|  |  |
| --- | --- |
| .data |  |
|  | nul: .byte 0x00 |
|  | endOfText: .byte 0x03 |
|  | space: .byte 0x20 |
|  | newLine: .byte 0x0A |
|  | plus: .byte 0x2B |
|  | minus: .byte 0x2D |
|  | inbuf: .asciz "################################################################" |
|  | outbuf: .asciz "################################################################" |
|  | InPos: .quad -1 # -1 = empty |
|  | OutPos: .quad 0 |
|  | counter: .quad 0 |
|  | MAXPOS: .quad 0 |
|  |  |
|  |  |
|  | .text |
|  |  |
|  | .global getchar |
|  | .global putText |
|  | .global setOutPos |
|  | .global getOutPos |
|  | .global putInt |
|  | .global outImage |
|  | .global getInt |
|  | .global inImage |
|  | .global getInPos |
|  | .global setInPos |
|  | .global putChar |
|  | .global getint |
|  | .global getText |
|  |  |
|  | # ------------------ INMATNING ------------------ |
|  | # inImage |
|  | inImage:movq $inbuf, %rdi # input gets stored in inbuf |
|  | movq $64,%rsi # max 63 chars |
|  | movq stdin, %rdx |
|  | call fgets # call for input |
|  | leaq inbuf, %rdi # load adress of inbuf to get the acctual "lenght" of the input |
|  | xor %r10, %r10 # counter input for "length" |
|  | getLen:incq %r10 # Add 1 to "length" |
|  | incq %rdi |
|  | movb (%rdi), %r11b |
|  | cmpb %r11b, nul # Nul terminated "string", so if nul we are at the end |
|  | jne getLen |
|  | decq %r10 # We dont care about nul |
|  | movq %r10, MAXPOS # Store the "length" in in MAXPOS (max position) for the input |
|  | call setInPosZero |
|  | ret |
|  | # getInt |
|  | getInt:call checkInPos # Are we at the end of the buffer? |
|  | leaq inbuf, %rdi |
|  | addq InPos, %rdi |
|  | xor %rdx, %rdx |
|  | xor %r11, %r11 |
|  | xor %rsi, %rsi # 0 = positive, 1 = negative |
|  | jmp readSign |
|  | noNumber:incq %rdi |
|  | incq InPos |
|  | readSign:movb (%rdi), %dl |
|  | cmpb $'-', %dl # is it negative? |
|  | je negativeInt |
|  | cmpb $'+', %dl # is it positive? |
|  | je buildIntBegin |
|  | cmpb $' ', %dl # no number? |
|  | je noNumber |
|  | jmp buildIntLoop # just a number means we can skip some routines |
|  | negativeInt:movq $1, %rsi # "dirty bit" to remember if it is a negative number |
|  | buildIntBegin: incq %rdi # if we arent looking at a number we move to the number |
|  | incq InPos |
|  | buildIntLoop:movb (%rdi), %dl |
|  | cmpb $'0', %dl |
|  | jl buildIntEnd # no number means we are done |
|  | cmpb $'9', %dl |
|  | jg buildIntEnd # no number means we are done |
|  | imulq $10, %r11 |
|  | subq $48, %rdx # ASCII to int |
|  | addq %rdx, %r11 |
|  | incq %rdi # move to next position |
|  | incq InPos # because above we need to increase |
|  | jmp buildIntLoop # continue loop |
|  | makeNegativeInt:negq %r11 # make r11 negative |
|  | movq $0, %rsi |
|  | buildIntEnd:cmpq $1, %rsi |
|  | je makeNegativeInt # if it is not negative we can just return |
|  | movq %r11, %rax |
|  | ret |
|  | # getText |
|  | getText:call checkInPos |
|  | leaq inbuf, %r11 |
|  | addq InPos, %r11 # Move to the current adress of the buffer |
|  | xor %rax, %rax # counter |
|  | getTextLoop:movb (%r11), %r10b |
|  | cmpb nul, %r10b # Are we at the end? |
|  | je getTextEnd |
|  | cmpq %rax, %rsi # Are we allowed to read more chars? |
|  | jz getTextEnd |
|  | movb %r10b, (%rdi) |
|  | incq %r11 |
|  | incq InPos |
|  | incq %rdi |
|  | incq %rax # Add to counter |
|  | jmp getTextLoop |
|  | getTextEnd:incq %rdi |
|  | movb nul, %r10b # Make into nul terminated "string" |
|  | movb %r10b, (%rdi) |
|  | ret |
|  | # getChar |
|  | getChar:call checkInPos |
|  | leaq inbuf, %rdi |
|  | addq InPos, %rdi |
|  | movq (%rdi), %rax |
|  | incq InPos |
|  | ret |
|  | getInPos:movq InPos, %rax |
|  | ret |
|  | # setInPos |
|  | setInPos:cmpq $0, %rdi |
