Programa de teste 0 (sample)

```
\\ UFRGS - Compiladores - Marcelo Johann - 2021/2

char c: 'x';
char d: 100;
int a: 'A';
int i: 1;
int v[10]: 'a' 0 0 0 0 0 0 0 0 0;
int matrix[100];
float f: 2/3;

\*
This is a comment
of multiple lines
*\

int main ()
 {
 a = 0;
 a = a - i;
 a = 5;
 v[a] = 55;
 print v[5];
 print a;
 i = 2;

 print "Digite um numero: \n";
 a = read;

 while i<10
   {
   i = incn(i,1);
   a = incn(a,1);
   };

 print "Incrementado algumas vezes a fica " , a ,"\n";

 if a==15 then
   {
   label-x:
   a = a - 1;
   print "A era=15\n";
   };
```

```
 if (i==100) then
   {
   print "Nao tem como isso....\n";
   }
 else
   print "OK!\n";
 if a > 0 then
   goto label-x;
 }

int incn (int x , int n )
 {
 return x+n;
 }

\\ end of file
```

Resultado teste 0

```
555Digite um numero:
15
Incrementado algumas vezes a fica 23
OK!
```

Programa de teste 1

```
int input-one: 0;
int input-two: 0;
int result: 0;

int main() {
   print "Teste 1:";

   print "Atribuições e operações aritmeticas com impressão:  [read, +, -,
*, /] \n";

   print "\n", "Enter the first value: ";
   input-one = read;
   print "Enter the second value: ";
   input-two = read;

   print "\n", "input1 + input2 = ", input-one + input-two;
   print "\n", "input1 - input2 = ", input-one - input-two;
   print "\n", "input1 * input2 = ", input-one * input-two;
   print "\n", "input1 / input2 = ", input-one / input-two, "\n";
}
```

Resultado teste 1

```
Teste 1:Atribuições e operações aritmeticas com impressão:  [read, +, -, *,
/]

Enter the first value: 36
Enter the second value: 4

input1 + input2 = 40
input1 - input2 = 32
input1 * input2 = 144
input1 / input2 = 9
```

Os logs do programa estão no arquivo test_programs/test_1.logs

Programa de teste 2

```
int input-one: 0;
int input-two: 0;

int result-fac: 0;

int a: 0;
int b: 1;

int result-fib: 0;

int result: 0;
int n: 0;

int fatorial(int argsa) {
   n = argsa;
   result = 1;
   while(n > 1){
       result = result * n;
       n = n - 1;
   };

   return result;
}


int fibonacci(int argsb) {
   n = argsb;

   if(n == 0) then {
       return 0;
   };

   if(n == 1) then {
       return 1;
   };

   result = 1;
   while(n > 1){
       result = a + b;
       a = b;
       b = result;
       n = n - 1;
```

```
    };

    return result;
}

int main() {
    print "Teste 1:";

    print "Calculo fatorial e fubinacci, [read, while, if, function call]
\n";

    print "\n", "Enter the first value: ";
    input-one = read;
    print "Enter the second value: ";
    input-two = read;

    result-fac = fatorial(input-one);
    result-fib = fibonacci(input-two);

    print "\nO valor de ", input-one, "! é ", result-fac;

    if(result-fac > result-fib) then {
        print " que é maior do que o ";
    } else {
        print " que é menor do que o ";
    };

    print input-two, "º numero de fibonacci, que é ", result-fib, "\n";

    a = result-fac - result-fib;
    b = 0;

    while(a > 0) {
        b = b + 1;
        a = a - result-fib;
    };

    if(0 < b) then {
        print "\nO fatorial de ", input-one, " é, pelo menos, ", b, "x maior
que o ", input-two, "º numero de fibonacci.\n";
    };

}
```

Resultado teste 2

```
Teste 1:Calculo fatorial e fubinacci, [read, while, if, function call]

Enter the first value: 7
Enter the second value: 12

O valor de 7! é 5040 que é maior do que o 12º numero de fibonacci, que é
144

O fatorial de 7 é, pelo menos, 34x maior que o 12º numero de fibonacci.
```

Os logs do programa estão no arquivo test_programs/test_2.logs

Programa de teste 3

```
int input: 0;

int result: 0;
int n: 0;
int a: 0;
int b: 0;

int total_n: 30;
int arr[20];

int columns: 6;
int count: 0;

int fibonacci(int argsb) {

   a = 0;
   b = 1;

   if(argsb == 0) then {
       return 0;
   };

   if(argsb == 1) then {
       return 1;
   };

   result = 1;
   while(argsb > 1){
       result = a + b;
       a = b;
       b = result;
       argsb = argsb - 1;
   };

   return result;
}

int main() {
   print "Teste 3:";

   print "Arrays \n";
```

```
   print "\n", "Enter a number: ";
   input = read;

   n = 0;

   while(n < total_n) {
       arr[n] = fibonacci(n);
       \\print "\n", n, ": ", arr[n];
       n = n+1;
   };

   n = 0;

   print "\nOs ",total_n, " primeiros numeros da sequencia de fibonacci
são: \n";
   while(n < total_n) {
       arr[n] = fibonacci(n);

       if input == arr[n] then print "[->", arr[n], "<-] " else print
arr[n], "\t";

       if input > arr[n] then if input < arr[n+1] then print "[*] ";

       n = n+1;

       if(count >= columns) then {
           print "\n";
           count = 0;
       } else {
           count = count + 1;
       };
   };

   print "\n";

}
```

Resultado teste 3

```
Teste 3:Arrays
Enter a number: 610

Os 30 primeiros numeros da sequencia de fibonacci são:
0       1       1       2       3       5       8
13      21      34      55      89      144     233
377     [->610<-] 987   1597    2584    4181    6765
10946   17711   28657   46368   75025   121393  196418
317811  514229
```

Os logs do programa estão no arquivo test_programs/test_2.logs

Programa de teste 4

```
int in-repeat: 0;
char first-char: 'a';

int result: 0;
int n: 0;

float fa: 2/3;
float fb: 5/6;

int count: 0;


int repeatchar(int repeat, char caracter, float argsfa, float argsfb) {


   print "As proximas ", repeat, " letras do alfabeto depois de [",
caracter, "] são: \n";

   while(count < repeat){
       print caracter, ", ";
       count = count + 1;
       caracter = caracter + 1;
   };

   print "\n\n O terceiro argumento é um float com valor: ", argsfa;
   print "\n\n O quarto argumento é um float com valor: ", argsfb, "\n";

   return result;
}

int main() {
   print "Teste 4:";

   print "tipos de dados e chamadas com multiplos argumentos [int, char,
float] \n";

   print "\n", "Digite o numero de letras: ";
   in-repeat = read;

   return repeatchar(in-repeat, first-char, fa, fb);
}
```

Resultado teste 4

```
Teste 4:tipos de dados e chamadas com multiplos argumentos [int, char,
float]

Digite o numero de letras: 21
As proximas 21 letras do alfabeto depois de [a] são:
a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u,

 O terceiro argumento é um float com valor: 2/3

 O quarto argumento é um float com valor: 5/6
```

Os logs do programa estão no arquivo test_programs/test_4.logs

Código Gerado

Programa de teste 0 (sample)

```
   .data
__TMP_VAR_11: .string  "\n"
__TMP_VAR_4: .long 0
_0: .long  0
_1: .long  1
_2: .long  2
_3: .long  3
_5: .long  5
__TMP_VAR_15: .string  "A era=15\n"
_a: .long  0
__TMP_VAR_20: .long    0
_c: .byte  120
_d: .byte  48
_f:    .long  2
  .long  3
_i: .long  1
_n: .long  0
__TMP_VAR_3: .long 0
_x: .long  0
__TMP_VAR_19: .long    0
__TMP_VAR_2: .string   "Digite um numero: \n"
__TMP_VAR_1: .long 0
__TMP_VAR_12: .string  "Incrementado algumas vezes a fica "
__TMP_VAR_0: .long 0
__TMP_VAR_16: .long    0
_10: .long 10
__TMP_VAR_13: .long    0
_100: .long    100
__TMP_VAR_17: .string  "Nao tem como isso....\n"
__TMP_VAR_9: .long 0
_15: .long 15
__TMP_VAR_8: .long 0
__TMP_VAR_10: .long    0
__TMP_VAR_7: .long 0
__TMP_VAR_14: .long    0
_55: .long 55
__TMP_VAR_6: .long 0
__TMP_VAR_18: .string  "OK!\n"
__TMP_VAR_5: .long 0
```

```
# PRINT
print_string_int:
  .string    "%d"

print_string_float:
  .string    "%d/%d"

print_string_char:
  .string    "%c"
print_string:
  .string    "%s"


#READ
read:
  .string    "%d"



# TAC_ARRAY
_v:
   .long 'a'     # v[0]
   .long 0   # v[1]
   .long 0   # v[2]
   .long 0   # v[3]
   .long 0   # v[4]
   .long 0   # v[5]
   .long 0   # v[6]
   .long 0   # v[7]
   .long 0   # v[8]
   .long 0   # v[9]

# TAC_BEGINFUN
  .text
  .globl main
main:
  pushq  %rbp
  movq   %rsp, %rbp


# TAC_MOVE //ASSIGN
   movl   _0(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax
```

```
# TAC_SUB
   movl   _a(%rip), %edx
   movl   _i(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_0(%rip)
   movl   $0, %eax



# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_0(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax



# TAC_MOVE //ASSIGN
   movl   _5(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax



# TAC_ARRAY SET_ELEMENT EXP
  movl   _a(%rip), %edx   #endereço
  movl   _55(%rip), %eax   #valor
  movslq %edx, %rdx
  leaq   0(,%rdx,4), %rcx
  leaq   _v(%rip), %rdx   #array
  movl   %eax, (%rcx,%rdx)

# TAC_ARRAY GET_ELEMENT
   movl   20+_v(%rip), %eax
   movl   %eax, __TMP_VAR_1(%rip)
   movl   $0, %eax
```

```
# TAC_PRINT_INT
   movl   __TMP_VAR_1(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT


# TAC_PRINT_INT
   movl   _a(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_MOVE //ASSIGN
   movl   _2(%rip), %eax
   movl   %eax, _i(%rip)
   movl   $0, %eax




# TAC_PRINT_STRING
   leaq   __TMP_VAR_2(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
  leaq   __TMP_VAR_3(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
  call   __isoc99_scanf@PLT

# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_3(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax
```

```
# TAC_LABEL
   __TMP_LABEL_0: # AUTO



# TAC_LT
  movl   _i(%rip), %edx
  movl   _10(%rip), %eax
  cmpl   %eax, %edx
  jl CMP_LBL_TEMP_0        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_4(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_1        # e pula para o final do bloco

CMP_LBL_TEMP_0:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_4(%rip)

CMP_LBL_TEMP_1:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_4(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_1


# TAC_TAC_FUN_CALL_ARGS

  movl   _1(%rip), %r10d
  pushq  %r10

  movl   _i(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   incn
  addq   $16, %rsp
  movl   %eax, __TMP_VAR_7(%rip) #move return to tempvar
```

```
# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_7(%rip), %eax
   movl   %eax, _i(%rip)
   movl   $0, %eax



# TAC_TAC_FUN_CALL_ARGS

  movl   _1(%rip), %r10d
  pushq  %r10

  movl   _a(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   incn
  addq   $16, %rsp
  movl   %eax, __TMP_VAR_10(%rip) #move return to tempvar


# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_10(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax



# TAC_JMP
  jmp    __TMP_LABEL_0


# TAC_LABEL
   __TMP_LABEL_1: # AUTO


# TAC_PRINT_STRING
   leaq   __TMP_VAR_12(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
```

```
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _a(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_11(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_EQ
  movl   _a(%rip), %edx
  movl   _15(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_2        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_13(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_3        # e pula para o final do bloco

CMP_LBL_TEMP_2:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_13(%rip)

CMP_LBL_TEMP_3:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_13(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_3


# TAC_LABEL
   __TMP_LABEL_2: # label__dash__x
```

```
# TAC_SUB
   movl   _a(%rip), %edx
   movl   _1(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_14(%rip)
   movl   $0, %eax



# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_14(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_15(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_3: # AUTO



# TAC_EQ
  movl   _i(%rip), %edx
  movl   _100(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_4        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_16(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_5        # e pula para o final do bloco

CMP_LBL_TEMP_4:        #se condição verdadeira seta 1
```

```
  movl   $1, __TMP_VAR_16(%rip)

CMP_LBL_TEMP_5:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_16(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_4


# TAC_PRINT_STRING
   leaq   __TMP_VAR_17(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_JMP
  jmp    __TMP_LABEL_5



# TAC_LABEL
   __TMP_LABEL_4: # AUTO



# TAC_PRINT_STRING
   leaq   __TMP_VAR_18(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_5: # AUTO
```

```
# TAC_GT
  movl   _a(%rip), %edx
  movl   _0(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_6        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_19(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_7        # e pula para o final do bloco

CMP_LBL_TEMP_6:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_19(%rip)

CMP_LBL_TEMP_7:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_19(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_6


# TAC_LABEL
   __TMP_LABEL_6: # AUTO



## TAC_ENDFUN
  popq   %rbp
  ret



# TAC_BEGINFUN
  .text
  .globl incn
incn:
  pushq  %rbp
  movq   %rsp, %rbp

  # TAC_DEC_FUNC_ARGS
   movl   $0, %edx
```

```
  movl   16(%rbp), %eax
  movl   %eax, _x(%rip)
  movl   24(%rbp), %eax
  movl   %eax, _n(%rip)

# TAC_ADD
   movl   _x(%rip), %edx
   movl   _n(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_20(%rip)
   movl   $0, %eax



## TAC_ENDFUN
  movl   __TMP_VAR_20(%rip), %eax #return
  popq   %rbp
  ret
```

Programa de teste 1

```
    .data
__TMP_VAR_11: .string  "input1 - input2 = "
__TMP_VAR_4: .long 0
_0: .long  0
__TMP_VAR_15: .string  "\n"
__TMP_VAR_3: .string   "\n"
__TMP_VAR_19: .string  "\n"
_input__dash__one: .long   0
__TMP_VAR_2: .string   "Enter the first value: "
__TMP_VAR_1: .string   "Atribuições e operações aritmeticas com impressão:
[read, +, -, *, /] \n"
__TMP_VAR_12: .string  "\n"
__TMP_VAR_0: .string   "Teste 1:"
__TMP_VAR_16: .long    0
__TMP_VAR_13: .long    0
__TMP_VAR_17: .string  "\n"
__TMP_VAR_9: .string   "\n"
_input__dash__two: .long   0
__TMP_VAR_8: .string   "input1 + input2 = "
_result: .long 0
__TMP_VAR_10: .long    0
__TMP_VAR_7: .long 0
__TMP_VAR_14: .string  "input1 * input2 = "
__TMP_VAR_6: .long 0
__TMP_VAR_18: .string  "input1 / input2 = "
__TMP_VAR_5: .string   "Enter the second value: "

# PRINT
print_string_int:
  .string    "%d"

print_string_float:
  .string    "%d/%d"

print_string_char:
  .string    "%c"
print_string:
  .string    "%s"


#READ
read:
  .string    "%d"
```

```
# TAC_BEGINFUN
  .text
  .globl main
main:
  pushq  %rbp
  movq   %rsp, %rbp


# TAC_PRINT_STRING
   leaq   __TMP_VAR_0(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_1(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_3(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_2(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
```

```
  leaq   __TMP_VAR_4(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
  call   __isoc99_scanf@PLT

# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_4(%rip), %eax
   movl   %eax, _input__dash__one(%rip)
   movl   $0, %eax




# TAC_PRINT_STRING
   leaq   __TMP_VAR_5(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
  leaq   __TMP_VAR_6(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
  call   __isoc99_scanf@PLT

# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_6(%rip), %eax
   movl   %eax, _input__dash__two(%rip)
   movl   $0, %eax




# TAC_PRINT_STRING
   leaq   __TMP_VAR_9(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
```

```
    leaq    __TMP_VAR_8(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax



# TAC_ADD
    movl    _input__dash__one(%rip), %edx
    movl    _input__dash__two(%rip), %eax
    addl    %edx, %eax
    movl    %eax, __TMP_VAR_7(%rip)
    movl    $0, %eax



# TAC_PRINT_INT
    movl    __TMP_VAR_7(%rip), %esi    # mov a to reg
    #movl   %eax, %esi
    leaq    print_string_int(%rip), %rdi
   call    printf@PLT



# TAC_PRINT_STRING
    leaq    __TMP_VAR_12(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax



# TAC_PRINT_STRING
    leaq    __TMP_VAR_11(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax



# TAC_SUB
    movl    _input__dash__one(%rip), %edx
    movl    _input__dash__two(%rip), %eax
    subl    %eax, %edx
```

```
    movl    %edx, %eax
    movl    %eax, __TMP_VAR_10(%rip)
    movl    $0, %eax



# TAC_PRINT_INT
    movl    __TMP_VAR_10(%rip), %esi    # mov a to reg
    #movl   %eax, %esi
    leaq    print_string_int(%rip), %rdi
   call    printf@PLT



# TAC_PRINT_STRING
    leaq    __TMP_VAR_15(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax



# TAC_PRINT_STRING
    leaq    __TMP_VAR_14(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax



# TAC_MULT
    movl    _input__dash__one(%rip), %edx
    movl    _input__dash__two(%rip), %eax
    imull   %edx, %eax
    movl    %eax, __TMP_VAR_13(%rip)
    movl    $0, %eax



# TAC_PRINT_INT
    movl    __TMP_VAR_13(%rip), %esi    # mov a to reg
    #movl   %eax, %esi
    leaq    print_string_int(%rip), %rdi
   call    printf@PLT
```

```
# TAC_PRINT_STRING
   leaq   __TMP_VAR_19(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_18(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_DIV
   movl   _input__dash__one(%rip), %eax
   movl   _input__dash__two(%rip), %ecx
           cltd
   idivl  %ecx
   movl   %eax, __TMP_VAR_16(%rip)
   movl   $0, %eax




# TAC_PRINT_INT
   movl   __TMP_VAR_16(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_17(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax
```

```
## TAC_ENDFUN
  popq   %rbp
  ret
```

