HANOI UNIVERSITY OF SCIENCE & TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FINAL PROJECT REPORT

IT3280E – ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE LAB

Course information

Course ID	Course title	Class ID
IT3280E	Assembly language and Computer architecture Lab	143684

Student information

Student's full name	Class	Student ID
Le Xuan Hieu	ICT 02 K66	20215201
Dinh Viet Quang	ICT 02 K66	20215235

Instructor: MSc. Le Ba Vui

Teaching Assistant: Do Gia Huy

Hanoi, January 2024

Table of Contents

I. Task 5: Infix and postfix expressions

1. Problem description

Create a program that can calculate an expression by evaluating the postfix expression. Requirements:

- Enter an infix expression from the console, for example: 9 + 2 + 8 * 6
- Print it in the postfix representation, for example: 9 2 + 8 6 * +
- Calculate and display the result to the console screen.

The operand must be an integer between 0 and 99. Operators include addition, subtraction, multiplication, division (/), division with remainder (%) and parenthesis.

2. Project implementation

a. Algorithm

- *Exception handling*

- *Parentheses Matching*: The program tracks the number of opening and closing parentheses (‘**(**’ and ‘**)**’) encountered in the infix expression. If at any point there is a closing parenthesis without a corresponding opening parenthesis, or if there are unmatched opening parentheses at the end, the expression is considered invalid.
- *Consecutive Operators and Operands*: The code checks for consecutive operators or operands in the infix expression. If two consecutive operators are found, the expression is considered invalid. For example, 5 * * 3 is invalid. If two consecutive operands are found, the expression is also considered invalid. For example, 2 3 + 4 is invalid.
- *Ending with an Operator*: The code checks if the infix expression ends with an operator. If it does, the expression is considered invalid. For instance, 3 + 4 * is invalid.
- *Division by Zero:* The program includes a check to ensure that division by zero is avoided. If the operator is ‘/’, and the divisor is 0, the result is set to 0 by default.

- *Convert infix to postfix expression:* Thanks to the convenience and easy implementation, Stack is one of the common approaches to convert the Infix expression to Postfix expression.

Step 1: Put an infix expression into a character string which is called infix.

Step 2: Create a new string to store the postfix expression, called postfix, and initialize a stack.

Step 3: Consider the infix characters one by one and follow the steps until end of expression:

- If the character is a number, save it to postfix.

- If the character is “(”, put it into stack.

- If the character is “)”: pop all the elements out of stack and put it into postfix until meet “(” (also being popped out of the stack).

- If the character is an operator and if the stack is empty, push it into the operator stack.

- If the considering operator has a higher priority than the operator at the top of the operator stack, push that operator into the stack.

- If the considering operator has lower priority or equal to the operator at the top of the operator stack, then take the operator at the top of the stack and put it in postfix. The work continues until the priority of the top of the stack is smaller than the considering operator, then push the considering operator into the stack.

Step 4: Perform step 3 until the end of the expression and all operands and operators are placed in postfix. At the end of the infix string, if there are elements in the stack, pop those elements from the stack and put them in postfix. Consequently, we have the postfix expression.

- *Calculate the expression:*

Step 1: Loop the entire postfix expression from left to right.

Step 2: Create a new value stack.

Step 3: If the considering element is an operand, push it into the stack.

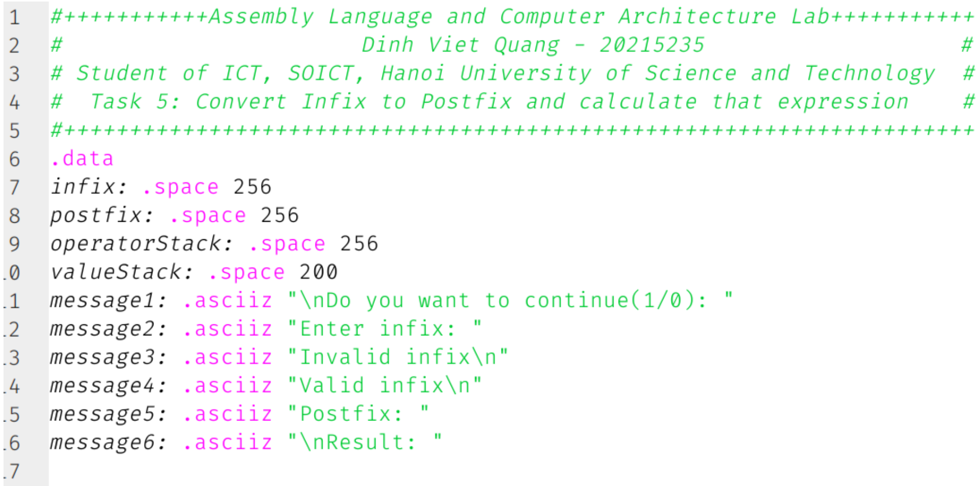
Step 4: If the considering element is an operator, pop the 2 operands from the top of stack, then check what the operation is and execute, the result is pushed back into the stack.