|  | jle setInPosZero |
|  | cmpq MAXPOS, %rdi |
|  | jge setInPosMAX |
|  | movq %rdi, InPos |
|  | ret |
|  | setInPosMAX:movq MAXPOS, %r10 |
|  | movq %r10, InPos |
|  | ret |
|  | setInPosZero:movq $0, InPos |
|  | ret |
|  | checkInPosCall:call inImage |
|  | checkInPos:cmpq $-1, InPos |
|  | jz checkInPosCall |
|  | movq MAXPOS, %r10 |
|  | cmpq %r10, InPos |
|  | jz checkInPosCall |
|  | ret |
|  | # ------------------ UTMATNING ------------------ |
|  | # outImage |
|  | outImage: leaq outbuf, %rdi |
|  | xor %eax, %eax |
|  | call outputFormating |
|  | call printf |
|  | movq $0, OutPos |
|  | movq OutPos, %rdi |
|  | addq OutPos, %rdi |
|  | ret |
|  | outputFormating: pushq %rdi # saving adress of outbuf to stack |
|  | movb nul, %r10b |
|  | addq OutPos, %rdi # moving to the end of the output |
|  | movb %r10b, (%rdi) # making end of the output to nul character |
|  | popq %rdi # making %rdi start of outbuf again |
|  | ret |
|  | # putInt |
|  | putInt:xor %r11, %r11 |
|  | xor %rcx, %rcx |
|  | xor %rax, %rax |
|  | xor %rdx, %rdx |
|  | xor %r9, %r9 # "dirty bit", keeps track of if the int is negative. = 0 if positve |
|  | movq $0, %r11 # acts as a counter |
|  | movq $10, %rcx |
|  | movq %rdi, %rax # moving 1 arg to be devided |
|  | leaq outbuf, %r10 # loading the adress of outbuf in r10 |
|  | addq OutPos, %r10 # move the adress to the correct one |
|  | positiveOrNegative:cmpq $0, %rdi |
|  | jge putIntStackLoop # if int is positive it jumps to make the rutine as "normal" |
|  | incq %r9 # the int is negative |
|  | negq %rdi # make the int positive |
|  | xor %rax, %rax |
|  | movq %rdi, %rax # moving 1 arg to be devided |
|  | putIntStackLoop:xor %rdx, %rdx |
|  | divq %rcx # %rax/%rcx, result in %rax and the remainder in %rdx |
|  | pushq %rdx # pushes the remainder to stack |
|  | incq %r11 # keeping track of the lenght of the int |
|  | cmpq $0, %rax # if rax is 0 all the didits are pushed on the stack |
|  | je makeNegative # no more didgits means we can rebuild the int into outbuf |
|  | jmp putIntStackLoop # still didgits in the int |
|  | makeNegative:cmpq $0, %r9 |
|  | je addToOutbufLoop # if positive we can skip this step |
|  | movb $'-', (%r10) |
|  | incq %r10 |
|  | incq OutPos |
|  | addToOutbufLoop:xor %rdx, %rdx # rdx = 0 |
|  | cmpq $0, %r11 # are all the didigits loaded in to outpuf? |
|  | jz addToOutbufEnd # ret |
|  | popq %rdx # load the didgit into rdx. from last -> first |
|  | addb $48, %dl # int -> char |
|  | movb %dl, (%r10) # put the char in outbuf |
|  | incq %r10 # move the outbuf pointer to the element before it |
|  | incq OutPos |
|  | decq %r11 # one less didgit has to be added |
|  | jmp addToOutbufLoop |
|  | addToOutbufEnd:ret |
|  | # putText |
|  | putText: leaq outbuf, %r9 |
|  | addq OutPos, %r9 |
|  | movb (%rdi), %dl |
|  | putTextLoop: movb %dl, (%r9) |
|  | incq %rdi |
|  | incq %r9 |
|  | incq OutPos |
|  | call checkOutPos |
|  | movb (%rdi), %dl |
|  | cmpb %dl, nul |
|  | jne putTextLoop |
|  | ret |
|  | # putChar |
|  | putChar:call checkOutPos |
|  | leaq outbuf, %rdx |
|  | addq OutPos, %rdx |
|  | movq %rdi, (%rdx) |
|  | incq OutPos |
|  | ret |
|  | # getOutPos |
|  | getOutPos:movq OutPos, %rax |
|  | ret |
|  | # setOutPos |
|  | setOutPos:cmpq $0, %rdi |
|  | jle setOutPosZero |
|  | cmpq $63, %rdi |
|  | jge setOutPosMAX |
|  | movq %rdi, OutPos |
|  | ret |
|  | setOutPosMAX:movq $63, OutPos |
|  | ret |
|  | setOutPosZero: |
|  | movq $0, OutPos |
|  | ret |
|  | checkOutPosCall: |
|  | call outImage |
|  | checkOutPos: |
|  | cmpq $63, OutPos |
|  | je checkOutPosCall |
|  | ret |
|  | # ------------------ GLOBAL ------------------ |
|  | justRet:ret |

Compile: gcc -no-pie -g Mprov64.s moduls.s