Programa de teste 2

```
   .data
__TMP_VAR_11: .string  "\n"
__TMP_VAR_4: .long 0
__TMP_VAR_25: .string  "º numero de fibonacci, que é "
_0: .long  0
_1: .long  1
__TMP_VAR_15: .long    0
_a: .long  0
_b: .long  1
__TMP_VAR_20: .string  "\nO valor de "
_n: .long  0
__TMP_VAR_3: .long 0
__TMP_VAR_19: .string  "! é "
_input__dash__one: .long   0
__TMP_VAR_2: .long 0
__TMP_VAR_26: .long    0
__TMP_VAR_1: .long 0
_argsa: .long  0
__TMP_VAR_21: .long    0
__TMP_VAR_12: .long    0
__TMP_VAR_0: .long 0
_result__dash__fac: .long  0
__TMP_VAR_16: .long    0
__TMP_VAR_27: .long    0
__TMP_VAR_30: .long    0
__TMP_VAR_22: .string  " que é maior do que o "
__TMP_VAR_31: .string  "º numero de fibonacci.\n"
__TMP_VAR_13: .string  "Enter the second value: "
__TMP_VAR_28: .long    0
__TMP_VAR_17: .long    0
__TMP_VAR_9: .string   "Calculo fatorial e fubinacci, [read, while, if,
function call] \n"
__TMP_VAR_32: .string  "x maior que o "
__TMP_VAR_23: .string  " que é menor do que o "
_input__dash__two: .long   0
__TMP_VAR_8: .string   "Teste 1:"
_result: .long 0
__TMP_VAR_10: .string  "Enter the first value: "
__TMP_VAR_33: .string  " é, pelo menos, "
_argsb: .long  0
__TMP_VAR_29: .long    0
__TMP_VAR_7: .long 0
_result__dash__fib: .long  0
```

```
__TMP_VAR_14: .long    0
__TMP_VAR_24: .string  "\n"
__TMP_VAR_6: .long 0
__TMP_VAR_18: .long    0
__TMP_VAR_34: .string  "\nO fatorial de "
__TMP_VAR_5: .long 0

# PRINT
print_string_int:
  .string    "%d"

print_string_float:
  .string    "%d/%d"

print_string_char:
  .string    "%c"
print_string:
  .string    "%s"


#READ
read:
  .string    "%d"



# TAC_BEGINFUN
  .text
  .globl fatorial
fatorial:
  pushq  %rbp
  movq   %rsp, %rbp

  # TAC_DEC_FUNC_ARGS
   movl   $0, %edx

  movl   16(%rbp), %eax
  movl   %eax, _argsa(%rip)

# TAC_MOVE //ASSIGN
   movl   _argsa(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax
```

```
# TAC_MOVE //ASSIGN
   movl   _1(%rip), %eax
   movl   %eax, _result(%rip)
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_0: # AUTO



# TAC_GT
  movl   _n(%rip), %edx
  movl   _1(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_0        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_0(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_1        # e pula para o final do bloco

CMP_LBL_TEMP_0:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_0(%rip)

CMP_LBL_TEMP_1:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_0(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_1


# TAC_MULT
   movl   _result(%rip), %edx
   movl   _n(%rip), %eax
   imull  %edx, %eax
   movl   %eax, __TMP_VAR_1(%rip)
```

```
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_1(%rip), %eax
   movl   %eax, _result(%rip)
   movl   $0, %eax




# TAC_SUB
   movl   _n(%rip), %edx
   movl   _1(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_2(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_2(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax




# TAC_JMP
  jmp    __TMP_LABEL_0



# TAC_LABEL
   __TMP_LABEL_1: # AUTO


## TAC_ENDFUN
  movl   _result(%rip), %eax #return
  popq   %rbp
```

```
  ret



# TAC_BEGINFUN
  .text
  .globl fibonacci
fibonacci:
  pushq  %rbp
  movq   %rsp, %rbp

  # TAC_DEC_FUNC_ARGS
   movl   $0, %edx

  movl   16(%rbp), %eax
  movl   %eax, _argsb(%rip)

# TAC_MOVE //ASSIGN
   movl   _argsb(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax




# TAC_EQ
  movl   _n(%rip), %edx
  movl   _0(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_2        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_3(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_3        # e pula para o final do bloco

CMP_LBL_TEMP_2:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_3(%rip)

CMP_LBL_TEMP_3:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_3(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
```

```
baixo
   jz __TMP_LABEL_2

## TAC_ENDFUN
  movl   _0(%rip), %eax #return
  popq   %rbp
  ret



# TAC_LABEL
   __TMP_LABEL_2: # AUTO



# TAC_EQ
  movl   _n(%rip), %edx
  movl   _1(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_4        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_4(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_5        # e pula para o final do bloco

CMP_LBL_TEMP_4:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_4(%rip)

CMP_LBL_TEMP_5:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_4(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_3

## TAC_ENDFUN
  movl   _1(%rip), %eax #return
  popq   %rbp
  ret
```

```
# TAC_LABEL
   __TMP_LABEL_3: # AUTO



# TAC_MOVE //ASSIGN
   movl   _1(%rip), %eax
   movl   %eax, _result(%rip)
   movl   $0, %eax




# TAC_LABEL
   __TMP_LABEL_4: # AUTO



# TAC_GT
  movl   _n(%rip), %edx
  movl   _1(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_6        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_5(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_7        # e pula para o final do bloco

CMP_LBL_TEMP_6:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_5(%rip)

CMP_LBL_TEMP_7:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_5(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_5


# TAC_ADD
   movl   _a(%rip), %edx
   movl   _b(%rip), %eax
```

```
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_6(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_6(%rip), %eax
   movl   %eax, _result(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   _b(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   _result(%rip), %eax
   movl   %eax, _b(%rip)
   movl   $0, %eax




# TAC_SUB
   movl   _n(%rip), %edx
   movl   _1(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_7(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_7(%rip), %eax
```

```
   movl   %eax, _n(%rip)
   movl   $0, %eax




# TAC_JMP
  jmp    __TMP_LABEL_4



# TAC_LABEL
   __TMP_LABEL_5: # AUTO


## TAC_ENDFUN
  movl   _result(%rip), %eax #return
  popq   %rbp
  ret



# TAC_BEGINFUN
  .text
  .globl main
main:
  pushq  %rbp
  movq   %rsp, %rbp


# TAC_PRINT_STRING
   leaq   __TMP_VAR_8(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_9(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax
```

```
# TAC_PRINT_STRING
   leaq   __TMP_VAR_11(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_10(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
  leaq   __TMP_VAR_12(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
  call   __isoc99_scanf@PLT

# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_12(%rip), %eax
   movl   %eax, _input__dash__one(%rip)
   movl   $0, %eax




# TAC_PRINT_STRING
   leaq   __TMP_VAR_13(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
  leaq   __TMP_VAR_14(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
```

```
  call   __isoc99_scanf@PLT

# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_14(%rip), %eax
   movl   %eax, _input__dash__two(%rip)
   movl   $0, %eax




# TAC_TAC_FUN_CALL_ARGS

  movl   _input__dash__one(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   fatorial
  addq   $8, %rsp
  movl   %eax, __TMP_VAR_16(%rip) #move return to tempvar


# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_16(%rip), %eax
   movl   %eax, _result__dash__fac(%rip)
   movl   $0, %eax




# TAC_TAC_FUN_CALL_ARGS

  movl   _input__dash__two(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   fibonacci
  addq   $8, %rsp
  movl   %eax, __TMP_VAR_18(%rip) #move return to tempvar


# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_18(%rip), %eax
   movl   %eax, _result__dash__fib(%rip)
   movl   $0, %eax
```

```
# TAC_PRINT_STRING
   leaq   __TMP_VAR_20(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _input__dash__one(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_19(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _result__dash__fac(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_GT
  movl   _result__dash__fac(%rip), %edx
  movl   _result__dash__fib(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_8        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_21(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_9        # e pula para o final do bloco

CMP_LBL_TEMP_8:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_21(%rip)
```

```
CMP_LBL_TEMP_9:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_21(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_6


# TAC_PRINT_STRING
   leaq   __TMP_VAR_22(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_JMP
  jmp    __TMP_LABEL_7



# TAC_LABEL
   __TMP_LABEL_6: # AUTO



# TAC_PRINT_STRING
   leaq   __TMP_VAR_23(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_7: # AUTO


# TAC_PRINT_INT
```

```
   movl   _input__dash__two(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_25(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _result__dash__fib(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_24(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_SUB
   movl   _result__dash__fac(%rip), %edx
   movl   _result__dash__fib(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_26(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_26(%rip), %eax
   movl   %eax, _a(%rip)
```

```
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   _0(%rip), %eax
   movl   %eax, _b(%rip)
   movl   $0, %eax




# TAC_LABEL
   __TMP_LABEL_8: # AUTO



# TAC_GT
  movl   _a(%rip), %edx
  movl   _0(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_10       # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_27(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_11       # e pula para o final do bloco

CMP_LBL_TEMP_10:       #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_27(%rip)

CMP_LBL_TEMP_11:   #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_27(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_9


# TAC_ADD
   movl   _b(%rip), %edx
   movl   _1(%rip), %eax
```

```
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_28(%rip)
   movl   $0, %eax



# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_28(%rip), %eax
   movl   %eax, _b(%rip)
   movl   $0, %eax



# TAC_SUB
   movl   _a(%rip), %edx
   movl   _result__dash__fib(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_29(%rip)
   movl   $0, %eax



# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_29(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax



# TAC_JMP
  jmp    __TMP_LABEL_8



# TAC_LABEL
   __TMP_LABEL_9: # AUTO
```

```
# TAC_LT
  movl   _0(%rip), %edx
  movl   _b(%rip), %eax
  cmpl   %eax, %edx
  jl CMP_LBL_TEMP_12       # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_30(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_13       # e pula para o final do bloco

CMP_LBL_TEMP_12:       #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_30(%rip)

CMP_LBL_TEMP_13:   #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_30(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_10


# TAC_PRINT_STRING
   leaq   __TMP_VAR_34(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _input__dash__one(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_33(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax
```

```
# TAC_PRINT_INT
   movl   _b(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_32(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _input__dash__two(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_31(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_10:    # AUTO



## TAC_ENDFUN
  popq   %rbp
  ret
```

Programa de teste 3

```
   .data
__TMP_VAR_35: .long    0
__TMP_VAR_11: .long    0
__TMP_VAR_4: .long 0
__TMP_VAR_25: .string  "\t"
_0: .long  0
_1: .long  1
_6: .long  6
__TMP_VAR_15: .string  "\nOs "
_a: .long  0
_b: .long  0
__TMP_VAR_20: .long    0
_n: .long  0
__TMP_VAR_3: .long 0
__TMP_VAR_36: .string  "\n"
__TMP_VAR_19: .long    0
_count: .long  0
__TMP_VAR_2: .long 0
__TMP_VAR_26: .long    0
__TMP_VAR_1: .long 0
__TMP_VAR_21: .long    0
__TMP_VAR_12: .long    0
_total_n: .long    30
__TMP_VAR_0: .long 0
__TMP_VAR_16: .long    0
_columns: .long    6
__TMP_VAR_27: .long    0
__TMP_VAR_30: .long    0
_input: .long  0
_20: .long 20
__TMP_VAR_22: .string  "<-] "
_30: .long 30
__TMP_VAR_31: .string  "[*] "
__TMP_VAR_13: .long    0
__TMP_VAR_28: .long    0
__TMP_VAR_17: .long    0
__TMP_VAR_9: .long 0
__TMP_VAR_32: .long    0
__TMP_VAR_23: .string  "[->"
__TMP_VAR_8: .string   "\n"
_result: .long 0
__TMP_VAR_10: .long    0
__TMP_VAR_33: .long    0
```

```
_argsb: .long  0
__TMP_VAR_29: .long    0
__TMP_VAR_7: .string   "Enter a number: "
__TMP_VAR_14: .string  " primeiros numeros da sequencia de fibonacci são:
\n"
__TMP_VAR_24: .long    0
__TMP_VAR_6: .string   "Arrays \n"
__TMP_VAR_18: .long    0
__TMP_VAR_34: .string  "\n"
__TMP_VAR_5: .string   "Teste 3:"

# PRINT
print_string_int:
  .string    "%d"

print_string_float:
  .string    "%d/%d"

print_string_char:
  .string    "%c"
print_string:
  .string    "%s"


#READ
read:
  .string    "%d"



# TAC_ARRAY
_arr:
   .long 0   # arr[0]
   .long 0   # arr[1]
   .long 0   # arr[2]
   .long 0   # arr[3]
   .long 0   # arr[4]
   .long 0   # arr[5]
   .long 0   # arr[6]
   .long 0   # arr[7]
   .long 0   # arr[8]
   .long 0   # arr[9]
   .long 0   # arr[10]
   .long 0   # arr[11]
```

```
   .long 0   # arr[12]
   .long 0   # arr[13]
   .long 0   # arr[14]
   .long 0   # arr[15]
   .long 0   # arr[16]
   .long 0   # arr[17]
   .long 0   # arr[18]
   .long 0   # arr[19]

# TAC_BEGINFUN
  .text
  .globl fibonacci
fibonacci:
  pushq  %rbp
  movq   %rsp, %rbp

  # TAC_DEC_FUNC_ARGS
   movl   $0, %edx

  movl   16(%rbp), %eax
  movl   %eax, _argsb(%rip)

# TAC_MOVE //ASSIGN
   movl   _0(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   _1(%rip), %eax
   movl   %eax, _b(%rip)
   movl   $0, %eax




# TAC_EQ
  movl   _argsb(%rip), %edx
  movl   _0(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_0        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_0(%rip)     # Se falsa seta 0
```

```
  jmp    CMP_LBL_TEMP_1        # e pula para o final do bloco

CMP_LBL_TEMP_0:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_0(%rip)

CMP_LBL_TEMP_1:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_0(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_0

## TAC_ENDFUN
  movl   _0(%rip), %eax #return
  popq   %rbp
  ret



# TAC_LABEL
   __TMP_LABEL_0: # AUTO



# TAC_EQ
  movl   _argsb(%rip), %edx
  movl   _1(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_2        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_1(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_3        # e pula para o final do bloco

CMP_LBL_TEMP_2:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_1(%rip)

CMP_LBL_TEMP_3:    #final do bloco



# TAC_JMPZ
```

```
  movl   __TMP_VAR_1(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_1

## TAC_ENDFUN
  movl   _1(%rip), %eax #return
  popq   %rbp
  ret



# TAC_LABEL
   __TMP_LABEL_1: # AUTO



# TAC_MOVE //ASSIGN
   movl   _1(%rip), %eax
   movl   %eax, _result(%rip)
   movl   $0, %eax




# TAC_LABEL
   __TMP_LABEL_2: # AUTO



# TAC_GT
  movl   _argsb(%rip), %edx
  movl   _1(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_4        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_2(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_5        # e pula para o final do bloco

CMP_LBL_TEMP_4:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_2(%rip)

CMP_LBL_TEMP_5:    #final do bloco
```

```
# TAC_JMPZ
  movl   __TMP_VAR_2(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_3


# TAC_ADD
   movl   _a(%rip), %edx
   movl   _b(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_3(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_3(%rip), %eax
   movl   %eax, _result(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   _b(%rip), %eax
   movl   %eax, _a(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   _result(%rip), %eax
   movl   %eax, _b(%rip)
   movl   $0, %eax
```

```
# TAC_SUB
   movl   _argsb(%rip), %edx
   movl   _1(%rip), %eax
   subl   %eax, %edx
   movl   %edx, %eax
   movl   %eax, __TMP_VAR_4(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_4(%rip), %eax
   movl   %eax, _argsb(%rip)
   movl   $0, %eax




# TAC_JMP
  jmp    __TMP_LABEL_2



# TAC_LABEL
   __TMP_LABEL_3: # AUTO


## TAC_ENDFUN
  movl   _result(%rip), %eax #return
  popq   %rbp
  ret



# TAC_BEGINFUN
  .text
  .globl main
main:
  pushq  %rbp
  movq   %rsp, %rbp


# TAC_PRINT_STRING
```

```
   leaq   __TMP_VAR_5(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_6(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_8(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_7(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
  leaq   __TMP_VAR_9(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
  call   __isoc99_scanf@PLT

# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_9(%rip), %eax
   movl   %eax, _input(%rip)
   movl   $0, %eax
```

```
# TAC_MOVE //ASSIGN
   movl   _0(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_4: # AUTO



# TAC_LT
  movl   _n(%rip), %edx
  movl   _total_n(%rip), %eax
  cmpl   %eax, %edx
  jl CMP_LBL_TEMP_6        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_10(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_7        # e pula para o final do bloco

CMP_LBL_TEMP_6:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_10(%rip)

CMP_LBL_TEMP_7:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_10(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_5


# TAC_TAC_FUN_CALL_ARGS

  movl   _n(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   fibonacci
```

```
  addq   $8, %rsp
  movl   %eax, __TMP_VAR_12(%rip) #move return to tempvar


# TAC_ARRAY SET_ELEMENT EXP
  movl   _n(%rip), %edx   #endereço
  movl   __TMP_VAR_12(%rip), %eax   #valor
  movslq %edx, %rdx
  leaq   0(,%rdx,4), %rcx
  leaq   _arr(%rip), %rdx   #array
  movl   %eax, (%rcx,%rdx)

# TAC_ADD
   movl   _n(%rip), %edx
   movl   _1(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_13(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_13(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax




# TAC_JMP
  jmp    __TMP_LABEL_4



# TAC_LABEL
   __TMP_LABEL_5: # AUTO



# TAC_MOVE //ASSIGN
   movl   _0(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax
```

```
# TAC_PRINT_STRING
   leaq   __TMP_VAR_15(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _total_n(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_14(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_12:    # AUTO



# TAC_LT
  movl   _n(%rip), %edx
  movl   _total_n(%rip), %eax
  cmpl   %eax, %edx
  jl CMP_LBL_TEMP_8        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_16(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_9        # e pula para o final do bloco

CMP_LBL_TEMP_8:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_16(%rip)

CMP_LBL_TEMP_9:    #final do bloco
```

```
# TAC_JMPZ
  movl   __TMP_VAR_16(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_13


# TAC_TAC_FUN_CALL_ARGS

  movl   _n(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   fibonacci
  addq   $8, %rsp
  movl   %eax, __TMP_VAR_18(%rip) #move return to tempvar


# TAC_ARRAY SET_ELEMENT EXP
  movl   _n(%rip), %edx   #endereço
  movl   __TMP_VAR_18(%rip), %eax   #valor
  movslq %edx, %rdx
  leaq   0(,%rdx,4), %rcx
  leaq   _arr(%rip), %rdx   #array
  movl   %eax, (%rcx,%rdx)

# TAC_ARRAY GET_ELEMENT EXP
   movl   _n(%rip), %eax
   cltq
   leaq   0(,%rax,4), %rdx
   leaq   _arr(%rip), %rax
   movl   (%rdx,%rax), %eax
   movl   %eax, __TMP_VAR_19(%rip)

# TAC_EQ
  movl   _input(%rip), %edx
  movl   __TMP_VAR_19(%rip), %eax
  cmpl   %eax, %edx
  je CMP_LBL_TEMP_10       # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_20(%rip)        # Se falsa seta 0
```

```
  jmp    CMP_LBL_TEMP_11       # e pula para o final do bloco

CMP_LBL_TEMP_10:       #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_20(%rip)

CMP_LBL_TEMP_11:   #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_20(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_6


# TAC_PRINT_STRING
   leaq   __TMP_VAR_23(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_ARRAY GET_ELEMENT EXP
   movl   _n(%rip), %eax
   cltq
   leaq   0(,%rax,4), %rdx
   leaq   _arr(%rip), %rax
   movl   (%rdx,%rax), %eax
   movl   %eax, __TMP_VAR_21(%rip)
# TAC_PRINT_INT
   movl   __TMP_VAR_21(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_22(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
```

```
   movl   $0, %eax



# TAC_JMP
  jmp    __TMP_LABEL_7



# TAC_LABEL
   __TMP_LABEL_6: # AUTO



# TAC_ARRAY GET_ELEMENT EXP
   movl   _n(%rip), %eax
   cltq
   leaq   0(,%rax,4), %rdx
   leaq   _arr(%rip), %rax
   movl   (%rdx,%rax), %eax
   movl   %eax, __TMP_VAR_24(%rip)
# TAC_PRINT_INT
   movl   __TMP_VAR_24(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_25(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_7: # AUTO



# TAC_ARRAY GET_ELEMENT EXP
   movl   _n(%rip), %eax
```

```
   cltq
   leaq   0(,%rax,4), %rdx
   leaq   _arr(%rip), %rax
   movl   (%rdx,%rax), %eax
   movl   %eax, __TMP_VAR_26(%rip)

# TAC_GT
  movl   _input(%rip), %edx
  movl   __TMP_VAR_26(%rip), %eax
  cmpl   %eax, %edx
  jg CMP_LBL_TEMP_12       # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_27(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_13       # e pula para o final do bloco

CMP_LBL_TEMP_12:       #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_27(%rip)

CMP_LBL_TEMP_13:   #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_27(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_9


# TAC_ADD
   movl   _n(%rip), %edx
   movl   _1(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_28(%rip)
   movl   $0, %eax




# TAC_ARRAY GET_ELEMENT EXP
   movl   __TMP_VAR_28(%rip), %eax
   cltq
   leaq   0(,%rax,4), %rdx
   leaq   _arr(%rip), %rax
```

```
   movl   (%rdx,%rax), %eax
   movl   %eax, __TMP_VAR_29(%rip)

# TAC_LT
  movl   _input(%rip), %edx
  movl   __TMP_VAR_29(%rip), %eax
  cmpl   %eax, %edx
  jl CMP_LBL_TEMP_14       # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_30(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_15       # e pula para o final do bloco

CMP_LBL_TEMP_14:       #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_30(%rip)

CMP_LBL_TEMP_15:   #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_30(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_8


# TAC_PRINT_STRING
   leaq   __TMP_VAR_31(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_8: # AUTO



# TAC_LABEL
   __TMP_LABEL_9: # AUTO
```

```
# TAC_ADD
   movl   _n(%rip), %edx
   movl   _1(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_32(%rip)
   movl   $0, %eax




# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_32(%rip), %eax
   movl   %eax, _n(%rip)
   movl   $0, %eax




# TAC_GTE
  movl   _count(%rip), %edx
  movl   _columns(%rip), %eax
  cmpl   %eax, %edx
  jge    CMP_LBL_TEMP_16       # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_33(%rip)        # Se falsa seta 0
  jmp    CMP_LBL_TEMP_17       # e pula para o final do bloco

CMP_LBL_TEMP_16:       #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_33(%rip)

CMP_LBL_TEMP_17:   #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_33(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_10


# TAC_PRINT_STRING
   leaq   __TMP_VAR_34(%rip), %rdi
   movl   $0, %eax
```

```
    call    printf@PLT
    movl    $0, %eax



# TAC_MOVE //ASSIGN
    movl    _0(%rip), %eax
    movl    %eax, _count(%rip)
    movl    $0, %eax




# TAC_JMP
   jmp     __TMP_LABEL_11



# TAC_LABEL
    __TMP_LABEL_10:     # AUTO



# TAC_ADD
    movl    _count(%rip), %edx
    movl    _1(%rip), %eax
    addl    %edx, %eax
    movl    %eax, __TMP_VAR_35(%rip)
    movl    $0, %eax




# TAC_MOVE //ASSIGN
    movl    __TMP_VAR_35(%rip), %eax
    movl    %eax, _count(%rip)
    movl    $0, %eax



# TAC_LABEL
    __TMP_LABEL_11:     # AUTO
```

```
# TAC_JMP
  jmp    __TMP_LABEL_12



# TAC_LABEL
   __TMP_LABEL_13:    # AUTO



# TAC_PRINT_STRING
   leaq   __TMP_VAR_36(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



## TAC_ENDFUN
  popq   %rbp
  ret
```