- *Code execution flow*
  - The program reads an infix expression from the user.
  - It checks the validity of the expression.
  - If the expression is valid, it converts it to a postfix expression and evaluates the result.
  - The result and the postfix expression are then printed.
  - The program prompts the user if they want to continue, and the loop repeats if the user chooses to continue.
- *Notes*
  - The code assumes that the input infix expression is space-separated and ends with a newline character.
  - The program handles basic arithmetic operators (+, -, *, /, %).
  - The conversion and evaluation are done using stacks.

b. Code explanation

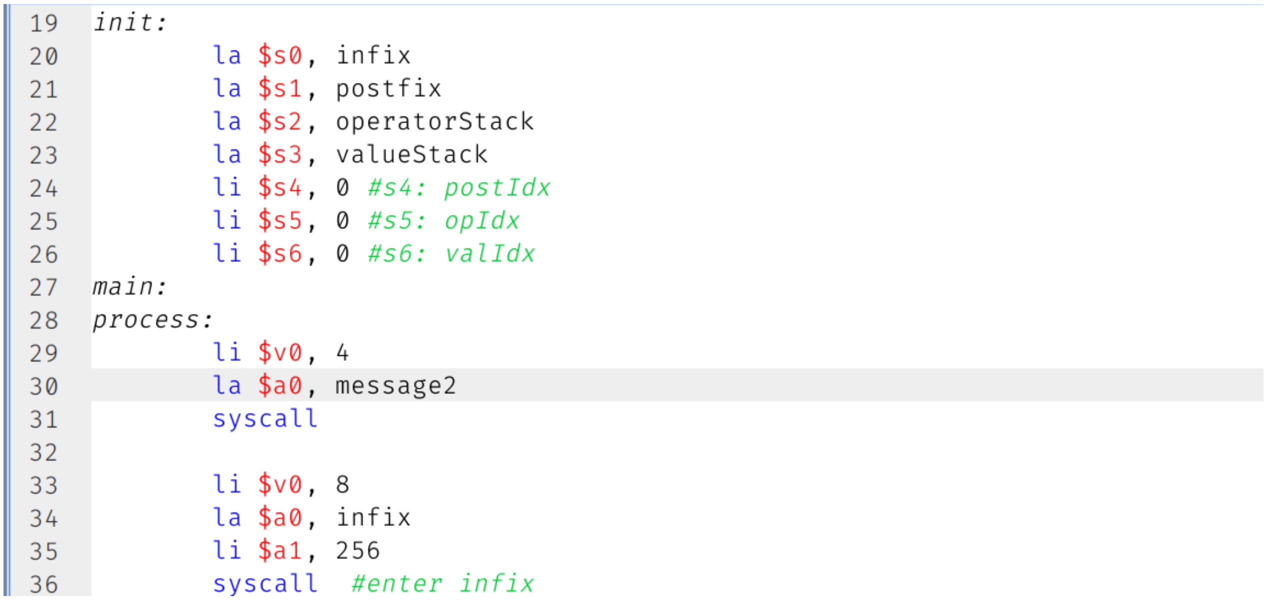
- Data section and Input section

- **infix, postfix, operatorStack, valueStack:** Arrays used to store the input infix expression, the resulting postfix expression, the operator stack, and the value stack, respectively.Infix to postfix conversion



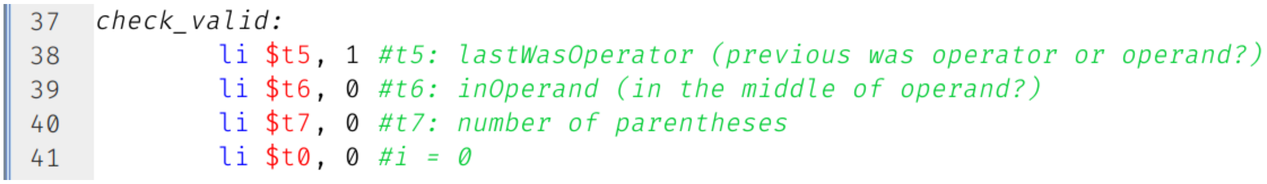
```
#+++++++++++Assembly Language and Computer Architecture Lab+++++++++++
#                      Dinh Viet Quang - 20215235                    #
# Student of ICT, SOICT, Hanoi University of Science and Technology  #
#  Task 5: Convert Infix to Postfix and calculate that expression    #
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.data
infix: .space 256
postfix: .space 256
operatorStack: .space 256
valueStack: .space 200
message1: .asciiz "\nDo you want to continue(1/0): "
message2: .asciiz "Enter infix: "
message3: .asciiz "Invalid infix\n"
message4: .asciiz "Valid infix\n"
message5: .asciiz "Postfix: "
message6: .asciiz "\nResult: "

