуры xxxsquare, вычисляющей квадрат заданного числа.**

**Здесь ххх – персональный код студента.**

**Построить эквивалентную программу на языке С++, используя правила преобразования, записанные в файле**

**TranslationRules.doc .**

**Откомпилировать и выполнить программу в системе С++.**

**…………………………………………………………………………………………………………………………………….**

**;SDMlab2.ss**

**(define (smallest-divisor n)**

**(find-divisor n 2))**

**(define (SDMsquare n)**

**(\* n n))**

**(define (find-divisor n test-divisor)**

**(cond ((> (SDMsquare test-divisor) n) n)**

**((divides? test-divisor n) test-divisor)**

**(else (find-divisor n (+ test-divisor 1)))))**

**(define (divides? a b)**

**(= (remainder b a) 0))**

**(define (prime? n)**

**(= n (smallest-divisor n)))**

**(prime? 1)**

**(prime? 2)**

**(prime? 3)**

**(prime? 4)**

**(prime? 5)**

**(prime? 6)**

**(prime? 7)**

**(prime? 8)**

**(prime? 9)**

**(prime? 010.)**

**………………………………………………………………………………………………………………………………………….**

**//SDMlab1.cpp**

**#include "mlisp.h"**

**double smallest\_\_divisor(double n);**

**double SDMsquare(double n);**

**double find\_\_divisor(double n, double test\_\_divisor);**

**bool divides\_Q(double a, double b);**

**bool prime\_Q(double n);**

**double smallest\_\_divisor(double n) {**

**return find\_\_divisor(n, 