Programa de teste 4

```
   .data
_first__dash__char: .byte  97
_fa:   .long  2
  .long  3
__TMP_VAR_11: .string  "tipos de dados e chamadas com multiplos argumentos
[int, char, float] \n"
__TMP_VAR_4: .string   ", "
_0: .long  0
_1: .long  1
_2: .long  2
_3: .long  3
_5: .long  5
_6: .long  6
__TMP_VAR_15: .long    0
_n: .long  0
__TMP_VAR_3: .long 0
_fb:   .long  5
  .long  6
__TMP_VAR_19: .long    0
_count: .long  0
__TMP_VAR_2: .string   "As proximas "
_repeat: .long 0
__TMP_VAR_1: .string   " letras do alfabeto depois de ["
__TMP_VAR_12: .string  "Digite o numero de letras: "
__TMP_VAR_0: .string   "] são: \n"
__TMP_VAR_16: .long    0
_in__dash__repeat: .long   0
__TMP_VAR_13: .string  "\n"
__TMP_VAR_17: .long    0
__TMP_VAR_9: .string   "\n\n O quarto argumento é um float com valor: "
__TMP_VAR_8: .string   "\n"
_result: .long 0
__TMP_VAR_10: .string  "Teste 4:"
__TMP_VAR_7: .string   "\n\n O terceiro argumento é um float com valor: "
__TMP_VAR_14: .long    0
_argsfb:   .long  0
  .long  0
_caracter: .byte   114
__TMP_VAR_6: .long 0
__TMP_VAR_18: .long    0
_argsfa:   .long  0
  .long  0
__TMP_VAR_5: .long 0
```

```
# PRINT
print_string_int:
  .string    "%d"

print_string_float:
  .string    "%d/%d"

print_string_char:
  .string    "%c"
print_string:
  .string    "%s"


#READ
read:
  .string    "%d"



# TAC_BEGINFUN
  .text
  .globl repeatchar
repeatchar:
  pushq  %rbp
  movq   %rsp, %rbp

  # TAC_DEC_FUNC_ARGS
   movl   $0, %edx

  movl   16(%rbp), %eax
  movl   %eax, _repeat(%rip)
  movl   24(%rbp), %eax
  movl   %eax, _caracter(%rip)
  movl   32(%rbp), %eax
  movl   40(%rbp), %edx
  movl   %eax, _argsfa(%rip)
  movl   %edx, 4+_argsfa(%rip)
  movl   48(%rbp), %eax
  movl   56(%rbp), %edx
  movl   %eax, _argsfb(%rip)
  movl   %edx, 4+_argsfb(%rip)

# TAC_PRINT_STRING
```

```
   leaq   __TMP_VAR_2(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_INT
   movl   _repeat(%rip), %esi   # mov a to reg
   #movl  %eax, %esi
   leaq   print_string_int(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_1(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_CHAR
   movzbl _caracter(%rip), %eax   # mov a to reg
   movsbl %al, %eax
   movl   %eax, %esi
   leaq   print_string_char(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_0(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_LABEL
   __TMP_LABEL_0: # AUTO



# TAC_LT
```

```
  movl   _count(%rip), %edx
  movl   _repeat(%rip), %eax
  cmpl   %eax, %edx
  jl CMP_LBL_TEMP_0        # se condição verdadeira pula para setar 1
  movl   $0, __TMP_VAR_3(%rip)     # Se falsa seta 0
  jmp    CMP_LBL_TEMP_1        # e pula para o final do bloco

CMP_LBL_TEMP_0:        #se condição verdadeira seta 1
  movl   $1, __TMP_VAR_3(%rip)

CMP_LBL_TEMP_1:    #final do bloco



# TAC_JMPZ
  movl   __TMP_VAR_3(%rip), %edx
  movl   $0, %eax
  cmpl   %eax, %edx     # Se condicional anterior for 0 pula o trecho a
baixo
   jz __TMP_LABEL_1

# TAC_PRINT_CHAR
   movzbl _caracter(%rip), %eax   # mov a to reg
   movsbl %al, %eax
   movl   %eax, %esi
   leaq   print_string_char(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_4(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_ADD
   movl   _count(%rip), %edx
   movl   _1(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_5(%rip)
   movl   $0, %eax
```

```
# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_5(%rip), %eax
   movl   %eax, _count(%rip)
   movl   $0, %eax



# TAC_ADD
   movl   _caracter(%rip), %edx
   movl   _1(%rip), %eax
   addl   %edx, %eax
   movl   %eax, __TMP_VAR_6(%rip)
   movl   $0, %eax



# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_6(%rip), %eax
   movl   %eax, _caracter(%rip)
   movl   $0, %eax



# TAC_JMP
  jmp    __TMP_LABEL_0


# TAC_LABEL
   __TMP_LABEL_1: # AUTO


# TAC_PRINT_STRING
   leaq   __TMP_VAR_7(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
```

```
   movl   $0, %eax


# TAC_PRINT_FLOAT
   movl   4+_argsfa(%rip), %edx   # mov a to reg
   movl   _argsfa(%rip), %eax   # mov a to reg
   movl   %eax, %esi
   leaq   print_string_float(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_9(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


# TAC_PRINT_FLOAT
   movl   4+_argsfb(%rip), %edx   # mov a to reg
   movl   _argsfb(%rip), %eax   # mov a to reg
   movl   %eax, %esi
   leaq   print_string_float(%rip), %rdi
  call   printf@PLT



# TAC_PRINT_STRING
   leaq   __TMP_VAR_8(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax


## TAC_ENDFUN
  movl   _result(%rip), %eax #return
  popq   %rbp
  ret



# TAC_BEGINFUN
  .text
```

```
  .globl main
main:
  pushq  %rbp
  movq   %rsp, %rbp


# TAC_PRINT_STRING
   leaq   __TMP_VAR_10(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_11(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_13(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_PRINT_STRING
   leaq   __TMP_VAR_12(%rip), %rdi
   movl   $0, %eax
   call   printf@PLT
   movl   $0, %eax



# TAC_READ
  leaq   __TMP_VAR_14(%rip), %rsi
  leaq   read(%rip), %rdi
  movl   $0, %eax
  call   __isoc99_scanf@PLT
```

```
# TAC_MOVE //ASSIGN
   movl   __TMP_VAR_14(%rip), %eax
   movl   %eax, _in__dash__repeat(%rip)
   movl   $0, %eax




# TAC_TAC_FUN_CALL_ARGS

  movl   4+_fb(%rip), %r10d
  pushq  %r10
  movl   _fb(%rip), %r10d
  pushq  %r10

  movl   4+_fa(%rip), %r10d
  pushq  %r10
  movl   _fa(%rip), %r10d
  pushq  %r10

  movl   _first__dash__char(%rip), %r10d
  pushq  %r10

  movl   _in__dash__repeat(%rip), %r10d
  pushq  %r10

  # TAC_TAC_FUN_CALL
  call   repeatchar
  addq   $48, %rsp
  movl   %eax, __TMP_VAR_19(%rip) #move return to tempvar

## TAC_ENDFUN
  movl   __TMP_VAR_19(%rip), %eax #return
  popq   %rbp
  ret
```