```

- **process:** The main process for reading infix expressions, checking their validity, converting them to postfix, calculate the expression value, and printing the result**.**

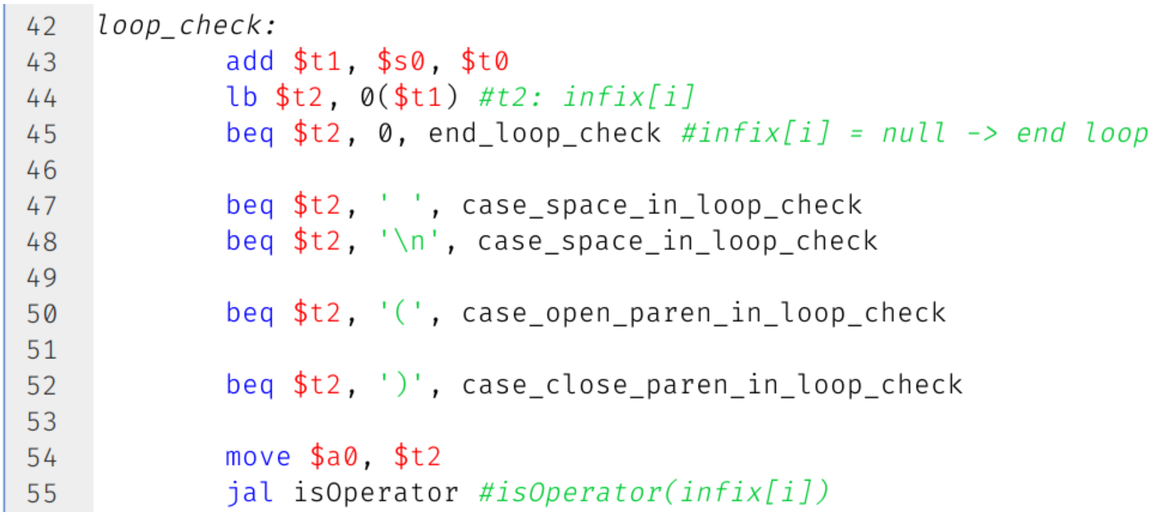


```
init:
        la $s0, infix
        la $s1, postfix
        la $s2, operatorStack
        la $s3, valueStack
        li $s4, 0 #s4: postIdx
        li $s5, 0 #s5: opIdx
        li $s6, 0 #s6: valIdx
main:
process:
        li $v0, 4
        la $a0, message2
        syscall

        li $v0, 8
        la $a0, infix
        li $a1, 256
        syscall  #enter infix
```

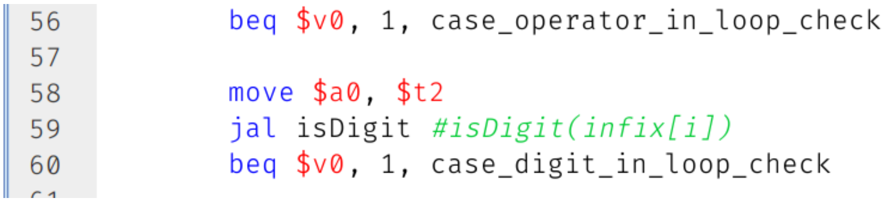
- **check_valid:** Checks the validity of the infix expression by using flags (‘lastWasOperator’, ‘inOperand’, ‘numParentheses’) and loops through the characters with the support of several helper section such as **isOperator** (Checks if a character is an operator) **isDigit** (Checks if a character is a digit).



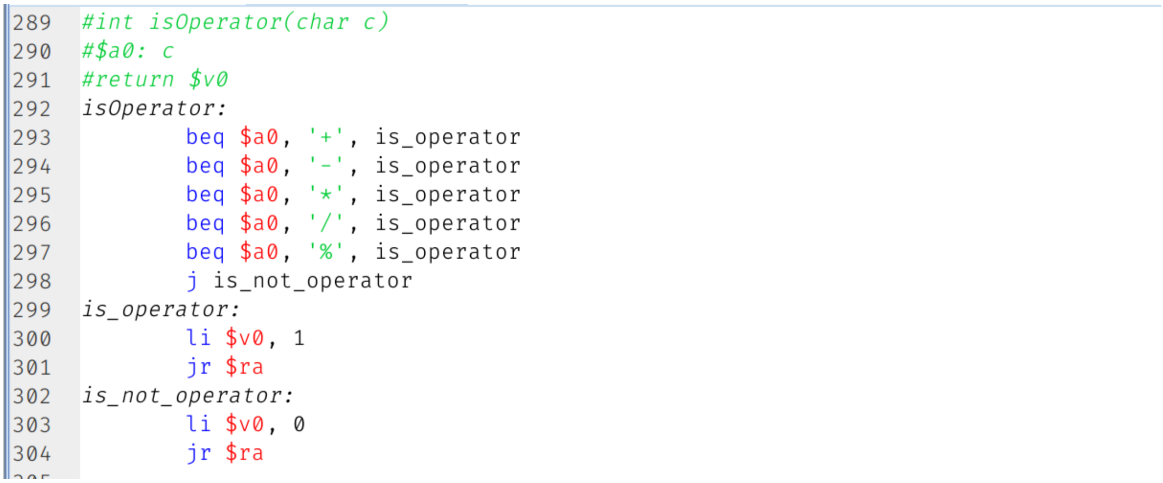
```
check_valid:
        li $t5, 1 #t5: lastWasOperator (previous was operator or operand?)
        li $t6, 0 #t6: inOperand (in the middle of operand?)
        li $t7, 0 #t7: number of parentheses
        li $t0, 0 #i = 0
```