2.);**

**}**

**double SDMsquare(double n) {**

**return n \* n;**

**}**

**double find\_\_divisor(double n, double test\_\_divisor) {**

**return (SDMsquare(test\_\_divisor) > n ? n**

**: divides\_Q(test\_\_divisor, n) ? test\_\_divisor**

**: find\_\_divisor(n, ++test\_\_divisor));**

**}**

**bool divides\_Q(double a, double b) {**

**return remainder(b, a) == 0.;**

**}**

**bool prime\_Q(double n) {**

**return (n == smallest\_\_divisor(n));**

**}**

**int main() {**

**display("Primes or not:"); newline();**

**display(prime\_Q(1)); newline();**

**display(prime\_Q(2)); newline();**

**display(prime\_Q(3)); newline();**

**display(prime\_Q(4)); newline();**

**display(prime\_Q(5)); newline();**

**display(prime\_Q(6)); newline();**

**display(prime\_Q(7)); newline();**

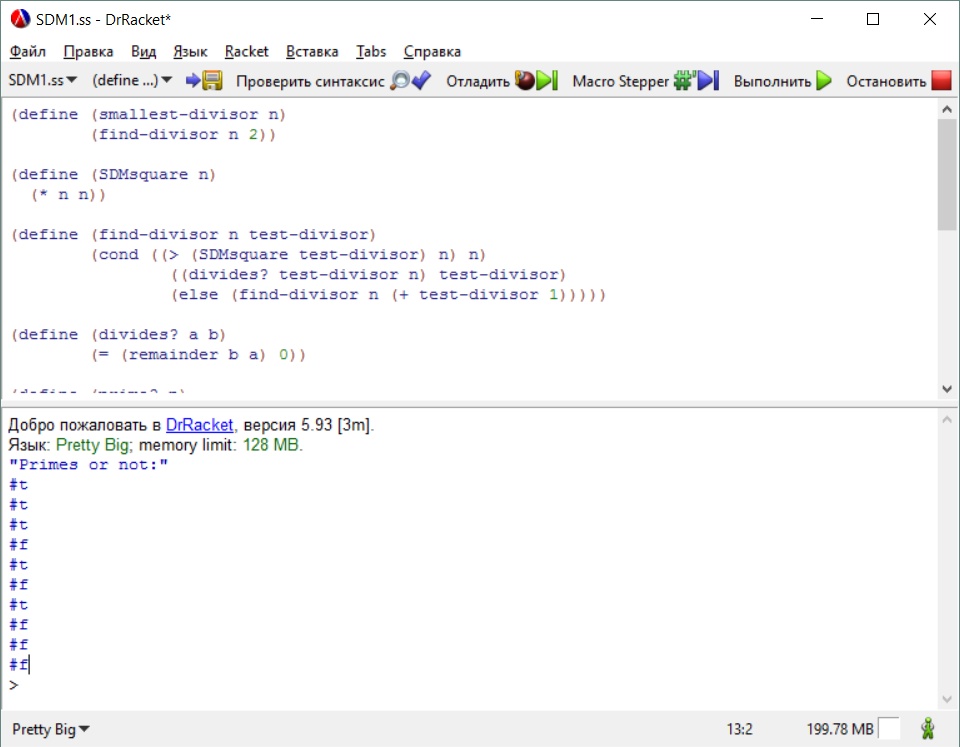
**display(prime\_Q(8)); newline();**

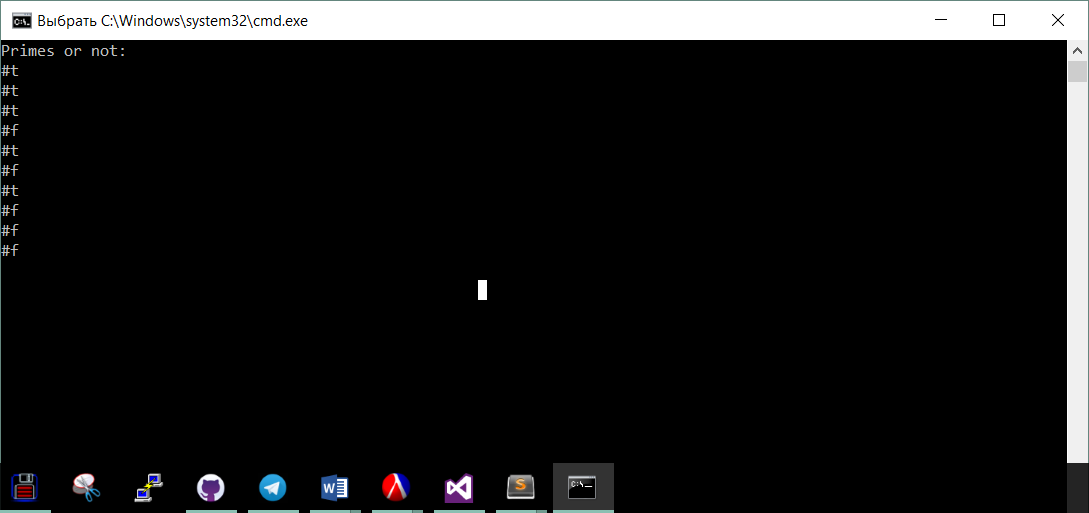
**display(prime\_Q(9)); newline();**

**display(prime\_Q(010.)); newline();**

**std::cin.get();**

**}**





**………………………………………………………………………………………………………………………………………….**

**Часть 2.**

**………………………………………………………………………………………………………………………………………….**

**Вариант 29:**



**Доработать учебный тренажер half-interval.ss, заменив определение процедуры fun, и решить методом половинного деления заданное уравнение.**