Logs de compilação

Programa de teste 0

```
patrick@DESKTOP-5MBUUP8:/mnt/d/UFRGS/2021-2/compiladores/Aulas/Trabalho/eta
pa6$ make clean && make && ./etapa6 test_programs/input.txt output.txt &&
gcc test_programs/input.s && ./a.out
rm lex.yy.c y.tab.c y.tab.h *.o etapa6
gcc -c -g main.c
yacc -d parser.y
parser.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
lex scanner.l
gcc -c -g  lex.yy.c
gcc -c -g  y.tab.c
gcc -c -g  hash.c
gcc -c -g  ast.c
gcc -c -g  semantic.c
gcc -c -g tacs.c
gcc -c -g  asm.c
gcc -c -g commons.c
gcc main.o lex.yy.o y.tab.o hash.o ast.o semantic.o tacs.o asm.o commons.o
-g -o etapa6


Semantics start

Undeclared semantic errors: 0

ast(AST_PROGRAM) {
 ast(AST_DECLARATION_LIST) {
   ast(AST_DECLARATION_GLOBAL_CHAR, c) {
     ast(AST_SYMBOL, 'x') {
   ast(AST_DECLARATION_LIST) {
     ast(AST_DECLARATION_GLOBAL_CHAR, d) {
       ast(AST_SYMBOL, 100) {
     ast(AST_DECLARATION_LIST) {
       ast(AST_DECLARATION_GLOBAL_INT, a) {
         ast(AST_SYMBOL, 'A') {
       ast(AST_DECLARATION_LIST) {
         ast(AST_DECLARATION_GLOBAL_INT, i) {
           ast(AST_SYMBOL, 1) {
         ast(AST_DECLARATION_LIST) {
           ast(AST_DECLARATION_GLOBAL_ARRAY_INT, v) {
             ast(AST_SYMBOL, 10) {
```

```
            ast(AST_ARRAY_VAL, 'a') {
              ast(AST_ARRAY_VAL, 0) {
                ast(AST_ARRAY_VAL, 0) {
                  ast(AST_ARRAY_VAL, 0) {
                    ast(AST_ARRAY_VAL, 0) {
                      ast(AST_ARRAY_VAL, 0) {
                        ast(AST_ARRAY_VAL, 0) {
                          ast(AST_ARRAY_VAL, 0) {
                            ast(AST_ARRAY_VAL, 0) {
                              ast(AST_ARRAY_VAL, 0) {
                                ast(AST_ARRAY_VAL, 0) {
                        ast(AST_DECLARATION_LIST) {
                          ast(AST_DECLARATION_GLOBAL_ARRAY_INT, matrix) {
                            ast(AST_SYMBOL, 100) {
                          ast(AST_DECLARATION_LIST) {
                            ast(AST_DECLARATION_GLOBAL_FLOAT, f) {
                              ast(AST_SYMBOL, 2) {
                              ast(AST_SYMBOL, 3) {
                            ast(AST_DECLARATION_LIST) {
                              ast(AST_DECLARATION_FUNCTION_INT, main) {
                                ast(AST_DECLARATION_FUNCTION_BODY) {
                                  ast(AST_COMMAND_BLOCK) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_ATTRIBUITION, a) {
                                        ast(AST_SYMBOL, 0) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_ATTRIBUITION, a) {
                                          ast(AST_SUB, __dash__) {
                                            ast(AST_SYMBOL, a) {
                                            ast(AST_SYMBOL, i) {
                                        ast(AST_COMMAND_LIST) {
                                          ast(AST_ATTRIBUITION, a) {
                                            ast(AST_SYMBOL, 5) {
                                          ast(AST_COMMAND_LIST) {
                                            ast(AST_ARRAY_ATTRIBUITION, v) {
                                              ast(AST_SYMBOL, a) {
                                              ast(AST_SYMBOL, 55) {
                                            ast(AST_COMMAND_LIST) {
                                              ast(AST_PRINT) {
                                                ast(AST_PRINT_EXP) {
                                                  ast(AST_ARRAY_ELEMENT, v) {
                                                    ast(AST_SYMBOL, 5) {
                                              ast(AST_COMMAND_LIST) {
                                                ast(AST_PRINT) {
                                                  ast(AST_PRINT_EXP) {
```

```
                                        ast(AST_SYMBOL, a) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_ATTRIBUITION, i) {
                                        ast(AST_SYMBOL, 2) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_PRINT) {
                                          ast(AST_PRINT_STRING) {
                                            ast(AST_SYMBOL, Digite um
numero: \n) {
                                        ast(AST_COMMAND_LIST) {
                                          ast(AST_ATTRIBUITION, a) {
                                            ast(AST_READ) {
                                          ast(AST_COMMAND_LIST) {
                                            ast(AST_FLUX_CONTROLL_WHILE) {
                                              ast(AST_LT, <) {
                                                ast(AST_SYMBOL, i) {
                                                ast(AST_SYMBOL, 10) {
                                              ast(AST_COMMAND_BLOCK) {
                                                ast(AST_COMMAND_LIST) {
                                                  ast(AST_ATTRIBUITION, i) {
                                                    ast(AST_FUNCTION_CALL,
incn) {

ast(AST_FUNCTION_CALL_ARGS) {
                                                              ast(AST_SYMBOL, i) {

ast(AST_FUNCTION_CALL_ARGS) {
                                                                ast(AST_SYMBOL, 1)
{
                                                            ast(AST_COMMAND_LIST) {
                                                              ast(AST_ATTRIBUITION, a)
{
                                                                ast(AST_FUNCTION_CALL,
incn) {

ast(AST_FUNCTION_CALL_ARGS) {
                                                                  ast(AST_SYMBOL, a)
{

ast(AST_FUNCTION_CALL_ARGS) {
                                                                    ast(AST_SYMBOL,
1) {
                                                          ast(AST_COMMAND_LIST) {
                                                            ast(AST_PRINT) {
```

```
                                                  ast(AST_PRINT_STRING) {
                                                    ast(AST_SYMBOL,
Incrementado algumas vezes a fica ) {
                                                    ast(AST_PRINT_EXP) {
                                                      ast(AST_SYMBOL, a) {
                                                      ast(AST_PRINT_STRING) {
                                                        ast(AST_SYMBOL, \n) {
                                                 ast(AST_COMMAND_LIST) {

ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                                    ast(AST_EQ, ==) {
                                                      ast(AST_SYMBOL, a) {
                                                      ast(AST_SYMBOL, 15) {
                                                    ast(AST_COMMAND_BLOCK) {
                                                      ast(AST_COMMAND_LIST) {
                                                        ast(AST_LABEL,
label__dash__x) {
                                                        ast(AST_COMMAND_LIST)
{

ast(AST_ATTRIBUITION, a) {
                                                             ast(AST_SUB,
__dash__) {
                                                               ast(AST_SYMBOL,
a) {
                                                               ast(AST_SYMBOL,
1) {

ast(AST_COMMAND_LIST) {
                                                             ast(AST_PRINT) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, A era=15\n) {
                                                 ast(AST_COMMAND_LIST) {

ast(AST_FLUX_CONTROLL_IF|ELSE) {

ast(AST_EXPRESSION_BLOCK) {
                                                        ast(AST_EQ, ==) {
                                                          ast(AST_SYMBOL, i) {
                                                          ast(AST_SYMBOL, 100)
{
                                                      ast(AST_COMMAND_BLOCK) {
```

```
                                                    ast(AST_COMMAND_LIST)
{
                                                      ast(AST_PRINT) {

ast(AST_PRINT_STRING) {
                                                          ast(AST_SYMBOL,
Nao tem como isso....\n) {
                                                  ast(AST_PRINT) {
                                                    ast(AST_PRINT_STRING)
{
                                                      ast(AST_SYMBOL,
OK!\n) {
                                                ast(AST_COMMAND_LIST) {

ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                                    ast(AST_GT, >) {
                                                      ast(AST_SYMBOL, a) {
                                                      ast(AST_SYMBOL, 0) {

ast(AST_FLUX_CONTROLL_GOTO, label__dash__x) {
                 ast(AST_DECLARATION_LIST) {
                   ast(AST_DECLARATION_FUNCTION_INT, incn) {
                     ast(AST_DECLARATION_FUNCTION_ARGS_INT, x) {
                       ast(AST_DECLARATION_FUNCTION_ARGS_INT, n) {
                     ast(AST_DECLARATION_FUNCTION_BODY) {
                       ast(AST_COMMAND_BLOCK) {
                         ast(AST_COMMAND_LIST) {
                           ast(AST_RETURN) {
                             ast(AST_ADD, +) {
                               ast(AST_SYMBOL, x) {
                               ast(AST_SYMBOL, n) {

hash after tacGenerateCode

_TMP_LABEL_0Table[23]    has main        type 259        datatype 1
datavalue 0     datastring (null)
Table[24]        has _TMP_VAR_11         type 1607       datatype 5
datavalue 0     datastring \n
Table[36]        has _TMP_VAR_4          type 1607       datatype 0
datavalue 0     datastring (null)
Table[43]        has +           type 269        datatype 0      datavalue
0     datastring (null)
Table[47]        has /           type 272        datatype 0      datavalue
0     datastring (null)
```

```
Table[48]        has 0           type 262        datatype 1      datavalue
0     datastring (null)
Table[49]        has 1           type 262        datatype 1      datavalue
1     datastring (null)
Table[50]        has _TMP_LABEL_1        type 265        datatype 0
datavalue 0     datastring (null)
Table[50]        has 2           type 262        datatype 1      datavalue
2     datastring (null)
Table[51]        has 3           type 262        datatype 1      datavalue
3     datastring (null)
Table[53]        has 5           type 262        datatype 1      datavalue
5     datastring (null)
Table[60]        has <           type 274        datatype 0      datavalue
0     datastring (null)
Table[62]        has >           type 276        datatype 0      datavalue
0     datastring (null)
Table[87]        has _TMP_VAR_15         type 1607       datatype 5
datavalue 0     datastring A era=15\n
Table[97]        has a           type 258        datatype 1      datavalue
0     datastring (null)
Table[99]        has _TMP_VAR_20         type 1607       datatype 0
datavalue 0     datastring (null)
Table[99]        has c           type 258        datatype 2      datavalue
120   datastring 'x'
Table[100]       has d           type 258        datatype 2      datavalue
48    datastring 100
Table[102]       has f           type 258        datatype 3      datavalue
2     datastring 3
Table[105]       has i           type 258        datatype 1      datavalue
1     datastring (null)
Table[110]       has n           type 260        datatype 1      datavalue
0     datastring (null)
Table[112]       has _TMP_VAR_3          type 1607       datatype 0
datavalue 0     datastring (null)
Table[118]       has v           type 261        datatype 10     datavalue
10    datastring (null)
Table[120]       has x           type 260        datatype 1      datavalue
0     datastring (null)
Table[150]       has _TMP_VAR_19         type 1607       datatype 0
datavalue 0     datastring (null)
Table[171]       has matrix      type 261        datatype 10     datavalue
100   datastring (null)
Table[188]       has _TMP_VAR_2          type 1607       datatype 5
datavalue 0     datastring Digite um numero: \n
```

```
Table[260]       has \n          type 264        datatype 5      datavalue
0     datastring (null)
Table[264]       has _TMP_VAR_1          type 1607       datatype 0
datavalue 0     datastring (null)
Table[289]       has _TMP_VAR_12         type 1607       datatype 5
datavalue 0     datastring Incrementado algumas vezes a fica
Table[340]       has _TMP_VAR_0          type 1607       datatype 0
datavalue 0     datastring (null)
Table[352]       has _TMP_VAR_16         type 1607       datatype 0
datavalue 0     datastring (null)
Table[364]       has label__dash__x      type 265        datatype 0
datavalue 0     datastring (null)
Table[401]       has _TMP_LABEL_6        type 265        datatype 0
datavalue 0     datastring (null)
Table[406]       has 10          type 262        datatype 1      datavalue
10    datastring (null)
Table[460]       has _TMP_LABEL_4        type 265        datatype 0
datavalue 0     datastring (null)
Table[510]       has incn        type 259        datatype 1      datavalue
0     datastring (null)
Table[519]       has _TMP_LABEL_2        type 265        datatype 265
datavalue 0     datastring label__dash__x
Table[554]       has _TMP_VAR_13         type 1607       datatype 0
datavalue 0     datastring (null)
Table[578]       has _TMP_LABEL_0        type 265        datatype 0
datavalue 0     datastring (null)
Table[592]       has OK!\n       type 264        datatype 5      datavalue
0     datastring (null)
Table[593]       has 100         type 262        datatype 1      datavalue
100   datastring (null)
Table[617]       has _TMP_VAR_17         type 1607       datatype 5
datavalue 0     datastring Nao tem como isso....\n
Table[653]       has _TMP_VAR_9          type 1607       datatype 0
datavalue 0     datastring (null)
Table[656]       has 15          type 262        datatype 1      datavalue
15    datastring (null)
Table[705]       has Digite um numero: \n        type 264        datatype 5
datavalue 0     datastring (null)
Table[729]       has _TMP_VAR_8          type 1607       datatype 0
datavalue 0     datastring (null)
Table[742]       has 'A'         type 263        datatype 2      datavalue
0     datastring (null)
Table[756]       has _TMP_VAR_10         type 1607       datatype 0
datavalue 0     datastring (null)
```

```
Table[779]       has Nao tem como isso....\n            type 264
datatype 5      datavalue 0     datastring (null)
Table[781]       has Incrementado algumas vezes a fica          type 264
datatype 5      datavalue 0     datastring (null)
Table[791]       has ==          type 277        datatype 0      datavalue
0     datastring (null)
Table[800]       has 'x'         type 263        datatype 2      datavalue
0     datastring (null)
Table[805]       has _TMP_VAR_7          type 1607       datatype 0
datavalue 0     datastring (null)
Table[812]       has 'a'         type 281        datatype 0      datavalue
0     datastring (null)
Table[814]       has __dash__            type 270        datatype 0
datavalue 0     datastring (null)
Table[819]       has _TMP_VAR_14         type 1607       datatype 0
datavalue 0     datastring (null)
Table[855]       has A era=15\n          type 264        datatype 5