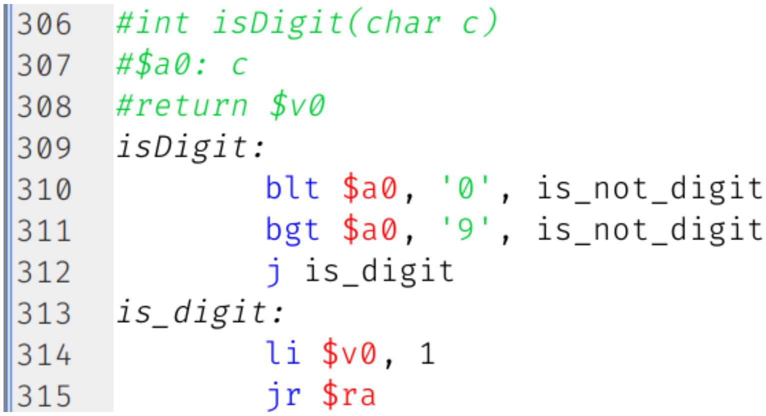
```
loop_check:
        add $t1, $s0, $t0
        lb $t2, 0($t1) #t2: infix[i]
        beq $t2, 0, end_loop_check #infix[i] = null -> end loop

        beq $t2, ' ', case_space_in_loop_check
        beq $t2, '\n', case_space_in_loop_check

        beq $t2, '(', case_open_paren_in_loop_check

        beq $t2, ')', case_close_paren_in_loop_check

        move $a0, $t2
        jal isOperator #isOperator(infix[i])
```



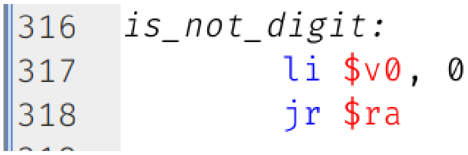
```
        beq $v0, 1, case_operator_in_loop_check

        move $a0, $t2
        jal isDigit #isDigit(infix[i])
        beq $v0, 1, case_digit_in_loop_check
```



```
#int isOperator(char c)
#$a0: c
#return $v0
isOperator:
        beq $a0, '+', is_operator
        beq $a0, '-', is_operator
        beq $a0, '*', is_operator
        beq $a0, '/', is_operator
        beq $a0, '%', is_operator
        j is_not_operator
is_operator:
        li $v0, 1
        jr $ra
is_not_operator:
        li $v0, 0
        jr $ra
```

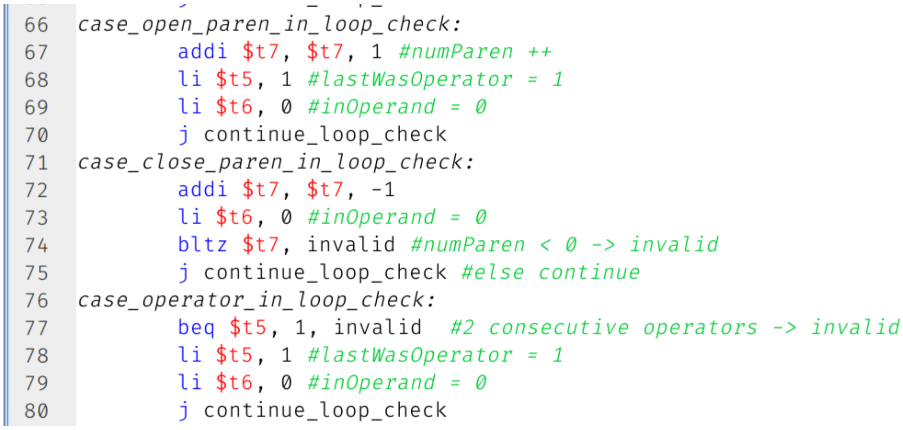


```
#int isDigit(char c)
#$a0: c
#return $v0
isDigit:
        blt $a0, '0', is_not_digit
        bgt $a0, '9', is_not_digit
        j is_digit
is_digit:
        li $v0, 1
        jr $ra
```