**Составить и протестировать эквивалентную программу на С++.**

**………………………………………………………………………………………………………………………………………….**

**; SDMlab2.ss**

**(define (half-interval-metod a b)**

**(define a-value 0)**

**(define b-value 0)**

**(set! a-value (fun a))**

**(set! b-value (fun b))**

**(cond((and(< a-value 0)(> b-value 0))**

**(try a b))**

**((and(> a-value 0)(< b-value 0))**

**(try b a))**

**(else(+ b 1))**

**)**

**)**

**(define(try neg-point pos-point)**

**(define midpoint 0)**

**(define test-value 0)**

**(set! midpoint (average neg-point pos-point))**

**(display "+")**

**(cond((close-enough? neg-point pos-point) midpoint)**

**(#t (set! test-value (fun midpoint))**

**(cond((> test-value 0)(try neg-point midpoint))**

**((< test-value 0)(try midpoint pos-point))**

**(else midpoint))**

**)**

**)**

**)**

**(define (close-enough? x y)**

**(<(abs (- x y))tolerance))**

**(define (average x y)(/(+ x y)2.))**

**(define (root a b)**

**(define temp 0)**

**(set! temp(half-interval-metod a b))**

**(newline)**

**(display"interval=\t[")**

**(display a)**

**(display" , ")**

**(display b)**

**(display"]\n")**

**(display"discrepancy=\t")**

**(display(fun temp))(newline)**

**(display"root=\t\t")**

**(display(if(=(- temp b 1)0)"[bad]" "[good]"))**

**temp**

**)**

**(define tolerance 0.00001)**

**(define(fun z)**

**(set! z (- z (/ 29 30)(/ pi)))**

**;========================================================**

**;C++: z - atan(2 \* z) - log(pi \* z) - 6**

**;========================================================**

**(- z**

**(atan (\* 2 z))**

**(log (\* pi z))**

**6)**

**)**

**"SDM variant 29"**

**(root 12 13**

**………………………………………………………………………………………………………………………………………….**

**#include "mlisp.h"**

**double half\_\_interval\_\_method(double a, double b);**

**double \_\_SDM\_\_try(double neg\_\_point, double pos\_\_point);**

**bool close\_\_enough\_Q(double x, double y);**

**double average(double x, double y);**

**double root(double a, double b);**

**extern double tolerance;**

**double fun(double z);**

**double half\_\_interval\_\_method(double a, double b) {**

**{**

**double a\_\_value = 0;**

**double b\_\_value = 0;**

**a\_\_value = fun(a);**

**b\_\_value = fun(b);**

**return ((a\_\_value < 0) && (b\_\_value > 0) ? \_\_SDM\_\_try(a, b)**

**: (a\_\_value > 0) && (b\_\_value < 0) ? \_\_SDM\_\_try(b, a)**

**: b + 1);**

**}**

**}**

**double \_\_SDM\_\_try(double neg\_\_point, double pos\_\_point) {**

**{**

**double midpoint = 0;**

**double test\_\_value = 0;**

**midpoint = average(neg\_\_point, pos\_\_point);**

**display("+");**

**return (close\_\_enough\_Q(neg\_\_point, pos\_\_point) ? midpoint**

**: true ? (test\_\_value = fun(midpoint),**

**(test\_\_value > 0 ? \_\_SDM\_\_try(neg\_\_point, midpoint)**

**: test\_\_value < 0 ? \_\_SDM\_\_try(midpoint, pos\_\_point)**

**: midpoint))**

**: \_infinity);**

**}**

**}**

**bool close\_\_enough\_Q(double x, double y) {**

**{**

**return \_\_SDM\_\_abs(x - y) < tolerance;**

**}**

**}**

**double average(double x, double y) {**

**return (x + y) / 2.;**

**}**

**double root(double a, double b) {**

**double temp = 0;**

**temp = half\_\_interval\_\_method(a, b);**

**newline();**

**display("interval=\t[");**

**display(a);**

**display(" , ");**

**display(b);**

**display("]\n");**

**display("discrepancy=\t");**

**display(fun(temp));**

**newline();**

**display("root=\t\t");**

**display((temp - b - 1) == 0 ? "[bad]" : "[good]");**

**return temp;**

**}**

**double tolerance = 0.00001;**

**double fun(double z) {**

**z = z - (29. / 30.) - (1. / pi);**

**return z - atan(2. \* z) - log(pi \* z) - 6.;**

**}**

**int main() {**

**display("SDM variant 29");**

**newline();**

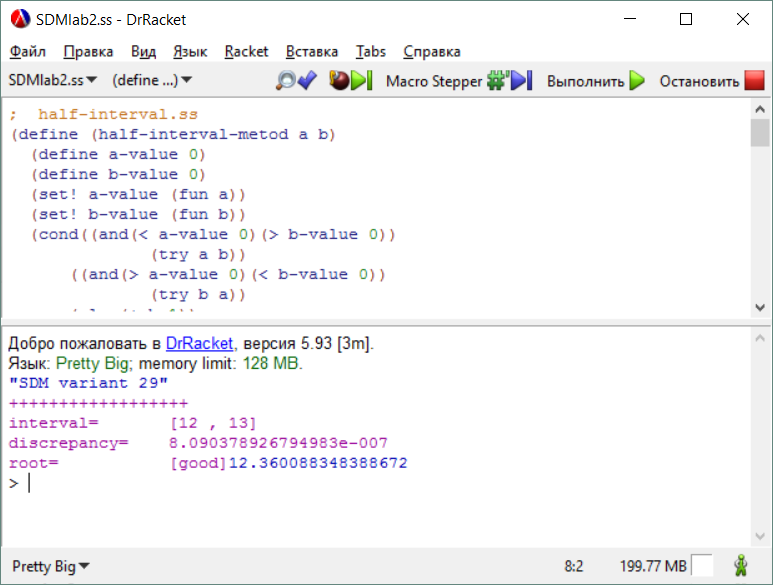
**display(root(12., 13.)); newline();**

**std::cin.get();**

**return 0;**

**}**

**………………………………………………………………………………………………………………………………………….**





**………………………………………………………………………………………………………………………………………….**

**Часть 3.**

**………………………………………………………………………………………………………………………………………….**

**Вариант 15:**

**5-10-15-20-25-50 ; Ответ для 100: 223**

**А) Изучить тренажер coin.cpp , в котором на языке С++**

**записана программа размена денег [1](стр.33-35).**

**В) Изменить программу в соответствии с вариантом задания**

**(файл var-coins.txt)**

**C) Декомпилировать программу в язык МИКРОЛИСП.**

**D) Сравнить результаты выполнения программ на МИКРОЛИСПе и С++.**

**………………………………………………………………………………………………………………………………………….**

**//coin.cpp**

**#include "mlisp.h"**

**double cc(double amount, double largest\_\_coin);**

**double count\_\_change(double amount);**

**double next\_\_coin(double coin);**

**double GR\_\_AMOUNT();**

**extern double LAGEST\_\_COIN;**

**extern double LAST\_\_DIGIT\_\_OF\_\_GROUP\_\_NUMBER;**

**double cc(double amount, double largest\_\_coin){**

**{return**

**( (amount == 0 || largest\_\_coin == 1.) ? 1.**

**: !(amount > 0 && largest\_\_coin > 0) ? 0**

**: (cc(amount, next\_\_coin(largest\_\_coin)) +**

**cc(amount - largest\_\_coin, largest\_\_coin)**

**)**

**);**

**}}**

**double count\_\_change(double amount){**

**{return**

**cc(amount, LAGEST\_\_COIN);**

**}}**

**double next\_\_coin(double coin){**

**{return**

**( coin ==50. ? 25.**

**: coin == 25. ? 20.**

**: coin == 20. ? 15.**

**: coin == 15. ? 5.**

**: coin == 5. ? 2.