datavalue 0     datastring (null)
Table[868]       has 55          type 262        datatype 1      datavalue
55    datastring (null)
Table[881]       has _TMP_VAR_6          type 1607       datatype 0
datavalue 0     datastring (null)
Table[882]       has _TMP_VAR_18         type 1607       datatype 5
datavalue 0     datastring OK!\n
Table[929]       has _TMP_LABEL_5        type 265        datatype 0
datavalue 0     datastring (null)
Table[957]       has _TMP_VAR_5          type 1607       datatype 0
datavalue 0     datastring (null)
Table[988]       has _TMP_LABEL_3        type 265        datatype 0
datavalue 0     datastring (null)


TAC(TAC_DEC_GLOBAL_CHAR, c, 'x', 0);
TAC(TAC_DEC_GLOBAL_CHAR, d, 100, 0);
TAC(TAC_DEC_GLOBAL_INT, a, 'A', 0);
TAC(TAC_DEC_GLOBAL_INT, i, 1, 0);
TAC(TAC_DEC_GLOBAL_ARR, v, 'a', 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
```

```
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, v, 0, 10);
TAC(TAC_DEC_GLOBAL_ARR, matrix, 0, 100);
TAC(TAC_DEC_GLOBAL_FLOAT, f, 2, 3);
TAC(TAC_BEGINFUN, main, _TMP_LABEL_6, 0);
TAC(TAC_MOVE, a, 0, 0);
TAC(TAC_SUB, _TMP_VAR_0, a, i);
TAC(TAC_MOVE, a, _TMP_VAR_0, 0);
TAC(TAC_MOVE, a, 5, 0);
TAC(TAC_ARR_SET_ELEMENT, v, a, 55);
TAC(TAC_ARR_GET_ELEMENT, _TMP_VAR_1, v, 5);
TAC(TAC_PRINT_INT, _TMP_VAR_1, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_1, 0, 0);
TAC(TAC_PRINT_INT, a, 0, 0);
TAC(TAC_PRINT, a, 0, 0);
TAC(TAC_MOVE, i, 2, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_2, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_2, 0, 0);
TAC(TAC_READ, _TMP_VAR_3, 0, 0);
TAC(TAC_MOVE, a, _TMP_VAR_3, 0);
TAC(TAC_LABEL, _TMP_LABEL_0, 0, 0);
TAC(TAC_LT, _TMP_VAR_4, i, 10);
TAC(TAC_JMPZ, _TMP_LABEL_1, _TMP_VAR_4, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_5, 1, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_6, i, _TMP_VAR_5);
TAC(TAC_FUN_CALL, _TMP_VAR_7, incn, 0);
TAC(TAC_MOVE, i, _TMP_VAR_7, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_8, 1, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_9, a, _TMP_VAR_8);
TAC(TAC_FUN_CALL, _TMP_VAR_10, incn, 0);
TAC(TAC_MOVE, a, _TMP_VAR_10, 0);
TAC(TAC_JMP, _TMP_LABEL_0, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_1, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_12, 0, 0);
TAC(TAC_PRINT_INT, a, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_11, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_11, 0, 0);
TAC(TAC_EQ, _TMP_VAR_13, a, 15);
TAC(TAC_JMPZ, _TMP_LABEL_3, _TMP_VAR_13, 0);
TAC(TAC_LABEL, _TMP_LABEL_2, label__dash__x, 0);
TAC(TAC_SUB, _TMP_VAR_14, a, 1);
TAC(TAC_MOVE, a, _TMP_VAR_14, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_15, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_15, 0, 0);
```

```
TAC(TAC_LABEL, _TMP_LABEL_3, 0, 0);
TAC(TAC_EQ, _TMP_VAR_16, i, 100);
TAC(TAC_JMPZ, _TMP_LABEL_4, _TMP_VAR_16, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_17, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_17, 0, 0);
TAC(TAC_JMP, _TMP_LABEL_5, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_4, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_18, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_18, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_5, 0, 0);
TAC(TAC_GT, _TMP_VAR_19, a, 0);
TAC(TAC_JMPZ, _TMP_LABEL_6, _TMP_VAR_19, 0);
TAC(TAC_LABEL, _TMP_LABEL_6, 0, 0);
TAC(TAC_ENDFUN, main, 0, 0);
TAC(TAC_DEC_FUNC_ARGS, n, 0, 0);
TAC(TAC_DEC_FUNC_ARGS, x, n, 0);
TAC(TAC_BEGINFUN, incn, _TMP_VAR_20, 0);
TAC(TAC_ADD, _TMP_VAR_20, x, n);
TAC(TAC_RETURN, _TMP_VAR_20, 0, 0);
TAC(TAC_ENDFUN, incn, x, 0);
values: 'a' 0 0 0 0 0 0 0 0 0
values:
condition: a==15 body_if: {
label__dash__x:
a=a - 1;
print "A era=15\n";

} body_else:

.condition: (i==100) body_if: {
print "Nao tem como isso....\n";

} body_else:print "OK!\n"

.condition: a>0 body_if: goto label__dash__x body_else:

.Numero de linhas: 59.
Compilation Success.
555Digite um numero:
15
Incrementado algumas vezes a fica 23
OK!
```

Programa de teste 1

```
patrick@DESKTOP-5MBUUP8:/mnt/d/UFRGS/2021-2/compiladores/Aulas/Trabalho/eta
pa6$ make clean && make && ./etapa6 test_programs/test_1.txt output.txt &&
gcc test_programs/test_1.s && ./a.out
rm lex.yy.c y.tab.c y.tab.h *.o etapa6
gcc -c -g main.c
yacc -d parser.y
parser.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
lex scanner.l
gcc -c -g  lex.yy.c
gcc -c -g  y.tab.c
gcc -c -g  hash.c
gcc -c -g  ast.c
gcc -c -g  semantic.c
gcc -c -g tacs.c
gcc -c -g  asm.c
gcc -c -g commons.c
gcc main.o lex.yy.o y.tab.o hash.o ast.o semantic.o tacs.o asm.o commons.o
-g -o etapa6


Semantics start

Undeclared semantic errors: 0

ast(AST_PROGRAM) {
 ast(AST_DECLARATION_LIST) {
   ast(AST_DECLARATION_GLOBAL_INT, input__dash__one) {
     ast(AST_SYMBOL, 0) {
   ast(AST_DECLARATION_LIST) {
     ast(AST_DECLARATION_GLOBAL_INT, input__dash__two) {
       ast(AST_SYMBOL, 0) {
     ast(AST_DECLARATION_LIST) {
       ast(AST_DECLARATION_GLOBAL_INT, result) {
         ast(AST_SYMBOL, 0) {
       ast(AST_DECLARATION_LIST) {
         ast(AST_DECLARATION_FUNCTION_INT, main) {
           ast(AST_DECLARATION_FUNCTION_BODY) {
             ast(AST_COMMAND_BLOCK) {
               ast(AST_COMMAND_LIST) {
                 ast(AST_PRINT) {
                   ast(AST_PRINT_STRING) {
                     ast(AST_SYMBOL, Teste 1:) {
                 ast(AST_COMMAND_LIST) {
```

```
                  ast(AST_PRINT) {
                    ast(AST_PRINT_STRING) {
                      ast(AST_SYMBOL, Atribuições e operações aritmeticas
com impressão:  [read, +, __dash__, *, /] \n) {
                  ast(AST_COMMAND_LIST) {
                    ast(AST_PRINT) {
                      ast(AST_PRINT_STRING) {
                        ast(AST_SYMBOL, \n) {
                        ast(AST_PRINT_STRING) {
                          ast(AST_SYMBOL, Enter the first value: ) {
                    ast(AST_COMMAND_LIST) {
                      ast(AST_ATTRIBUITION, input__dash__one) {
                        ast(AST_READ) {
                      ast(AST_COMMAND_LIST) {
                        ast(AST_PRINT) {
                          ast(AST_PRINT_STRING) {
                            ast(AST_SYMBOL, Enter the second value: ) {
                        ast(AST_COMMAND_LIST) {
                          ast(AST_ATTRIBUITION, input__dash__two) {
                            ast(AST_READ) {
                          ast(AST_COMMAND_LIST) {
                            ast(AST_PRINT) {
                              ast(AST_PRINT_STRING) {
                                ast(AST_SYMBOL, \n) {
                                ast(AST_PRINT_STRING) {
                                  ast(AST_SYMBOL, input1 + input2 = ) {
                                  ast(AST_PRINT_EXP) {
                                    ast(AST_ADD, +) {
                                      ast(AST_SYMBOL, input__dash__one) {
                                      ast(AST_SYMBOL, input__dash__two) {
                            ast(AST_COMMAND_LIST) {
                              ast(AST_PRINT) {
                                ast(AST_PRINT_STRING) {
                                  ast(AST_SYMBOL, \n) {
                                  ast(AST_PRINT_STRING) {
                                    ast(AST_SYMBOL, input1 __dash__ input2
= ) {

                                    ast(AST_PRINT_EXP) {
                                      ast(AST_SUB, __dash__) {
                                        ast(AST_SYMBOL, input__dash__one)
{

                                        ast(AST_SYMBOL, input__dash__two)
{
                                ast(AST_COMMAND_LIST) {
```

```
                                ast(AST_PRINT) {
                                  ast(AST_PRINT_STRING) {
                                    ast(AST_SYMBOL, \n) {
                                    ast(AST_PRINT_STRING) {
                                      ast(AST_SYMBOL, input1 * input2 = )
{
                                      ast(AST_PRINT_EXP) {
                                        ast(AST_MULT, *) {
                                          ast(AST_SYMBOL,
input__dash__one) {
                                          ast(AST_SYMBOL,
input__dash__two) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_PRINT) {
                                    ast(AST_PRINT_STRING) {
                                      ast(AST_SYMBOL, \n) {
                                      ast(AST_PRINT_STRING) {
                                        ast(AST_SYMBOL, input1 / input2 =
) {
                                        ast(AST_PRINT_EXP) {
                                          ast(AST_DIV, /) {
                                            ast(AST_SYMBOL,
input__dash__one) {
                                            ast(AST_SYMBOL,
input__dash__two) {
                                          ast(AST_PRINT_STRING) {
                                            ast(AST_SYMBOL, \n) {

hash after tacGenerateCode

Table[0]         has input1 * input2 =           type 264        datatype 5
datavalue 0     datastring (null)
Table[23]        has main        type 259        datatype 1      datavalue
0     datastring (null)
Table[24]        has _TMP_VAR_11         type 1607       datatype 5
datavalue 0     datastring input1 __dash__ input2 =
Table[36]        has _TMP_VAR_4          type 1607       datatype 0
datavalue 0     datastring (null)
Table[37]        has Enter the first value:              type 264
datatype 5      datavalue 0     datastring (null)
Table[42]        has *           type 271        datatype 0      datavalue
0     datastring (null)
Table[43]        has +           type 269        datatype 0      datavalue
0     datastring (null)
```

```
Table[47]        has /           type 272        datatype 0      datavalue
0     datastring (null)
Table[48]        has 0           type 262        datatype 1      datavalue
0     datastring (null)
Table[75]        has Teste 1:            type 264        datatype 5
datavalue 0     datastring (null)
Table[87]        has _TMP_VAR_15         type 1607       datatype 5
datavalue 0     datastring \n
Table[112]       has _TMP_VAR_3          type 1607       datatype 5
datavalue 0     datastring \n
Table[150]       has _TMP_VAR_19         type 1607       datatype 5
datavalue 0     datastring \n
Table[154]       has input__dash__one            type 258        datatype 1
datavalue 0     datastring (null)
Table[188]       has _TMP_VAR_2          type 1607       datatype 5
datavalue 0     datastring Enter the first value:
Table[208]       has input1 / input2 =           type 264        datatype 5
datavalue 0     datastring (null)
Table[241]       has input1 + input2 =           type 264        datatype 5
datavalue 0     datastring (null)
Table[260]       has \n          type 264        datatype 5      datavalue
0     datastring (null)
Table[264]       has _TMP_VAR_1          type 1607       datatype 5
datavalue 0     datastring Atribuições e operações aritmeticas com
impressão:  [read, +, __dash__, *, /] \n
Table[289]       has _TMP_VAR_12         type 1607       datatype 5
datavalue 0     datastring \n
Table[340]       has _TMP_VAR_0          type 1607       datatype 5
datavalue 0     datastring Teste 1:
Table[352]       has _TMP_VAR_16         type 1607       datatype 0
datavalue 0     datastring (null)
Table[554]       has _TMP_VAR_13         type 1607       datatype 0
datavalue 0     datastring (null)
Table[617]       has _TMP_VAR_17         type 1607       datatype 5
datavalue 0     datastring \n
Table[653]       has _TMP_VAR_9          type 1607       datatype 5
datavalue 0     datastring \n
Table[686]       has input__dash__two            type 258        datatype 1
datavalue 0     datastring (null)
Table[717]       has Enter the second value:             type 264
datatype 5      datavalue 0     datastring (null)
Table[729]       has _TMP_VAR_8          type 1607       datatype 5
datavalue 0     datastring input1 + input2 =
Table[748]       has result      type 258        datatype 1      datavalue
```

```
0     datastring (null)
Table[756]       has _TMP_VAR_10         type 1607       datatype 0
datavalue 0     datastring (null)
Table[805]       has _TMP_VAR_7          type 1607       datatype 0
datavalue 0     datastring (null)
Table[814]       has __dash__            type 270        datatype 0
datavalue 0     datastring (null)
Table[819]       has _TMP_VAR_14         type 1607       datatype 5
datavalue 0     datastring input1 * input2 =
Table[876]       has input1 __dash__ input2 =            type 264
datatype 5      datavalue 0     datastring (null)
Table[881]       has _TMP_VAR_6          type 1607       datatype 0
datavalue 0     datastring (null)
Table[882]       has _TMP_VAR_18         type 1607       datatype 5
datavalue 0     datastring input1 / input2 =
Table[943]       has Atribuições e operações aritmeticas com impressão:
[read, +, __dash__, *, /] \n            type 264        datatype 5
datavalue 0     datastring (null)
Table[957]       has _TMP_VAR_5          type 1607       datatype 5
datavalue 0     datastring Enter the second value:


TAC(TAC_DEC_GLOBAL_INT, input__dash__one, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, input__dash__two, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, result, 0, 0);
TAC(TAC_BEGINFUN, main, _TMP_VAR_17, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_0, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_0, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_1, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_1, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_3, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_2, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_2, 0, 0);
TAC(TAC_READ, _TMP_VAR_4, 0, 0);
TAC(TAC_MOVE, input__dash__one, _TMP_VAR_4, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_5, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_5, 0, 0);
TAC(TAC_READ, _TMP_VAR_6, 0, 0);
TAC(TAC_MOVE, input__dash__two, _TMP_VAR_6, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_9, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_8, 0, 0);
TAC(TAC_ADD, _TMP_VAR_7, input__dash__one, input__dash__two);
TAC(TAC_PRINT_INT, _TMP_VAR_7, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_7, 0, 0);
```

```
TAC(TAC_PRINT_STRING, _TMP_VAR_12, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_11, 0, 0);
TAC(TAC_SUB, _TMP_VAR_10, input__dash__one, input__dash__two);
TAC(TAC_PRINT_INT, _TMP_VAR_10, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_10, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_15, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_14, 0, 0);
TAC(TAC_MULT, _TMP_VAR_13, input__dash__one, input__dash__two);
TAC(TAC_PRINT_INT, _TMP_VAR_13, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_13, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_19, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_18, 0, 0);
TAC(TAC_DIV, _TMP_VAR_16, input__dash__one, input__dash__two);
TAC(TAC_PRINT_INT, _TMP_VAR_16, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_17, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_17, 0, 0);
TAC(TAC_ENDFUN, main, 0, 0);
Numero de linhas: 19.
Compilation Success.

Teste 1:Atribuições e operações aritmeticas com impressão:  [read, +, -, *,
/]

Enter the first value: 36
Enter the second value: 4

input1 + input2 = 40
input1 - input2 = 32
input1 * input2 = 144
input1 / input2 = 9
```