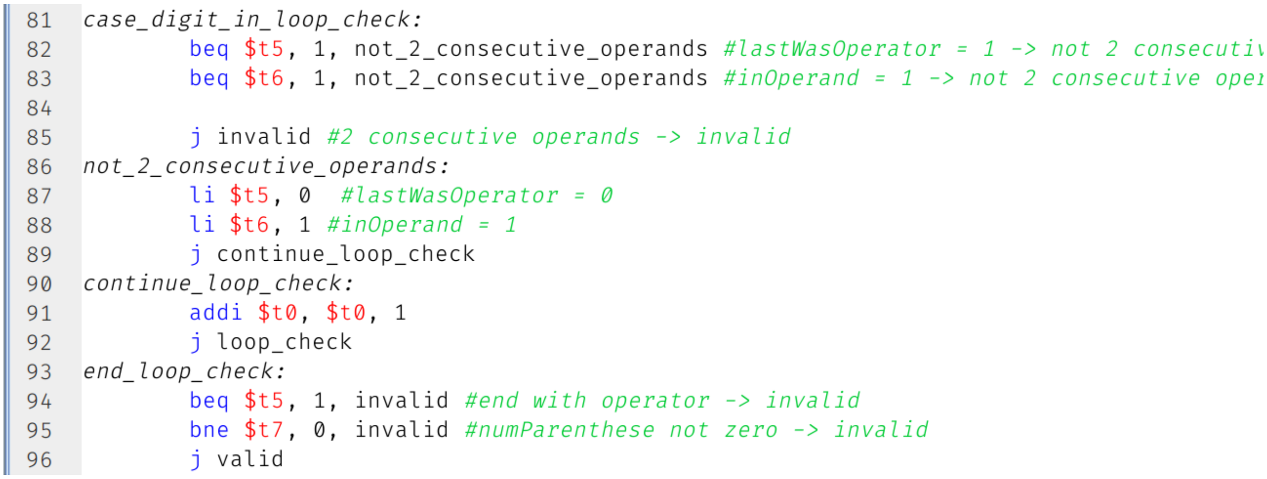


```
is_not_digit:
        li $v0, 0
        jr $ra
```

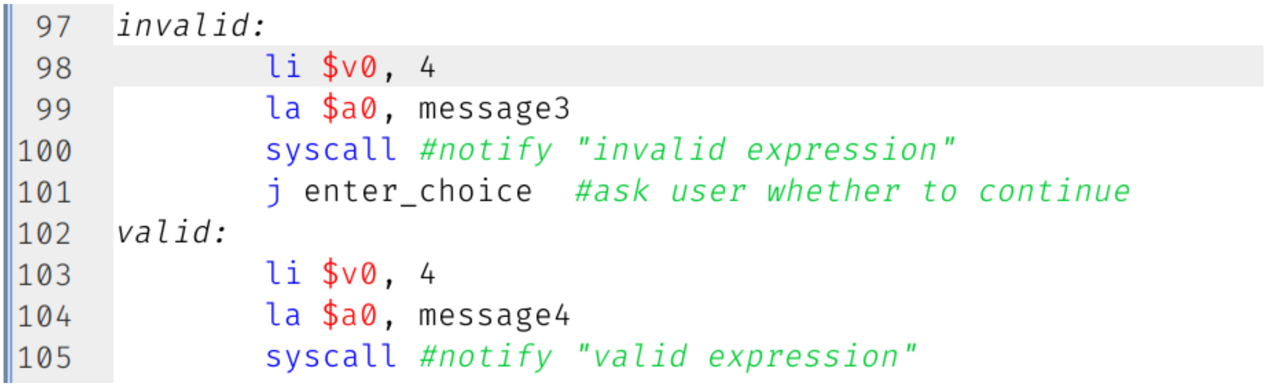
- Validation check includes:
    - **case_space_in_loop_check:** If the character is a space or newline, set **inOperand** to 0 and continue the loop
    - **case_open_paren_in_loop_check, case_close_paren_in_loop_check:** If the character is an open parenthesis '(', increment the count of parentheses and set flags accordingly. If the character is a close parenthesis ')', decrease the count of parentheses. If the count becomes negative, the expression is invalid.
    - **case_operator_in_loop_check, case_digit_in_loop_check, not_2_consecutive_operands:** If the character is an operator or digit, check for consecutive operators and digits. If found, the expression is invalid by updating register accordingly.



```
case_open_paren_in_loop_check:
        addi $t7, $t7, 1 #numParen ++
        li $t5, 1 #lastWasOperator = 1
        li $t6, 0 #inOperand = 0
        j continue_loop_check
case_close_paren_in_loop_check:
        addi $t7, $t7, -1
        li $t6, 0 #inOperand = 0
        bltz $t7, invalid #numParen < 0 -> invalid
        j continue_loop_check #else continue
case_operator_in_loop_check:
        beq $t5, 1, invalid  #2 consecutive operators -> invalid
        li $t5, 1 #lastWasOperator = 1
        li $t6, 0 #inOperand = 0
        j continue_loop_check
```