**

**: 0**

**);**

**}}**

**double GR\_\_AMOUNT(){{**

**return 10. \* LAST\_\_DIGIT\_\_OF\_\_GROUP\_\_NUMBER;**

**}}**

**double LAGEST\_\_COIN=50.;**

**double LAST\_\_DIGIT\_\_OF\_\_GROUP\_\_NUMBER=4.;**

**int main(){**

**display(" xxx variant 0");newline();**

**display(" 3-5-20");newline();**

**display("count\_\_change for 100 \t= ");**

**display(count\_\_change(100.));newline();**

**display("count\_\_change for ");**

**display(GR\_\_AMOUNT());**

**display(" \t= ");**

**display(count\_\_change(GR\_\_AMOUNT()));newline();**

**std::cin.get();**

**return 0;**

**}**

**………………………………………………………………………………………………………………………………………….**

**; coin.ss**

**(define (cc amount largest-coin)**

**(cond ((or (= amount 0) (= largest-coin 1)) 1)**

**((not(and(> amount 0) (> largest-coin 0))) 0)**

**(else(+(cc amount (next-coin largest-coin))**

**(cc(- amount largest-coin) largest-coin)))))**

**(define (count-change amount)**

**(cc amount LAGEST-COIN))**

**(define (next-coin coin)**

**(cond((= coin 50) 25)**

**((= coin 25) 20)**

**((= coin 20) 15)**

**((= coin 15) 10)**

**((= coin 10) 5)**

**(else 0)))**

**(define (GR-AMOUNT)**

**(\* 10 LAST-DIGIT-OF-GROUP-NUMBER))**

**(define LAGEST-COIN 50)**

**(define LAST-DIGIT-OF-GROUP-NUMBER 4)**

**(display "SDM variant 15")**

**(newline)**

**(display "5-10-15-20-25-50")**

**(newline)**

**(display "count-change for 100 \t= ")**

**(display (count-change 100))**

**(newline)**

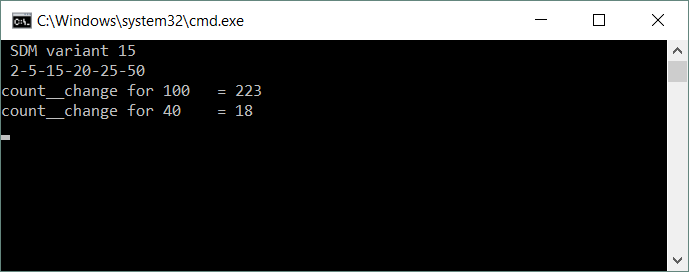
**(display "count-change for ")**

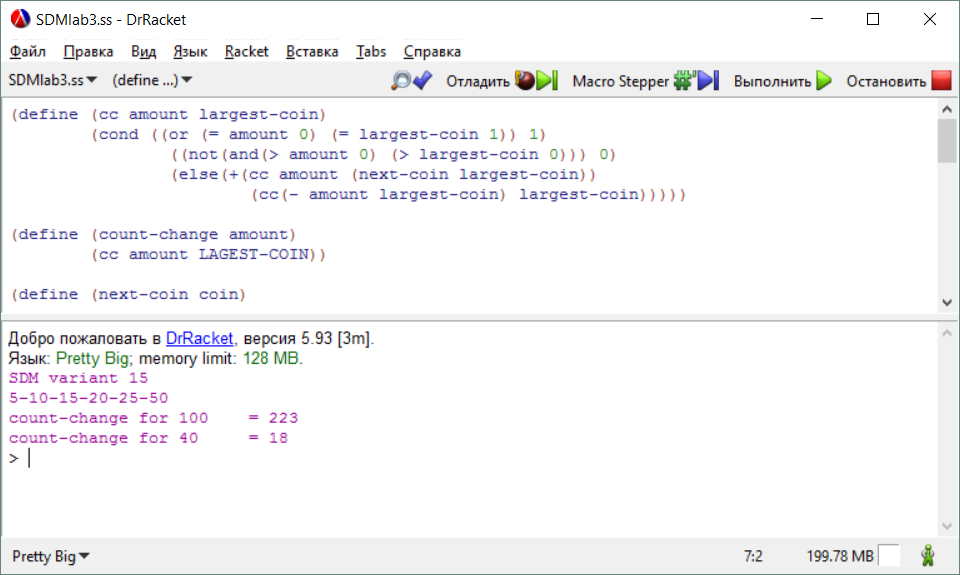
**(display (GR-AMOUNT))**

**(display " \t= ")**

**(display (count-change (GR-AMOUNT)))**

**………………………………………………………………………………………………………………………………………….**





**………………………………………………………………………………………………………………………………………….**

**Выводы.**

**В ходе выполнения лабораторных работ 1-3, мною были изучены базовые навыки работы с языком МИКРОЛИСП, а также отображением программ на нем в язык С++ (с которым я уже был знаком ранее).**

**МИКРОЛИСП мне показался довольно интересным языком, потому что для работы с ним нужно немного изменить мышление. В нем выражения пишутся не в привычной для многих людей инфиксной (нормальной) нотации, а в префиксной. Также порадовал факт, что можно пользоваться такими конструкциями как (- a b c d) = a – b – c – d, что в ряде случаев упрощает написание программы.**

**Однако знакомство с этим чудесным языком не единственная задача данных лабораторных работ. Главной их частью является отображение программ с одного языка на другой и обратно.**

**В первых двух работах я «переводил» с МИКРОЛИСПа на С++. А в третьей в обратную сторону. Все преобразования проводились строго по определенным правилам. Такая работа довольно интересна, т.к. показывает, что один и тот же алгоритм может быть реализован на разных языках, но с абсолютно идентичным результатом.**

**Таким образом, полученный опыт пригодиться мне в будущих лабораторных работах и написании компилятора.**