Programa de teste 2

```
patrick@DESKTOP-5MBUUP8:/mnt/d/UFRGS/2021-2/compiladores/Aulas/Trabalho/eta
pa6$ make clean && make && ./etapa6 test_programs/test_2.txt output.txt &&
gcc test_programs/test_2.s && ./a.out
rm lex.yy.c y.tab.c y.tab.h *.o etapa6
gcc -c -g main.c
yacc -d parser.y
parser.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
lex scanner.l
gcc -c -g  lex.yy.c
gcc -c -g  y.tab.c
gcc -c -g  hash.c
gcc -c -g  ast.c
gcc -c -g  semantic.c
gcc -c -g tacs.c
gcc -c -g  asm.c
gcc -c -g commons.c
gcc main.o lex.yy.o y.tab.o hash.o ast.o semantic.o tacs.o asm.o commons.o
-g -o etapa6


Semantics start

Undeclared semantic errors: 0

ast(AST_PROGRAM) {
 ast(AST_DECLARATION_LIST) {
   ast(AST_DECLARATION_GLOBAL_INT, input__dash__one) {
     ast(AST_SYMBOL, 0) {
   ast(AST_DECLARATION_LIST) {
     ast(AST_DECLARATION_GLOBAL_INT, input__dash__two) {
       ast(AST_SYMBOL, 0) {
     ast(AST_DECLARATION_LIST) {
       ast(AST_DECLARATION_GLOBAL_INT, result__dash__fac) {
         ast(AST_SYMBOL, 0) {
       ast(AST_DECLARATION_LIST) {
         ast(AST_DECLARATION_GLOBAL_INT, a) {
           ast(AST_SYMBOL, 0) {
         ast(AST_DECLARATION_LIST) {
           ast(AST_DECLARATION_GLOBAL_INT, b) {
             ast(AST_SYMBOL, 1) {
           ast(AST_DECLARATION_LIST) {
             ast(AST_DECLARATION_GLOBAL_INT, result__dash__fib) {
               ast(AST_SYMBOL, 0) {
```

```
            ast(AST_DECLARATION_LIST) {
              ast(AST_DECLARATION_GLOBAL_INT, result) {
                ast(AST_SYMBOL, 0) {
              ast(AST_DECLARATION_LIST) {
                ast(AST_DECLARATION_GLOBAL_INT, n) {
                  ast(AST_SYMBOL, 0) {
                ast(AST_DECLARATION_LIST) {
                  ast(AST_DECLARATION_FUNCTION_INT, fatorial) {
                    ast(AST_DECLARATION_FUNCTION_ARGS_INT, argsa) {
                    ast(AST_DECLARATION_FUNCTION_BODY) {
                      ast(AST_COMMAND_BLOCK) {
                        ast(AST_COMMAND_LIST) {
                          ast(AST_ATTRIBUITION, n) {
                            ast(AST_SYMBOL, argsa) {
                          ast(AST_COMMAND_LIST) {
                            ast(AST_ATTRIBUITION, result) {
                              ast(AST_SYMBOL, 1) {
                            ast(AST_COMMAND_LIST) {
                              ast(AST_FLUX_CONTROLL_WHILE) {
                                ast(AST_EXPRESSION_BLOCK) {
                                  ast(AST_GT, >) {
                                    ast(AST_SYMBOL, n) {
                                    ast(AST_SYMBOL, 1) {
                                ast(AST_COMMAND_BLOCK) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_ATTRIBUITION, result) {
                                      ast(AST_MULT, *) {
                                        ast(AST_SYMBOL, result) {
                                        ast(AST_SYMBOL, n) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_ATTRIBUITION, n) {
                                        ast(AST_SUB, __dash__) {
                                          ast(AST_SYMBOL, n) {
                                          ast(AST_SYMBOL, 1) {
                            ast(AST_COMMAND_LIST) {
                              ast(AST_RETURN) {
                                ast(AST_SYMBOL, result) {
                    ast(AST_DECLARATION_LIST) {
                      ast(AST_DECLARATION_FUNCTION_INT, fibonacci) {
                        ast(AST_DECLARATION_FUNCTION_ARGS_INT, argsb) {
                        ast(AST_DECLARATION_FUNCTION_BODY) {
                          ast(AST_COMMAND_BLOCK) {
                            ast(AST_COMMAND_LIST) {
                              ast(AST_ATTRIBUITION, n) {
```

```
                                ast(AST_SYMBOL, argsb) {
                              ast(AST_COMMAND_LIST) {
                                ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                  ast(AST_EXPRESSION_BLOCK) {
                                    ast(AST_EQ, ==) {
                                      ast(AST_SYMBOL, n) {
                                      ast(AST_SYMBOL, 0) {
                                  ast(AST_COMMAND_BLOCK) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_RETURN) {
                                        ast(AST_SYMBOL, 0) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                    ast(AST_EXPRESSION_BLOCK) {
                                      ast(AST_EQ, ==) {
                                        ast(AST_SYMBOL, n) {
                                        ast(AST_SYMBOL, 1) {
                                    ast(AST_COMMAND_BLOCK) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_RETURN) {
                                          ast(AST_SYMBOL, 1) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_ATTRIBUITION, result) {
                                      ast(AST_SYMBOL, 1) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_FLUX_CONTROLL_WHILE) {
                                        ast(AST_EXPRESSION_BLOCK) {
                                          ast(AST_GT, >) {
                                            ast(AST_SYMBOL, n) {
                                            ast(AST_SYMBOL, 1) {
                                        ast(AST_COMMAND_BLOCK) {
                                          ast(AST_COMMAND_LIST) {
                                            ast(AST_ATTRIBUITION, result) {
                                              ast(AST_ADD, +) {
                                                ast(AST_SYMBOL, a) {
                                                ast(AST_SYMBOL, b) {
                                            ast(AST_COMMAND_LIST) {
                                              ast(AST_ATTRIBUITION, a) {
                                                ast(AST_SYMBOL, b) {
                                              ast(AST_COMMAND_LIST) {
                                                ast(AST_ATTRIBUITION, b) {
                                                  ast(AST_SYMBOL, result) {
                                                ast(AST_COMMAND_LIST) {
                                                  ast(AST_ATTRIBUITION, n) {
```

```
                                                    ast(AST_SUB, __dash__) {
                                                      ast(AST_SYMBOL, n) {
                                                      ast(AST_SYMBOL, 1) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_RETURN) {
                                          ast(AST_SYMBOL, result) {
                          ast(AST_DECLARATION_LIST) {
                            ast(AST_DECLARATION_FUNCTION_INT, main) {
                              ast(AST_DECLARATION_FUNCTION_BODY) {
                                ast(AST_COMMAND_BLOCK) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_PRINT) {
                                      ast(AST_PRINT_STRING) {
                                        ast(AST_SYMBOL, Teste 1:) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_PRINT) {
                                        ast(AST_PRINT_STRING) {
                                          ast(AST_SYMBOL, Calculo fatorial e
fubinacci, [read, while, if, function call] \n) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_PRINT) {
                                          ast(AST_PRINT_STRING) {
                                            ast(AST_SYMBOL, \n) {
                                            ast(AST_PRINT_STRING) {
                                              ast(AST_SYMBOL, Enter the first
value: ) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_ATTRIBUITION,
input__dash__one) {
                                          ast(AST_READ) {
                                        ast(AST_COMMAND_LIST) {
                                          ast(AST_PRINT) {
                                            ast(AST_PRINT_STRING) {
                                              ast(AST_SYMBOL, Enter the second
value: ) {
                                          ast(AST_COMMAND_LIST) {
                                            ast(AST_ATTRIBUITION,
input__dash__two) {
                                              ast(AST_READ) {
                                            ast(AST_COMMAND_LIST) {
                                              ast(AST_ATTRIBUITION,
result__dash__fac) {
                                                ast(AST_FUNCTION_CALL,
fatorial) {
```

```
                                                    ast(AST_FUNCTION_CALL_ARGS)
{
                                                      ast(AST_SYMBOL,
input__dash__one) {
                                                  ast(AST_COMMAND_LIST) {
                                                    ast(AST_ATTRIBUITION,
result__dash__fib) {
                                                      ast(AST_FUNCTION_CALL,
fibonacci) {
ast(AST_FUNCTION_CALL_ARGS) {
                                                          ast(AST_SYMBOL,
input__dash__two) {
                                                    ast(AST_COMMAND_LIST) {
                                                      ast(AST_PRINT) {
                                                        ast(AST_PRINT_STRING) {
                                                          ast(AST_SYMBOL, \n0
valor de ) {
                                                          ast(AST_PRINT_EXP) {
                                                            ast(AST_SYMBOL,
input__dash__one) {
                                                            ast(AST_PRINT_STRING)
{
                                                              ast(AST_SYMBOL, ! é
) {
                                                              ast(AST_PRINT_EXP) {
                                                                ast(AST_SYMBOL,
result__dash__fac) {
                                                      ast(AST_COMMAND_LIST) {
ast(AST_FLUX_CONTROLL_IF|ELSE) {
ast(AST_EXPRESSION_BLOCK) {
                                                            ast(AST_GT, >) {
                                                              ast(AST_SYMBOL,
result__dash__fac) {
                                                              ast(AST_SYMBOL,
result__dash__fib) {
                                                          ast(AST_COMMAND_BLOCK) {
                                                            ast(AST_COMMAND_LIST)
{
                                                              ast(AST_PRINT) {
ast(AST_PRINT_STRING) {
```

```
                                                                ast(AST_SYMBOL,
que é maior do que o ) {
                                                          ast(AST_COMMAND_BLOCK) {
                                                            ast(AST_COMMAND_LIST)
{
                                                              ast(AST_PRINT) {
ast(AST_PRINT_STRING) {
                                                                  ast(AST_SYMBOL,
que é menor do que o ) {
                                                        ast(AST_COMMAND_LIST) {
                                                          ast(AST_PRINT) {
                                                            ast(AST_PRINT_EXP) {
                                                              ast(AST_SYMBOL,
input__dash__two) {
ast(AST_PRINT_STRING) {
                                                                ast(AST_SYMBOL, º
numero de fibonacci, que é ) {
                                                                ast(AST_PRINT_EXP)
{
                                                                  ast(AST_SYMBOL,
result__dash__fib) {
ast(AST_PRINT_STRING) {
ast(AST_SYMBOL, \n) {
                                                          ast(AST_COMMAND_LIST) {
                                                            ast(AST_ATTRIBUITION,
a) {
                                                              ast(AST_SUB,
__dash__) {
                                                                ast(AST_SYMBOL,
result__dash__fac) {
                                                                ast(AST_SYMBOL,
result__dash__fib) {
                                                            ast(AST_COMMAND_LIST)
{
ast(AST_ATTRIBUITION, b) {
                                                                ast(AST_SYMBOL, 0)
{
ast(AST_COMMAND_LIST) {
```

```
ast(AST_FLUX_CONTROLL_WHILE) {
ast(AST_EXPRESSION_BLOCK) {
                                                                ast(AST_GT, >)
{
ast(AST_SYMBOL, a) {
ast(AST_SYMBOL, 0) {
ast(AST_COMMAND_BLOCK) {
ast(AST_COMMAND_LIST) {
ast(AST_ATTRIBUITION, b) {
ast(AST_ADD, +) {
ast(AST_SYMBOL, b) {
ast(AST_SYMBOL, 1) {
ast(AST_COMMAND_LIST) {
ast(AST_ATTRIBUITION, a) {
ast(AST_SUB, __dash__) {
ast(AST_SYMBOL, a) {
ast(AST_SYMBOL, result__dash__fib) {
ast(AST_COMMAND_LIST) {
ast(AST_FLUX_CONTROLL_IF|ELSE) {
ast(AST_EXPRESSION_BLOCK) {
                                                                  ast(AST_LT,
<) {
ast(AST_SYMBOL, 0) {
ast(AST_SYMBOL, b) {
```

```
ast(AST_COMMAND_BLOCK) {

ast(AST_COMMAND_LIST) {

ast(AST_PRINT) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, \nO fatorial de ) {

ast(AST_PRINT_EXP) {

ast(AST_SYMBOL, input__dash__one) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL,  é, pelo menos, ) {

ast(AST_PRINT_EXP) {

ast(AST_SYMBOL, b) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, x maior que o ) {

ast(AST_PRINT_EXP) {

ast(AST_SYMBOL, input__dash__two) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, º numero de fibonacci.\n) {

hash after tacGenerateCode

Table[23]        has main        type 259        datatype 1      datavalue
0     datastring (null)
Table[24]        has _TMP_VAR_11         type 1607       datatype 5
datavalue 0     datastring \n
Table[36]        has _TMP_VAR_4          type 1607       datatype 0
datavalue 0     datastring (null)
Table[37]        has Enter the first value:              type 264
```

```
datatype 5      datavalue 0     datastring (null)
Table[42]       has *           type 271        datatype 0      datavalue
0     datastring (null)
Table[43]       has +           type 269        datatype 0      datavalue
0     datastring (null)
Table[47]       has _TMP_VAR_25         type 1607       datatype 5
datavalue 0     datastring º numero de fibonacci, que é
Table[48]       has 0           type 262        datatype 1      datavalue
0     datastring (null)
Table[49]       has 1           type 262        datatype 1      datavalue
1     datastring (null)
Table[50]       has _TMP_LABEL_1        type 265        datatype 0
datavalue 0     datastring (null)
Table[60]       has <           type 274        datatype 0      datavalue
0     datastring (null)
Table[60]       has ! é         type 264        datatype 5      datavalue
0     datastring (null)
Table[62]       has >           type 276        datatype 0      datavalue
0     datastring (null)
Table[75]       has Teste 1:            type 264        datatype 5
datavalue 0     datastring (null)
Table[77]       has fibonacci           type 259        datatype 1
datavalue 0     datastring (null)
Table[87]       has _TMP_VAR_15         type 1607       datatype 0
datavalue 0     datastring (null)
Table[97]       has a           type 258        datatype 1      datavalue
0     datastring (null)
Table[98]       has b           type 258        datatype 1      datavalue
1     datastring (null)
Table[99]       has _TMP_VAR_20         type 1607       datatype 5
datavalue 0     datastring \nO valor de
Table[110]      has n           type 258        datatype 1      datavalue
0     datastring (null)
Table[112]      has _TMP_VAR_3          type 1607       datatype 0
datavalue 0     datastring (null)
Table[150]      has _TMP_VAR_19         type 1607       datatype 5
datavalue 0     datastring ! é
Table[154]      has input__dash__one            type 258        datatype 1
datavalue 0     datastring (null)
Table[188]      has _TMP_VAR_2          type 1607       datatype 0
datavalue 0     datastring (null)
Table[220]      has º numero de fibonacci.\n            type 264
datatype 5      datavalue 0     datastring (null)
Table[236]      has _TMP_VAR_26         type 1607       datatype 0
```

```
datavalue 0     datastring (null)
Table[260]       has \n          type 264        datatype 5      datavalue
0     datastring (null)
Table[263]       has  é, pelo menos,             type 264        datatype 5
datavalue 0     datastring (null)
Table[264]       has _TMP_VAR_1          type 1607       datatype 0
datavalue 0     datastring (null)
Table[272]       has argsa       type 260        datatype 1      datavalue
0     datastring (null)
Table[288]       has _TMP_VAR_21         type 1607       datatype 0
datavalue 0     datastring (null)
Table[289]       has _TMP_VAR_12         type 1607       datatype 0
datavalue 0     datastring (null)
Table[340]       has _TMP_VAR_0          type 1607       datatype 0
datavalue 0     datastring (null)
Table[342]       has _TMP_LABEL_8        type 265        datatype 0
datavalue 0     datastring (null)
Table[344]       has  que é maior do que o       type 264        datatype 5
datavalue 0     datastring (null)
Table[346]       has result__dash__fac           type 258        datatype 1
datavalue 0     datastring (null)
Table[352]       has _TMP_VAR_16         type 1607       datatype 0
datavalue 0     datastring (null)
Table[357]       has \nO fatorial de             type 264        datatype 5
datavalue 0     datastring (null)
Table[371]       has  que é menor do que o       type 264        datatype 5
datavalue 0     datastring (null)
Table[401]       has _TMP_LABEL_6        type 265        datatype 0
datavalue 0     datastring (null)
Table[425]       has _TMP_VAR_27         type 1607       datatype 0
datavalue 0     datastring (null)
Table[439]       has _TMP_VAR_30         type 1607       datatype 0
datavalue 0     datastring (null)
Table[454]       has _TMP_LABEL_10       type 265        datatype 0
datavalue 0     datastring (null)
Table[460]       has _TMP_LABEL_4        type 265        datatype 0
datavalue 0     datastring (null)
Table[477]       has _TMP_VAR_22         type 1607       datatype 5
datavalue 0     datastring  que é maior do que o
Table[519]       has _TMP_LABEL_2        type 265        datatype 0
datavalue 0     datastring (null)
Table[552]       has _TMP_VAR_31         type 1607       datatype 5
datavalue 0     datastring º numero de fibonacci.\n
Table[554]       has _TMP_VAR_13         type 1607       datatype 5
```

```
datavalue 0     datastring Enter the second value:
Table[556]       has fatorial            type 259        datatype 1
datavalue 0     datastring (null)
Table[578]       has _TMP_LABEL_0        type 265        datatype 0
datavalue 0     datastring (null)
Table[581]       has Calculo fatorial e fubinacci, [read, while, if,
function call] \n           type 264        datatype 5      datavalue 0
datastring (null)
Table[614]       has _TMP_VAR_28         type 1607       datatype 0
datavalue 0     datastring (null)
Table[617]       has _TMP_VAR_17         type 1607       datatype 0
datavalue 0     datastring (null)
Table[653]       has _TMP_VAR_9          type 1607       datatype 5
datavalue 0     datastring Calculo fatorial e fubinacci, [read, while, if,
function call] \n
Table[665]       has _TMP_VAR_32         type 1607       datatype 5
datavalue 0     datastring x maior que o
Table[666]       has _TMP_VAR_23         type 1607       datatype 5
datavalue 0     datastring  que é menor do que o
Table[686]       has input__dash__two            type 258        datatype 1
datavalue 0     datastring (null)
Table[717]       has Enter the second value:             type 264
datatype 5      datavalue 0     datastring (null)
Table[729]       has _TMP_VAR_8          type 1607       datatype 5
datavalue 0     datastring Teste 1:
Table[748]       has result      type 258        datatype 1      datavalue
0     datastring (null)
Table[756]       has _TMP_VAR_10         type 1607       datatype 5
datavalue 0     datastring Enter the first value:
Table[778]       has _TMP_VAR_33         type 1607       datatype 5
datavalue 0     datastring  é, pelo menos,
Table[791]       has ==          type 277        datatype 0      datavalue
0     datastring (null)
Table[799]       has argsb       type 260        datatype 1      datavalue
0     datastring (null)
Table[803]       has _TMP_VAR_29         type 1607       datatype 0
datavalue 0     datastring (null)
Table[805]       has _TMP_VAR_7          type 1607       datatype 0
datavalue 0     datastring (null)
Table[809]       has result__dash__fib           type 258        datatype 1
datavalue 0     datastring (null)
Table[811]       has _TMP_LABEL_9        type 265        datatype 0
datavalue 0     datastring (null)
Table[814]       has __dash__            type 270        datatype 0
```

```
datavalue 0     datastring (null)
Table[819]       has _TMP_VAR_14         type 1607       datatype 0
datavalue 0     datastring (null)
Table[855]       has _TMP_VAR_24         type 1607       datatype 5
datavalue 0     datastring \n
Table[870]       has _TMP_LABEL_7        type 265        datatype 0
datavalue 0     datastring (null)
Table[874]       has \nO valor de        type 264        datatype 5
datavalue 0     datastring (null)
Table[881]       has _TMP_VAR_6          type 1607       datatype 0
datavalue 0     datastring (null)
Table[882]       has _TMP_VAR_18         type 1607       datatype 0
datavalue 0     datastring (null)
Table[891]       has _TMP_VAR_34         type 1607       datatype 5
datavalue 0     datastring \nO fatorial de
Table[900]       has º numero de fibonacci, que é        type 264
datatype 5      datavalue 0     datastring (null)
Table[913]       has x maior que o       type 264        datatype 5
datavalue 0     datastring (null)
Table[929]       has _TMP_LABEL_5        type 265        datatype 0
datavalue 0     datastring (null)
Table[957]       has _TMP_VAR_5          type 1607       datatype 0
datavalue 0     datastring (null)
Table[988]       has _TMP_LABEL_3        type 265        datatype 0
datavalue 0     datastring (null)


TAC(TAC_DEC_GLOBAL_INT, input__dash__one, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, input__dash__two, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, result__dash__fac, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, a, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, b, 1, 0);
TAC(TAC_DEC_GLOBAL_INT, result__dash__fib, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, result, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, n, 0, 0);
TAC(TAC_DEC_FUNC_ARGS, argsa, 0, 0);
TAC(TAC_BEGINFUN, fatorial, result, 0);
TAC(TAC_MOVE, n, argsa, 0);
TAC(TAC_MOVE, result, 1, 0);
TAC(TAC_LABEL, _TMP_LABEL_0, 0, 0);
TAC(TAC_GT, _TMP_VAR_0, n, 1);
TAC(TAC_JMPZ, _TMP_LABEL_1, _TMP_VAR_0, 0);
TAC(TAC_MULT, _TMP_VAR_1, result, n);
TAC(TAC_MOVE, result, _TMP_VAR_1, 0);
```

```
TAC(TAC_SUB, _TMP_VAR_2, n, 1);
TAC(TAC_MOVE, n, _TMP_VAR_2, 0);
TAC(TAC_JMP, _TMP_LABEL_0, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_1, 0, 0);
TAC(TAC_RETURN, result, 0, 0);
TAC(TAC_ENDFUN, fatorial, argsa, 0);
TAC(TAC_DEC_FUNC_ARGS, argsb, 0, 0);
TAC(TAC_BEGINFUN, fibonacci, result, 0);
TAC(TAC_MOVE, n, argsb, 0);
TAC(TAC_EQ, _TMP_VAR_3, n, 0);
TAC(TAC_JMPZ, _TMP_LABEL_2, _TMP_VAR_3, 0);
TAC(TAC_RETURN, 0, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_2, 0, 0);
TAC(TAC_EQ, _TMP_VAR_4, n, 1);
TAC(TAC_JMPZ, _TMP_LABEL_3, _TMP_VAR_4, 0);
TAC(TAC_RETURN, 1, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_3, 0, 0);
TAC(TAC_MOVE, result, 1, 0);
TAC(TAC_LABEL, _TMP_LABEL_4, 0, 0);
TAC(TAC_GT, _TMP_VAR_5, n, 1);
TAC(TAC_JMPZ, _TMP_LABEL_5, _TMP_VAR_5, 0);
TAC(TAC_ADD, _TMP_VAR_6, a, b);
TAC(TAC_MOVE, result, _TMP_VAR_6, 0);
TAC(TAC_MOVE, a, b, 0);
TAC(TAC_MOVE, b, result, 0);
TAC(TAC_SUB, _TMP_VAR_7, n, 1);
TAC(TAC_MOVE, n, _TMP_VAR_7, 0);
TAC(TAC_JMP, _TMP_LABEL_4, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_5, 0, 0);
TAC(TAC_RETURN, result, 0, 0);
TAC(TAC_ENDFUN, fibonacci, argsb, 0);
TAC(TAC_BEGINFUN, main, _TMP_LABEL_10, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_8, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_8, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_9, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_9, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_11, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_10, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_10, 0, 0);
TAC(TAC_READ, _TMP_VAR_12, 0, 0);
TAC(TAC_MOVE, input__dash__one, _TMP_VAR_12, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_13, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_13, 0, 0);
TAC(TAC_READ, _TMP_VAR_14, 0, 0);
```

```
TAC(TAC_MOVE, input__dash__two, _TMP_VAR_14, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_15, input__dash__one, 0);
TAC(TAC_FUN_CALL, _TMP_VAR_16, fatorial, 0);
TAC(TAC_MOVE, result__dash__fac, _TMP_VAR_16, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_17, input__dash__two, 0);
TAC(TAC_FUN_CALL, _TMP_VAR_18, fibonacci, 0);
TAC(TAC_MOVE, result__dash__fib, _TMP_VAR_18, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_20, 0, 0);
TAC(TAC_PRINT_INT, input__dash__one, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_19, 0, 0);
TAC(TAC_PRINT_INT, result__dash__fac, 0, 0);
TAC(TAC_PRINT, result__dash__fac, 0, 0);
TAC(TAC_GT, _TMP_VAR_21, result__dash__fac, result__dash__fib);
TAC(TAC_JMPZ, _TMP_LABEL_6, _TMP_VAR_21, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_22, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_22, 0, 0);
TAC(TAC_JMP, _TMP_LABEL_7, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_6, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_23, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_23, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_7, 0, 0);
TAC(TAC_PRINT_INT, input__dash__two, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_25, 0, 0);
TAC(TAC_PRINT_INT, result__dash__fib, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_24, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_24, 0, 0);
TAC(TAC_SUB, _TMP_VAR_26, result__dash__fac, result__dash__fib);
TAC(TAC_MOVE, a, _TMP_VAR_26, 0);
TAC(TAC_MOVE, b, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_8, 0, 0);
TAC(TAC_GT, _TMP_VAR_27, a, 0);
TAC(TAC_JMPZ, _TMP_LABEL_9, _TMP_VAR_27, 0);
TAC(TAC_ADD, _TMP_VAR_28, b, 1);
TAC(TAC_MOVE, b, _TMP_VAR_28, 0);
TAC(TAC_SUB, _TMP_VAR_29, a, result__dash__fib);
TAC(TAC_MOVE, a, _TMP_VAR_29, 0);
TAC(TAC_JMP, _TMP_LABEL_8, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_9, 0, 0);
TAC(TAC_LT, _TMP_VAR_30, 0, b);
TAC(TAC_JMPZ, _TMP_LABEL_10, _TMP_VAR_30, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_34, 0, 0);
TAC(TAC_PRINT_INT, input__dash__one, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_33, 0, 0);
TAC(TAC_PRINT_INT, b, 0, 0);
```

```
TAC(TAC_PRINT_STRING, _TMP_VAR_32, 0, 0);
TAC(TAC_PRINT_INT, input__dash__two, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_31, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_31, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_10, 0, 0);
TAC(TAC_ENDFUN, main, 0, 0);
condition: (n==0) body_if: {
return 0;

} body_else:

.condition: (n==1) body_if: {
return 1;

} body_else:

.condition: (result__dash__fac>result__dash__fib) body_if: {
print " que é maior do que o ";

} body_else:{
print " que é menor do que o ";

}

.condition: (0<b) body_if: {
print "\nO fatorial de ", input__dash__one, " é, pelo menos, ", b, "x maior
que o ", input__dash__two, "º numero de fibonacci.\n";

} body_else:

.Numero de linhas: 83.
Compilation Success.
Teste 1:Calculo fatorial e fubinacci, [read, while, if, function call]

Enter the first value: 7
Enter the second value: 12

O valor de 7! é 5040 que é maior do que o 12º numero de fibonacci, que é
144

O fatorial de 7 é, pelo menos, 34x maior que o 12º numero de fibonacci.
```