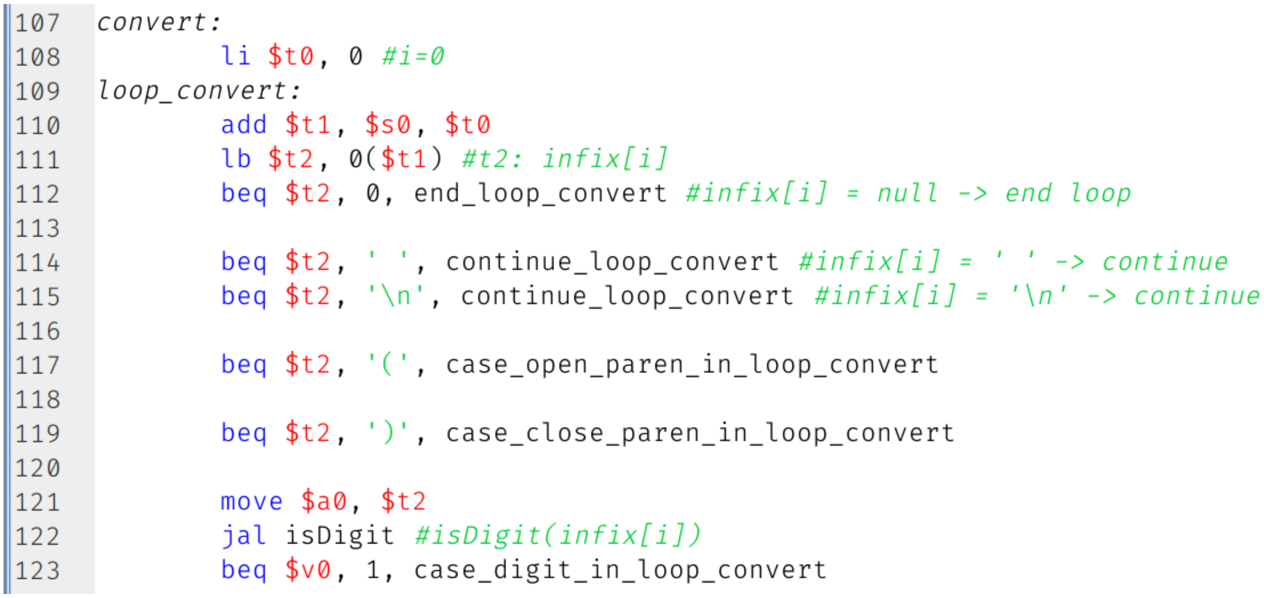
```
case_digit_in_loop_check:
        beq $t5, 1, not_2_consecutive_operands #lastWasOperator = 1 -> not 2 consecutiv
        beq $t6, 1, not_2_consecutive_operands #inOperand = 1 -> not 2 consecutive ope

        j invalid #2 consecutive operands -> invalid
not_2_consecutive_operands:
        li $t5, 0  #lastWasOperator = 0
        li $t6, 1 #inOperand = 1
        j continue_loop_check
continue_loop_check:
        addi $t0, $t0, 1
        j loop_check
end_loop_check:
        beq $t5, 1, invalid #end with operator -> invalid
        bne $t7, 0, invalid #numParenthese not zero -> invalid
        j valid
```



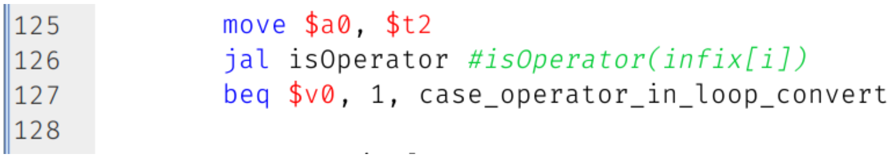
```
invalid:
        li $v0, 4
        la $a0, message3
        syscall #notify "invalid expression"
        j enter_choice  #ask user whether to continue
valid:
        li $v0, 4
        la $a0, message4
        syscall #notify "valid expression"
```

- Infix to Postfix expression conversion section and Calculate postfix expression.