Programa de teste 3

```
patrick@DESKTOP-5MBUUP8:/mnt/d/UFRGS/2021-2/compiladores/Aulas/Trabalho/eta
pa6$ make clean && make && ./etapa6 test_programs/test_3.txt output.txt &&
gcc test_programs/test_3.s && ./a.out
rm lex.yy.c y.tab.c y.tab.h *.o etapa6
gcc -c -g main.c
yacc -d parser.y
parser.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
lex scanner.l
gcc -c -g  lex.yy.c
gcc -c -g  y.tab.c
gcc -c -g  hash.c
gcc -c -g  ast.c
gcc -c -g  semantic.c
gcc -c -g tacs.c
gcc -c -g  asm.c
gcc -c -g commons.c
gcc main.o lex.yy.o y.tab.o hash.o ast.o semantic.o tacs.o asm.o commons.o
-g -o etapa6


Semantics start

Undeclared semantic errors: 0

ast(AST_PROGRAM) {
 ast(AST_DECLARATION_LIST) {
   ast(AST_DECLARATION_GLOBAL_INT, input) {
     ast(AST_SYMBOL, 0) {
   ast(AST_DECLARATION_LIST) {
     ast(AST_DECLARATION_GLOBAL_INT, result) {
       ast(AST_SYMBOL, 0) {
     ast(AST_DECLARATION_LIST) {
       ast(AST_DECLARATION_GLOBAL_INT, n) {
         ast(AST_SYMBOL, 0) {
       ast(AST_DECLARATION_LIST) {
         ast(AST_DECLARATION_GLOBAL_INT, a) {
           ast(AST_SYMBOL, 0) {
         ast(AST_DECLARATION_LIST) {
           ast(AST_DECLARATION_GLOBAL_INT, b) {
             ast(AST_SYMBOL, 0) {
           ast(AST_DECLARATION_LIST) {
             ast(AST_DECLARATION_GLOBAL_INT, total_n) {
               ast(AST_SYMBOL, 30) {
```

```
            ast(AST_DECLARATION_LIST) {
              ast(AST_DECLARATION_GLOBAL_ARRAY_INT, arr) {
                ast(AST_SYMBOL, 20) {
              ast(AST_DECLARATION_LIST) {
                ast(AST_DECLARATION_GLOBAL_INT, columns) {
                  ast(AST_SYMBOL, 6) {
                ast(AST_DECLARATION_LIST) {
                  ast(AST_DECLARATION_GLOBAL_INT, count) {
                    ast(AST_SYMBOL, 0) {
                  ast(AST_DECLARATION_LIST) {
                    ast(AST_DECLARATION_FUNCTION_INT, fibonacci) {
                      ast(AST_DECLARATION_FUNCTION_ARGS_INT, argsb) {
                      ast(AST_DECLARATION_FUNCTION_BODY) {
                        ast(AST_COMMAND_BLOCK) {
                          ast(AST_COMMAND_LIST) {
                            ast(AST_ATTRIBUITION, a) {
                              ast(AST_SYMBOL, 0) {
                            ast(AST_COMMAND_LIST) {
                              ast(AST_ATTRIBUITION, b) {
                                ast(AST_SYMBOL, 1) {
                              ast(AST_COMMAND_LIST) {
                                ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                  ast(AST_EXPRESSION_BLOCK) {
                                    ast(AST_EQ, ==) {
                                      ast(AST_SYMBOL, argsb) {
                                      ast(AST_SYMBOL, 0) {
                                  ast(AST_COMMAND_BLOCK) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_RETURN) {
                                        ast(AST_SYMBOL, 0) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                    ast(AST_EXPRESSION_BLOCK) {
                                      ast(AST_EQ, ==) {
                                        ast(AST_SYMBOL, argsb) {
                                        ast(AST_SYMBOL, 1) {
                                    ast(AST_COMMAND_BLOCK) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_RETURN) {
                                          ast(AST_SYMBOL, 1) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_ATTRIBUITION, result) {
                                      ast(AST_SYMBOL, 1) {
                                    ast(AST_COMMAND_LIST) {
```

```
                                        ast(AST_FLUX_CONTROLL_WHILE) {
                                          ast(AST_EXPRESSION_BLOCK) {
                                            ast(AST_GT, >) {
                                              ast(AST_SYMBOL, argsb) {
                                              ast(AST_SYMBOL, 1) {
                                          ast(AST_COMMAND_BLOCK) {
                                            ast(AST_COMMAND_LIST) {
                                              ast(AST_ATTRIBUITION, result)
{
                                                ast(AST_ADD, +) {
                                                  ast(AST_SYMBOL, a) {
                                                  ast(AST_SYMBOL, b) {
                                                ast(AST_COMMAND_LIST) {
                                                  ast(AST_ATTRIBUITION, a) {
                                                    ast(AST_SYMBOL, b) {
                                                  ast(AST_COMMAND_LIST) {
                                                    ast(AST_ATTRIBUITION, b) {
                                                      ast(AST_SYMBOL, result)
{
                                                    ast(AST_COMMAND_LIST) {
                                                      ast(AST_ATTRIBUITION,
argsb) {
                                                        ast(AST_SUB, __dash__)
{
                                                          ast(AST_SYMBOL,
argsb) {
                                                          ast(AST_SYMBOL, 1) {
                                        ast(AST_COMMAND_LIST) {
                                          ast(AST_RETURN) {
                                            ast(AST_SYMBOL, result) {
                      ast(AST_DECLARATION_LIST) {
                        ast(AST_DECLARATION_FUNCTION_INT, main) {
                          ast(AST_DECLARATION_FUNCTION_BODY) {
                            ast(AST_COMMAND_BLOCK) {
                              ast(AST_COMMAND_LIST) {
                                ast(AST_PRINT) {
                                  ast(AST_PRINT_STRING) {
                                    ast(AST_SYMBOL, Teste 3:) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_PRINT) {
                                    ast(AST_PRINT_STRING) {
                                      ast(AST_SYMBOL, Arrays \n) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_PRINT) {
```

```
                                      ast(AST_PRINT_STRING) {
                                        ast(AST_SYMBOL, \n) {
                                        ast(AST_PRINT_STRING) {
                                          ast(AST_SYMBOL, Enter a number: )
{
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_ATTRIBUITION, input) {
                                        ast(AST_READ) {
                                      ast(AST_COMMAND_LIST) {
                                        ast(AST_ATTRIBUITION, n) {
                                          ast(AST_SYMBOL, 0) {
                                        ast(AST_COMMAND_LIST) {
                                          ast(AST_FLUX_CONTROLL_WHILE) {
                                            ast(AST_EXPRESSION_BLOCK) {
                                              ast(AST_LT, <) {
                                                ast(AST_SYMBOL, n) {
                                                ast(AST_SYMBOL, total_n) {
                                            ast(AST_COMMAND_BLOCK) {
                                              ast(AST_COMMAND_LIST) {
                                                ast(AST_ARRAY_ATTRIBUITION,
arr) {
                                                  ast(AST_SYMBOL, n) {
                                                  ast(AST_FUNCTION_CALL,
fibonacci) {

ast(AST_FUNCTION_CALL_ARGS) {
                                                      ast(AST_SYMBOL, n) {
                                                ast(AST_COMMAND_LIST) {
                                                  ast(AST_ATTRIBUITION, n) {
                                                    ast(AST_ADD, +) {
                                                      ast(AST_SYMBOL, n) {
                                                      ast(AST_SYMBOL, 1) {
                                          ast(AST_COMMAND_LIST) {
                                            ast(AST_ATTRIBUITION, n) {
                                              ast(AST_SYMBOL, 0) {
                                            ast(AST_COMMAND_LIST) {
                                              ast(AST_PRINT) {
                                                ast(AST_PRINT_STRING) {
                                                  ast(AST_SYMBOL, \nOs ) {
                                                  ast(AST_PRINT_EXP) {
                                                    ast(AST_SYMBOL, total_n)
{
                                                      ast(AST_PRINT_STRING) {
                                                        ast(AST_SYMBOL,
```

```
primeiros numeros da sequencia de fibonacci são: \n) {
                                                ast(AST_COMMAND_LIST) {
                                                  ast(AST_FLUX_CONTROLL_WHILE)
{
                                                    ast(AST_EXPRESSION_BLOCK)
{
                                                      ast(AST_LT, <) {
                                                        ast(AST_SYMBOL, n) {
                                                        ast(AST_SYMBOL,
total_n) {
                                                    ast(AST_COMMAND_BLOCK) {
                                                      ast(AST_COMMAND_LIST) {

ast(AST_ARRAY_ATTRIBUITION, arr) {
                                                          ast(AST_SYMBOL, n) {

ast(AST_FUNCTION_CALL, fibonacci) {

ast(AST_FUNCTION_CALL_ARGS) {
                                                             ast(AST_SYMBOL,
n) {
                                                        ast(AST_COMMAND_LIST)
{

ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                                           ast(AST_EQ, ==) {
                                                             ast(AST_SYMBOL,
input) {

ast(AST_ARRAY_ELEMENT, arr) {

ast(AST_SYMBOL, n) {
                                                           ast(AST_PRINT) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, [__dash__>) {

ast(AST_PRINT_EXP) {

ast(AST_ARRAY_ELEMENT, arr) {

ast(AST_SYMBOL, n) {
```

```
ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, <__dash__] ) {
                                                        ast(AST_PRINT) {

ast(AST_PRINT_EXP) {

ast(AST_ARRAY_ELEMENT, arr) {

ast(AST_SYMBOL, n) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, \t) {

ast(AST_COMMAND_LIST) {

ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                                      ast(AST_GT, >) {

ast(AST_SYMBOL, input) {

ast(AST_ARRAY_ELEMENT, arr) {

ast(AST_SYMBOL, n) {

ast(AST_FLUX_CONTROLL_IF|ELSE) {
                                                        ast(AST_LT, <)
{

ast(AST_SYMBOL, input) {

ast(AST_ARRAY_ELEMENT, arr) {

ast(AST_ADD, +) {

ast(AST_SYMBOL, n) {

ast(AST_SYMBOL, 1) {
                                                        ast(AST_PRINT)
{

ast(AST_PRINT_STRING) {
```

```
ast(AST_SYMBOL, [*] ) {

ast(AST_COMMAND_LIST) {

ast(AST_ATTRIBUITION, n) {
                                                                ast(AST_ADD,
+) {

ast(AST_SYMBOL, n) {

ast(AST_SYMBOL, 1) {

ast(AST_COMMAND_LIST) {

ast(AST_FLUX_CONTROLL_IF|ELSE) {

ast(AST_EXPRESSION_BLOCK) {

ast(AST_GTE, >=) {

ast(AST_SYMBOL, count) {

ast(AST_SYMBOL, columns) {

ast(AST_COMMAND_BLOCK) {

ast(AST_COMMAND_LIST) {

ast(AST_PRINT) {

ast(AST_PRINT_STRING) {

ast(AST_SYMBOL, \n) {

ast(AST_COMMAND_LIST) {

ast(AST_ATTRIBUITION, count) {

ast(AST_SYMBOL, 0) {

ast(AST_COMMAND_BLOCK) {

ast(AST_COMMAND_LIST) {
```

```
ast(AST_ATTRIBUITION, count) {

ast(AST_ADD, +) {

ast(AST_SYMBOL, count) {

ast(AST_SYMBOL, 1) {
                                                ast(AST_COMMAND_LIST) {
                                                  ast(AST_PRINT) {
                                                    ast(AST_PRINT_STRING) {
                                                      ast(AST_SYMBOL, \n) {

hash after tacGenerateCode

Table[7]         has _TMP_VAR_35         type 1607       datatype 0
datavalue 0     datastring (null)
Table[23]        has main        type 259        datatype 1      datavalue
0     datastring (null)
Table[24]        has _TMP_VAR_11         type 1607       datatype 0
datavalue 0     datastring (null)
Table[35]        has Teste 3:            type 264        datatype 5
datavalue 0     datastring (null)
Table[36]        has _TMP_VAR_4          type 1607       datatype 0
datavalue 0     datastring (null)
Table[43]        has +           type 269        datatype 0      datavalue
0     datastring (null)
Table[47]        has _TMP_VAR_25         type 1607       datatype 5
datavalue 0     datastring \t
Table[48]        has 0           type 262        datatype 1      datavalue
0     datastring (null)
Table[49]        has 1           type 262        datatype 1      datavalue
1     datastring (null)
Table[50]        has _TMP_LABEL_1        type 265        datatype 0
datavalue 0     datastring (null)
Table[54]        has 6           type 262        datatype 1      datavalue
6     datastring (null)
Table[60]        has <           type 274        datatype 0      datavalue
0     datastring (null)
Table[62]        has >           type 276        datatype 0      datavalue
0     datastring (null)
Table[77]        has fibonacci           type 259        datatype 1
datavalue 0     datastring (null)
Table[87]        has _TMP_VAR_15         type 1607       datatype 5
datavalue 0     datastring \nOs
```

```
Table[97]        has a           type 258        datatype 1      datavalue
0     datastring (null)
Table[98]        has b           type 258        datatype 1      datavalue
0     datastring (null)
Table[99]        has _TMP_VAR_20         type 1607       datatype 0
datavalue 0     datastring (null)
Table[110]       has n           type 258        datatype 1      datavalue
0     datastring (null)
Table[112]       has _TMP_VAR_3          type 1607       datatype 0
datavalue 0     datastring (null)
Table[120]       has _TMP_VAR_36         type 1607       datatype 5
datavalue 0     datastring \n
Table[126]       has Arrays \n           type 264        datatype 5
datavalue 0     datastring (null)
Table[150]       has _TMP_VAR_19         type 1607       datatype 0
datavalue 0     datastring (null)
Table[154]       has count       type 258        datatype 1      datavalue
0     datastring (null)
Table[174]       has  primeiros numeros da sequencia de fibonacci são: \n
type 264        datatype 5      datavalue 0     datastring (null)
Table[180]       has <__dash__]          type 264        datatype 5
datavalue 0     datastring (null)
Table[188]       has _TMP_VAR_2          type 1607       datatype 0
datavalue 0     datastring (null)
Table[192]       has [__dash__>          type 264        datatype 5
datavalue 0     datastring (null)
Table[199]       has Enter a number:             type 264        datatype 5
datavalue 0     datastring (null)
Table[236]       has _TMP_VAR_26         type 1607       datatype 0
datavalue 0     datastring (null)
Table[260]       has \n          type 264        datatype 5      datavalue
0     datastring (null)
Table[264]       has _TMP_VAR_1          type 1607       datatype 0
datavalue 0     datastring (null)
Table[288]       has _TMP_VAR_21         type 1607       datatype 0
datavalue 0     datastring (null)
Table[289]       has _TMP_VAR_12         type 1607       datatype 0
datavalue 0     datastring (null)
Table[325]       has total_n             type 258        datatype 1
datavalue 30    datastring (null)
Table[340]       has _TMP_VAR_0          type 1607       datatype 0
datavalue 0     datastring (null)
Table[342]       has _TMP_LABEL_8        type 265        datatype 0
datavalue 0     datastring (null)
```

```
Table[352]       has _TMP_VAR_16         type 1607       datatype 0
datavalue 0     datastring (null)
Table[365]       has columns             type 258        datatype 1
datavalue 6     datastring (null)
Table[401]       has _TMP_LABEL_6        type 265        datatype 0
datavalue 0     datastring (null)
Table[425]       has _TMP_VAR_27         type 1607       datatype 0
datavalue 0     datastring (null)
Table[439]       has _TMP_VAR_30         type 1607       datatype 0
datavalue 0     datastring (null)
Table[440]       has input       type 258        datatype 1      datavalue
0     datastring (null)
Table[454]       has _TMP_LABEL_10       type 265        datatype 0
datavalue 0     datastring (null)
Table[454]       has 20          type 262        datatype 1      datavalue
20    datastring (null)
Table[460]       has _TMP_LABEL_4        type 265        datatype 0
datavalue 0     datastring (null)
Table[477]       has _TMP_VAR_22         type 1607       datatype 5
datavalue 0     datastring <__dash__]
Table[502]       has 30          type 262        datatype 1      datavalue
30    datastring (null)
Table[505]       has _TMP_LABEL_11       type 265        datatype 0
datavalue 0     datastring (null)
Table[519]       has _TMP_LABEL_2        type 265        datatype 0
datavalue 0     datastring (null)
Table[552]       has _TMP_VAR_31         type 1607       datatype 5
datavalue 0     datastring [*]
Table[553]       has arr         type 261        datatype 10     datavalue
20    datastring (null)
Table[554]       has _TMP_VAR_13         type 1607       datatype 0
datavalue 0     datastring (null)
Table[556]       has _TMP_LABEL_12       type 265        datatype 0
datavalue 0     datastring (null)
Table[578]       has _TMP_LABEL_0        type 265        datatype 0
datavalue 0     datastring (null)
Table[607]       has _TMP_LABEL_13       type 265        datatype 0
datavalue 0     datastring (null)
Table[614]       has _TMP_VAR_28         type 1607       datatype 0
datavalue 0     datastring (null)
Table[617]       has _TMP_VAR_17         type 1607       datatype 0
datavalue 0     datastring (null)
Table[653]       has _TMP_VAR_9          type 1607       datatype 0
datavalue 0     datastring (null)
```

```
Table[665]       has _TMP_VAR_32         type 1607       datatype 0
datavalue 0     datastring (null)
Table[666]       has _TMP_VAR_23         type 1607       datatype 5
datavalue 0     datastring [__dash__>
Table[729]       has _TMP_VAR_8          type 1607       datatype 5
datavalue 0     datastring \n
Table[748]       has result      type 258        datatype 1      datavalue
0     datastring (null)
Table[756]       has _TMP_VAR_10         type 1607       datatype 0
datavalue 0     datastring (null)
Table[778]       has _TMP_VAR_33         type 1607       datatype 0
datavalue 0     datastring (null)
Table[791]       has ==          type 277        datatype 0      datavalue
0     datastring (null)
Table[799]       has argsb       type 260        datatype 1      datavalue
0     datastring (null)
Table[803]       has _TMP_VAR_29         type 1607       datatype 0
datavalue 0     datastring (null)
Table[805]       has _TMP_VAR_7          type 1607       datatype 5
datavalue 0     datastring Enter a number:
Table[811]       has _TMP_LABEL_9        type 265        datatype 0
datavalue 0     datastring (null)
Table[814]       has __dash__            type 270        datatype 0
datavalue 0     datastring (null)
Table[818]       has \t          type 264        datatype 5      datavalue
0     datastring (null)
Table[819]       has _TMP_VAR_14         type 1607       datatype 5
datavalue 0     datastring  primeiros numeros da sequencia de fibonacci
são: \n
Table[852]       has >=          type 275        datatype 0      datavalue
0     datastring (null)
Table[855]       has _TMP_VAR_24         type 1607       datatype 0
datavalue 0     datastring (null)
Table[870]       has _TMP_LABEL_7        type 265        datatype 0
datavalue 0     datastring (null)
Table[880]       has [*]         type 264        datatype 5      datavalue
0     datastring (null)
Table[881]       has _TMP_VAR_6          type 1607       datatype 5
datavalue 0     datastring Arrays \n
Table[882]       has _TMP_VAR_18         type 1607       datatype 0
datavalue 0     datastring (null)
Table[891]       has _TMP_VAR_34         type 1607       datatype 5
datavalue 0     datastring \n
Table[929]       has _TMP_LABEL_5        type 265        datatype 0
```

```
datavalue 0     datastring (null)
Table[957]       has _TMP_VAR_5          type 1607       datatype 5
datavalue 0     datastring Teste 3:
Table[959]       has \nOs        type 264        datatype 5      datavalue
0     datastring (null)
Table[988]       has _TMP_LABEL_3        type 265        datatype 0
datavalue 0     datastring (null)


TAC(TAC_DEC_GLOBAL_INT, input, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, result, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, n, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, a, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, b, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, total_n, 30, 0);
TAC(TAC_DEC_GLOBAL_ARR, arr, 0, 20);
TAC(TAC_DEC_GLOBAL_INT, columns, 6, 0);
TAC(TAC_DEC_GLOBAL_INT, count, 0, 0);
TAC(TAC_DEC_FUNC_ARGS, argsb, 0, 0);
TAC(TAC_BEGINFUN, fibonacci, result, 0);
TAC(TAC_MOVE, a, 0, 0);
TAC(TAC_MOVE, b, 1, 0);
TAC(TAC_EQ, _TMP_VAR_0, argsb, 0);
TAC(TAC_JMPZ, _TMP_LABEL_0, _TMP_VAR_0, 0);
TAC(TAC_RETURN, 0, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_0, 0, 0);
TAC(TAC_EQ, _TMP_VAR_1, argsb, 1);
TAC(TAC_JMPZ, _TMP_LABEL_1, _TMP_VAR_1, 0);
TAC(TAC_RETURN, 1, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_1, 0, 0);
TAC(TAC_MOVE, result, 1, 0);
TAC(TAC_LABEL, _TMP_LABEL_2, 0, 0);
TAC(TAC_GT, _TMP_VAR_2, argsb, 1);
TAC(TAC_JMPZ, _TMP_LABEL_3, _TMP_VAR_2, 0);
TAC(TAC_ADD, _TMP_VAR_3, a, b);
TAC(TAC_MOVE, result, _TMP_VAR_3, 0);
TAC(TAC_MOVE, a, b, 0);
TAC(TAC_MOVE, b, result, 0);
TAC(TAC_SUB, _TMP_VAR_4, argsb, 1);
TAC(TAC_MOVE, argsb, _TMP_VAR_4, 0);
TAC(TAC_JMP, _TMP_LABEL_2, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_3, 0, 0);
TAC(TAC_RETURN, result, 0, 0);
TAC(TAC_ENDFUN, fibonacci, argsb, 0);
```

```
TAC(TAC_BEGINFUN, main, _TMP_VAR_36, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_5, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_5, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_6, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_6, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_8, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_7, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_7, 0, 0);
TAC(TAC_READ, _TMP_VAR_9, 0, 0);
TAC(TAC_MOVE, input, _TMP_VAR_9, 0);
TAC(TAC_MOVE, n, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_4, 0, 0);
TAC(TAC_LT, _TMP_VAR_10, n, total_n);
TAC(TAC_JMPZ, _TMP_LABEL_5, _TMP_VAR_10, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_11, n, 0);
TAC(TAC_FUN_CALL, _TMP_VAR_12, fibonacci, 0);
TAC(TAC_ARR_SET_ELEMENT, arr, n, _TMP_VAR_12);
TAC(TAC_ADD, _TMP_VAR_13, n, 1);
TAC(TAC_MOVE, n, _TMP_VAR_13, 0);
TAC(TAC_JMP, _TMP_LABEL_4, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_5, 0, 0);
TAC(TAC_MOVE, n, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_15, 0, 0);
TAC(TAC_PRINT_INT, total_n, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_14, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_14, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_12, 0, 0);
TAC(TAC_LT, _TMP_VAR_16, n, total_n);
TAC(TAC_JMPZ, _TMP_LABEL_13, _TMP_VAR_16, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_17, n, 0);
TAC(TAC_FUN_CALL, _TMP_VAR_18, fibonacci, 0);
TAC(TAC_ARR_SET_ELEMENT, arr, n, _TMP_VAR_18);
TAC(TAC_ARR_GET_ELEMENT, _TMP_VAR_19, arr, n);
TAC(TAC_EQ, _TMP_VAR_20, input, _TMP_VAR_19);
TAC(TAC_JMPZ, _TMP_LABEL_6, _TMP_VAR_20, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_23, 0, 0);
TAC(TAC_ARR_GET_ELEMENT, _TMP_VAR_21, arr, n);
TAC(TAC_PRINT_INT, _TMP_VAR_21, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_22, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_22, 0, 0);
TAC(TAC_JMP, _TMP_LABEL_7, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_6, 0, 0);
TAC(TAC_ARR_GET_ELEMENT, _TMP_VAR_24, arr, n);
TAC(TAC_PRINT_INT, _TMP_VAR_24, 0, 0);
```

```
TAC(TAC_PRINT_STRING, _TMP_VAR_25, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_25, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_7, 0, 0);
TAC(TAC_ARR_GET_ELEMENT, _TMP_VAR_26, arr, n);
TAC(TAC_GT, _TMP_VAR_27, input, _TMP_VAR_26);
TAC(TAC_JMPZ, _TMP_LABEL_9, _TMP_VAR_27, 0);
TAC(TAC_ADD, _TMP_VAR_28, n, 1);
TAC(TAC_ARR_GET_ELEMENT, _TMP_VAR_29, arr, _TMP_VAR_28);
TAC(TAC_LT, _TMP_VAR_30, input, _TMP_VAR_29);
TAC(TAC_JMPZ, _TMP_LABEL_8, _TMP_VAR_30, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_31, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_31, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_8, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_9, 0, 0);
TAC(TAC_ADD, _TMP_VAR_32, n, 1);
TAC(TAC_MOVE, n, _TMP_VAR_32, 0);
TAC(TAC_GTE, _TMP_VAR_33, count, columns);
TAC(TAC_JMPZ, _TMP_LABEL_10, _TMP_VAR_33, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_34, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_34, 0, 0);
TAC(TAC_MOVE, count, 0, 0);
TAC(TAC_JMP, _TMP_LABEL_11, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_10, 0, 0);
TAC(TAC_ADD, _TMP_VAR_35, count, 1);
TAC(TAC_MOVE, count, _TMP_VAR_35, 0);
TAC(TAC_LABEL, _TMP_LABEL_11, 0, 0);
TAC(TAC_JMP, _TMP_LABEL_12, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_13, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_36, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_36, 0, 0);
TAC(TAC_ENDFUN, main, 0, 0);
values:
condition: (==0) body_if: {
return 0;

} body_else:

.condition: (==1) body_if: {
return 1;

} body_else:

.condition: input==arr[n] body_if: print "[__dash__>", arr[n], "<__dash__]
" body_else:print arr[n], "\t"
```

```
.condition: input<arr[n+1] body_if: print "[*] " body_else:

.condition: input>arr[n] body_if: if input<arr[n+1] then print "[*] "
body_else:

.condition: (count>=columns) body_if: {
print "\n";
count=0;

} body_else:{
count=count+1;

}

.Numero de linhas: 76.
Compilation Success.
Teste 3:Arrays

Enter a number: 610

Os 30 primeiros numeros da sequencia de fibonacci são:
0       1       1       2       3       5       8
13      21      34      55      89      144     233
377     [->610<-] 987   1597    2584    4181    6765
10946   17711   28657   46368   75025   121393  196418
317811  514229
```