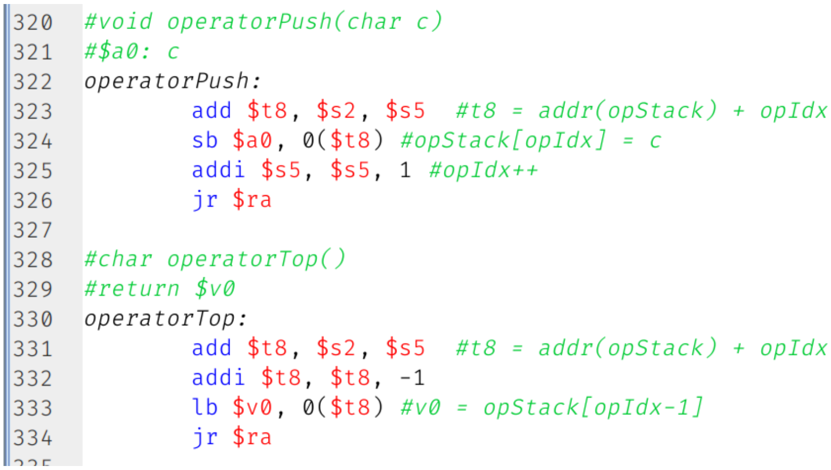
- **convert:** Converts the valid infix expression to postfix**.**
- Helper function:
    - **operatorPush:** Pushes an operator onto the operator stack.
    - **operatorTop:** Retrieves the top operator from the operator stack.
    - **postfixAppend:** Appends a character to the postfix expression.
    - **valuePush:** Pushes a value onto the value stack.
    - **valueTop:** Retrieves the top value from the value stack.
    - **calculate:** Performs the calculation based on the operator.



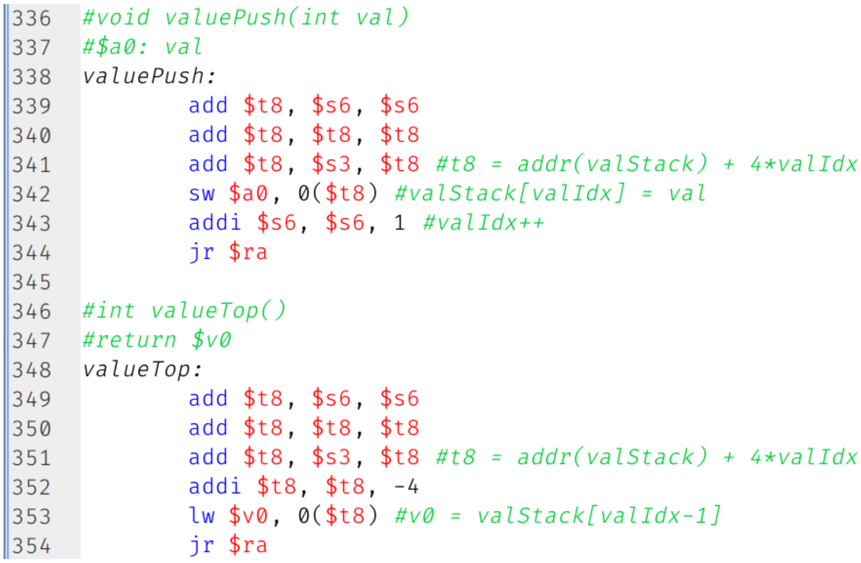
```
convert:
        li $t0, 0 #i=0
loop_convert:
        add $t1, $s0, $t0
        lb $t2, 0($t1) #t2: infix[i]
        beq $t2, 0, end_loop_convert #infix[i] = null -> end loop

        beq $t2, ' ', continue_loop_convert #infix[i] = ' ' -> continue
        beq $t2, '\n', continue_loop_convert #infix[i] = '\n' -> continue

        beq $t2, '(', case_open_paren_in_loop_convert

        beq $t2, ')', case_close_paren_in_loop_convert

        move $a0, $t2
        jal isDigit #isDigit(infix[i])
        beq $v0, 1, case_digit_in_loop_convert
```



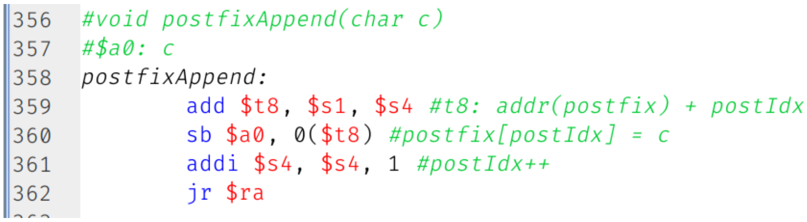
```
        move $a0, $t2
        jal isOperator #isOperator(infix[i])
        beq $v0, 1, case_operator_in_loop_convert

```



```
#void operatorPush(char c)
#$a0: c
operatorPush:
        add $t8, $s2, $s5  #t8 = addr(opStack) + opIdx
        sb $a0, 0($t8) #opStack[opIdx] = c
        addi $s5, $s5, 1 #opIdx++
        jr $ra

#char operatorTop()
#return $v0
operatorTop:
        add $t8, $s2, $s5  #t8 = addr(opStack) + opIdx
        addi $t8, $t8, -1
        lb $v0, 0($t8) #v0 = opStack[opIdx-1]
        jr $ra
```