Programa de teste 4

```
patrick@DESKTOP-5MBUUP8:/mnt/d/UFRGS/2021-2/compiladores/Aulas/Trabalho/eta
pa6$ make clean && make && ./etapa6 test_programs/test_4.txt output.txt &&
gcc test_programs/test_4.s && ./a.out
rm lex.yy.c y.tab.c y.tab.h *.o etapa6
gcc -c -g main.c
yacc -d parser.y
parser.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
lex scanner.l
gcc -c -g  lex.yy.c
gcc -c -g  y.tab.c
gcc -c -g  hash.c
gcc -c -g  ast.c
gcc -c -g  semantic.c
gcc -c -g tacs.c
gcc -c -g  asm.c
gcc -c -g commons.c
gcc main.o lex.yy.o y.tab.o hash.o ast.o semantic.o tacs.o asm.o commons.o
-g -o etapa6


Semantics start

Undeclared semantic errors: 0

ast(AST_PROGRAM) {
 ast(AST_DECLARATION_LIST) {
   ast(AST_DECLARATION_GLOBAL_INT, in__dash__repeat) {
     ast(AST_SYMBOL, 0) {
   ast(AST_DECLARATION_LIST) {
     ast(AST_DECLARATION_GLOBAL_CHAR, first__dash__char) {
       ast(AST_SYMBOL, 'a') {
     ast(AST_DECLARATION_LIST) {
       ast(AST_DECLARATION_GLOBAL_INT, result) {
         ast(AST_SYMBOL, 0) {
       ast(AST_DECLARATION_LIST) {
         ast(AST_DECLARATION_GLOBAL_INT, n) {
           ast(AST_SYMBOL, 0) {
         ast(AST_DECLARATION_LIST) {
           ast(AST_DECLARATION_GLOBAL_FLOAT, fa) {
             ast(AST_SYMBOL, 2) {
             ast(AST_SYMBOL, 3) {
           ast(AST_DECLARATION_LIST) {
             ast(AST_DECLARATION_GLOBAL_FLOAT, fb) {
```

```
                ast(AST_SYMBOL, 5) {
                ast(AST_SYMBOL, 6) {
              ast(AST_DECLARATION_LIST) {
                ast(AST_DECLARATION_GLOBAL_INT, count) {
                  ast(AST_SYMBOL, 0) {
                ast(AST_DECLARATION_LIST) {
                  ast(AST_DECLARATION_FUNCTION_INT, repeatchar) {
                    ast(AST_DECLARATION_FUNCTION_ARGS_INT, repeat) {
                      ast(AST_DECLARATION_FUNCTION_ARGS_CHAR, caracter) {
                        ast(AST_DECLARATION_FUNCTION_ARGS_FLOAT, argsfa) {
                          ast(AST_DECLARATION_FUNCTION_ARGS_FLOAT, argsfb) {
                    ast(AST_DECLARATION_FUNCTION_BODY) {
                      ast(AST_COMMAND_BLOCK) {
                        ast(AST_COMMAND_LIST) {
                          ast(AST_PRINT) {
                            ast(AST_PRINT_STRING) {
                              ast(AST_SYMBOL, As proximas ) {
                              ast(AST_PRINT_EXP) {
                                ast(AST_SYMBOL, repeat) {
                                ast(AST_PRINT_STRING) {
                                  ast(AST_SYMBOL,  letras do alfabeto depois
de [) {
                                  ast(AST_PRINT_EXP) {
                                    ast(AST_SYMBOL, caracter) {
                                    ast(AST_PRINT_STRING) {
                                      ast(AST_SYMBOL, ] são: \n) {
                          ast(AST_COMMAND_LIST) {
                            ast(AST_FLUX_CONTROLL_WHILE) {
                              ast(AST_EXPRESSION_BLOCK) {
                                ast(AST_LT, <) {
                                  ast(AST_SYMBOL, count) {
                                  ast(AST_SYMBOL, repeat) {
                                ast(AST_COMMAND_BLOCK) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_PRINT) {
                                      ast(AST_PRINT_EXP) {
                                        ast(AST_SYMBOL, caracter) {
                                        ast(AST_PRINT_STRING) {
                                          ast(AST_SYMBOL, , ) {
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_ATTRIBUITION, count) {
                                        ast(AST_ADD, +) {
                                          ast(AST_SYMBOL, count) {
                                          ast(AST_SYMBOL, 1) {
```

```
                                    ast(AST_COMMAND_LIST) {
                                      ast(AST_ATTRIBUITION, caracter) {
                                        ast(AST_ADD, +) {
                                          ast(AST_SYMBOL, caracter) {
                                          ast(AST_SYMBOL, 1) {
                              ast(AST_COMMAND_LIST) {
                                ast(AST_PRINT) {
                                  ast(AST_PRINT_STRING) {
                                    ast(AST_SYMBOL, \n\n O terceiro argumento
é um float com valor: ) {
                                    ast(AST_PRINT_EXP) {
                                      ast(AST_SYMBOL, argsfa) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_PRINT) {
                                    ast(AST_PRINT_STRING) {
                                      ast(AST_SYMBOL, \n\n O quarto argumento
é um float com valor: ) {
                                    ast(AST_PRINT_EXP) {
                                      ast(AST_SYMBOL, argsfb) {
                                      ast(AST_PRINT_STRING) {
                                        ast(AST_SYMBOL, \n) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_RETURN) {
                                    ast(AST_SYMBOL, result) {
                  ast(AST_DECLARATION_LIST) {
                    ast(AST_DECLARATION_FUNCTION_INT, main) {
                      ast(AST_DECLARATION_FUNCTION_BODY) {
                        ast(AST_COMMAND_BLOCK) {
                          ast(AST_COMMAND_LIST) {
                            ast(AST_PRINT) {
                              ast(AST_PRINT_STRING) {
                                ast(AST_SYMBOL, Teste 4:) {
                            ast(AST_COMMAND_LIST) {
                              ast(AST_PRINT) {
                                ast(AST_PRINT_STRING) {
                                  ast(AST_SYMBOL, tipos de dados e chamadas
com multiplos argumentos [int, char, float] \n) {
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_PRINT) {
                                    ast(AST_PRINT_STRING) {
                                      ast(AST_SYMBOL, \n) {
                                      ast(AST_PRINT_STRING) {
                                        ast(AST_SYMBOL, Digite o numero de
letras: ) {
```

```
                                ast(AST_COMMAND_LIST) {
                                  ast(AST_ATTRIBUITION, in__dash__repeat) {
                                    ast(AST_READ) {
                                  ast(AST_COMMAND_LIST) {
                                    ast(AST_RETURN) {
                                      ast(AST_FUNCTION_CALL, repeatchar) {
                                        ast(AST_FUNCTION_CALL_ARGS) {
                                          ast(AST_SYMBOL, in__dash__repeat)
{
                                          ast(AST_FUNCTION_CALL_ARGS) {
                                            ast(AST_SYMBOL,
first__dash__char) {
                                            ast(AST_FUNCTION_CALL_ARGS) {
                                              ast(AST_SYMBOL, fa) {
                                              ast(AST_FUNCTION_CALL_ARGS) {
                                                ast(AST_SYMBOL, fb) {

hash after tacGenerateCode

Table[14]        has first__dash__char           type 258        datatype 2
datavalue 97    datastring 'a'
Table[15]        has Teste 4:            type 264        datatype 5
datavalue 0     datastring (null)
Table[21]        has fa          type 258        datatype 3      datavalue
2     datastring 3
Table[23]        has main        type 259        datatype 1      datavalue
0     datastring (null)
Table[24]        has _TMP_VAR_11         type 1607       datatype 5
datavalue 0     datastring tipos de dados e chamadas com multiplos
argumentos [int, char, float] \n
Table[36]        has _TMP_VAR_4          type 1607       datatype 5
datavalue 0     datastring ,
Table[43]        has +           type 269        datatype 0      datavalue
0     datastring (null)
Table[47]        has /           type 272        datatype 0      datavalue
0     datastring (null)
Table[48]        has 0           type 262        datatype 1      datavalue
0     datastring (null)
Table[49]        has 1           type 262        datatype 1      datavalue
1     datastring (null)
Table[50]        has _TMP_LABEL_1        type 265        datatype 0
datavalue 0     datastring (null)
Table[50]        has 2           type 262        datatype 1      datavalue
2     datastring (null)
```

```
Table[51]       has 3          type 262       datatype 1      datavalue
3     datastring (null)
Table[53]       has 5          type 262       datatype 1      datavalue
5     datastring (null)
Table[54]       has 6          type 262       datatype 1      datavalue
6     datastring (null)
Table[60]       has <          type 274       datatype 0      datavalue
0     datastring (null)
Table[87]       has _TMP_VAR_15         type 1607       datatype 0
datavalue 0     datastring (null)
Table[101]      has \n\n O quarto argumento é um float com valor:
type 264        datatype 5      datavalue 0     datastring (null)
Table[110]      has n          type 258       datatype 1      datavalue
0     datastring (null)
Table[112]      has _TMP_VAR_3          type 1607       datatype 0
datavalue 0     datastring (null)
Table[124]      has fb         type 258       datatype 3      datavalue
5     datastring 6
Table[150]      has _TMP_VAR_19         type 1607       datatype 0
datavalue 0     datastring (null)
Table[154]      has count      type 258       datatype 1      datavalue
0     datastring (null)
Table[188]      has _TMP_VAR_2          type 1607       datatype 5
datavalue 0     datastring As proximas
Table[188]      has tipos de dados e chamadas com multiplos argumentos
[int, char, float] \n          type 264       datatype 5      datavalue
0     datastring (null)
Table[213]      has repeat     type 260       datatype 1      datavalue
0     datastring (null)
Table[260]      has \n         type 264       datatype 5      datavalue
0     datastring (null)
Table[264]      has _TMP_VAR_1          type 1607       datatype 5
datavalue 0     datastring  letras do alfabeto depois de [
Table[289]      has _TMP_VAR_12         type 1607       datatype 5
datavalue 0     datastring Digite o numero de letras:
Table[310]      has  letras do alfabeto depois de [            type 264
datatype 5      datavalue 0     datastring (null)
Table[324]      has repeatchar         type 259       datatype 1
datavalue 0     datastring (null)
Table[340]      has _TMP_VAR_0          type 1607       datatype 5
datavalue 0     datastring ] são: \n
Table[352]      has _TMP_VAR_16         type 1607       datatype 0
datavalue 0     datastring (null)
Table[443]      has ,          type 264       datatype 5      datavalue
```

```
0     datastring (null)
Table[461]       has in__dash__repeat            type 258        datatype 1
datavalue 0     datastring (null)
Table[554]       has _TMP_VAR_13         type 1607       datatype 5
datavalue 0     datastring \n
Table[578]       has _TMP_LABEL_0        type 265        datatype 0
datavalue 0     datastring (null)
Table[617]       has _TMP_VAR_17         type 1607       datatype 0
datavalue 0     datastring (null)
Table[623]       has Digite o numero de letras:          type 264
datatype 5      datavalue 0     datastring (null)
Table[653]       has _TMP_VAR_9          type 1607       datatype 5
datavalue 0     datastring \n\n O quarto argumento é um float com valor:
Table[667]       has \n\n O terceiro argumento é um float com valor:
type 264        datatype 5      datavalue 0     datastring (null)
Table[729]       has _TMP_VAR_8          type 1607       datatype 5
datavalue 0     datastring \n
Table[748]       has result      type 258        datatype 1      datavalue
0     datastring (null)
Table[756]       has _TMP_VAR_10         type 1607       datatype 5
datavalue 0     datastring Teste 4:
Table[805]       has _TMP_VAR_7          type 1607       datatype 5
datavalue 0     datastring \n\n O terceiro argumento é um float com valor:
Table[812]       has 'a'         type 263        datatype 2      datavalue
0     datastring (null)
Table[819]       has _TMP_VAR_14         type 1607       datatype 0
datavalue 0     datastring (null)
Table[839]       has argsfb      type 260        datatype 3      datavalue
0     datastring
Table[876]       has caracter            type 260        datatype 2
datavalue 114   datastring argsfa
Table[881]       has _TMP_VAR_6          type 1607       datatype 0
datavalue 0     datastring (null)
Table[882]       has _TMP_VAR_18         type 1607       datatype 0
datavalue 0     datastring (null)
Table[915]       has ] são: \n           type 264        datatype 5
datavalue 0     datastring (null)
Table[922]       has argsfa      type 260        datatype 3      datavalue
0     datastring
Table[957]       has _TMP_VAR_5          type 1607       datatype 0
datavalue 0     datastring (null)
Table[957]       has As proximas         type 264        datatype 5
datavalue 0     datastring (null)
```

```
TAC(TAC_DEC_GLOBAL_INT, in__dash__repeat, 0, 0);
TAC(TAC_DEC_GLOBAL_CHAR, first__dash__char, 'a', 0);
TAC(TAC_DEC_GLOBAL_INT, result, 0, 0);
TAC(TAC_DEC_GLOBAL_INT, n, 0, 0);
TAC(TAC_DEC_GLOBAL_FLOAT, fa, 2, 3);
TAC(TAC_DEC_GLOBAL_FLOAT, fb, 5, 6);
TAC(TAC_DEC_GLOBAL_INT, count, 0, 0);
TAC(TAC_UNKNOWN, argsfb, 0, 0);
TAC(TAC_UNKNOWN, argsfa, argsfb, 0);
TAC(TAC_DEC_FUNC_ARGS, caracter, argsfa, 0);
TAC(TAC_DEC_FUNC_ARGS, repeat, caracter, 0);
TAC(TAC_BEGINFUN, repeatchar, result, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_2, 0, 0);
TAC(TAC_PRINT_INT, repeat, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_1, 0, 0);
TAC(TAC_PRINT_CHAR, caracter, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_0, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_0, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_0, 0, 0);
TAC(TAC_LT, _TMP_VAR_3, count, repeat);
TAC(TAC_JMPZ, _TMP_LABEL_1, _TMP_VAR_3, 0);
TAC(TAC_PRINT_CHAR, caracter, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_4, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_4, 0, 0);
TAC(TAC_ADD, _TMP_VAR_5, count, 1);
TAC(TAC_MOVE, count, _TMP_VAR_5, 0);
TAC(TAC_ADD, _TMP_VAR_6, caracter, 1);
TAC(TAC_MOVE, caracter, _TMP_VAR_6, 0);
TAC(TAC_JMP, _TMP_LABEL_0, 0, 0);
TAC(TAC_LABEL, _TMP_LABEL_1, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_7, 0, 0);
TAC(TAC_UNKNOWN, argsfa, 0, 0);
TAC(TAC_PRINT, argsfa, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_9, 0, 0);
TAC(TAC_UNKNOWN, argsfb, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_8, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_8, 0, 0);
TAC(TAC_RETURN, result, 0, 0);
TAC(TAC_ENDFUN, repeatchar, repeat, 0);
TAC(TAC_BEGINFUN, main, _TMP_VAR_19, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_10, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_10, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_11, 0, 0);
```

```
TAC(TAC_PRINT, _TMP_VAR_11, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_13, 0, 0);
TAC(TAC_PRINT_STRING, _TMP_VAR_12, 0, 0);
TAC(TAC_PRINT, _TMP_VAR_12, 0, 0);
TAC(TAC_READ, _TMP_VAR_14, 0, 0);
TAC(TAC_MOVE, in__dash__repeat, _TMP_VAR_14, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_15, fb, 0);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_16, fa, _TMP_VAR_15);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_17, first__dash__char, _TMP_VAR_16);
TAC(TAC_FUNC_CALL_ARGS, _TMP_VAR_18, in__dash__repeat, _TMP_VAR_17);
TAC(TAC_FUN_CALL, _TMP_VAR_19, repeatchar, 0);
TAC(TAC_RETURN, _TMP_VAR_19, 0, 0);
TAC(TAC_ENDFUN, main, 0, 0);
Numero de linhas: 40.
Compilation Success.
Teste 4:tipos de dados e chamadas com multiplos argumentos [int, char,
float]

Digite o numero de letras: 21
As proximas 21 letras do alfabeto depois de [a] são:
a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u,

O terceiro argumento é um float com valor: 2/3

O quarto argumento é um float com valor: 5/6
```