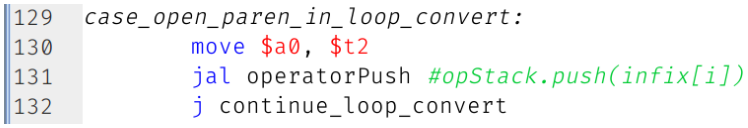


```
#void valuePush(int val)
#$a0: val
valuePush:
        add $t8, $s6, $s6
        add $t8, $t8, $t8
        add $t8, $s3, $t8 #t8 = addr(valStack) + 4*valIdx
        sw $a0, 0($t8) #valStack[valIdx] = val
        addi $s6, $s6, 1 #valIdx++
        jr $ra

#int valueTop()
#return $v0
valueTop:
        add $t8, $s6, $s6
        add $t8, $t8, $t8
        add $t8, $s3, $t8 #t8 = addr(valStack) + 4*valIdx
        addi $t8, $t8, -4
        lw $v0, 0($t8) #v0 = valStack[valIdx-1]
        jr $ra
```

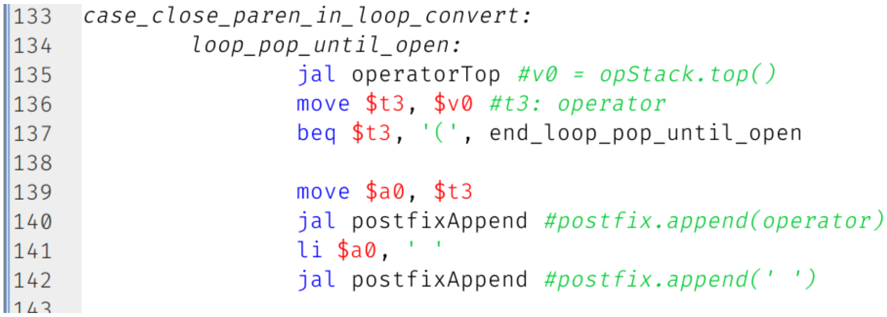


```
#void postfixAppend(char c)
#$a0: c
postfixAppend:
        add $t8, $s1, $s4 #t8: addr(postfix) + postIdx
        sb $a0, 0($t8) #postfix[postIdx] = c
        addi $s4, $s4, 1 #postIdx++
        jr $ra
```

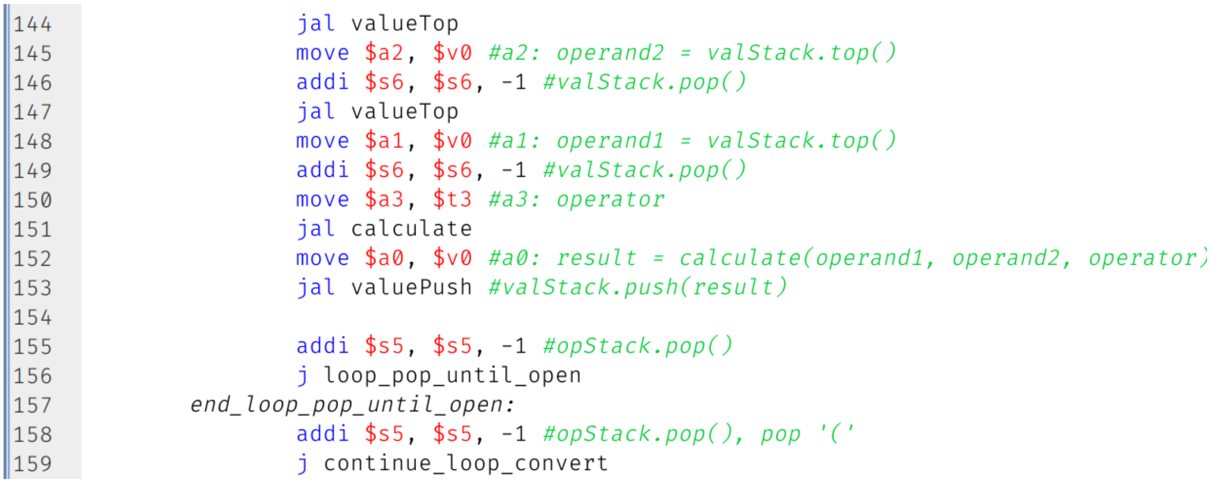
- **'case_open_paren_in_loop_convert':** handles the case where an open parenthesis is encountered in the infix expression. It pushes the open parenthesis onto the operator stack and continues processing the next character in the loop**.**
- **'case_close_paren_in_loop_convert':** handles the conversion of when a close parenthesis ')' is encountered. It pops operators from the operator stack and performs corresponding operations until an open parenthesis '(' is encountered, and then it continues processing the next character in the loop. This ensures proper handling of parentheses in the conversion process.



```
case_open_paren_in_loop_convert:
        move $a0, $t2
        jal operatorPush #opStack.push(infix[i])
        j continue_loop_convert
```

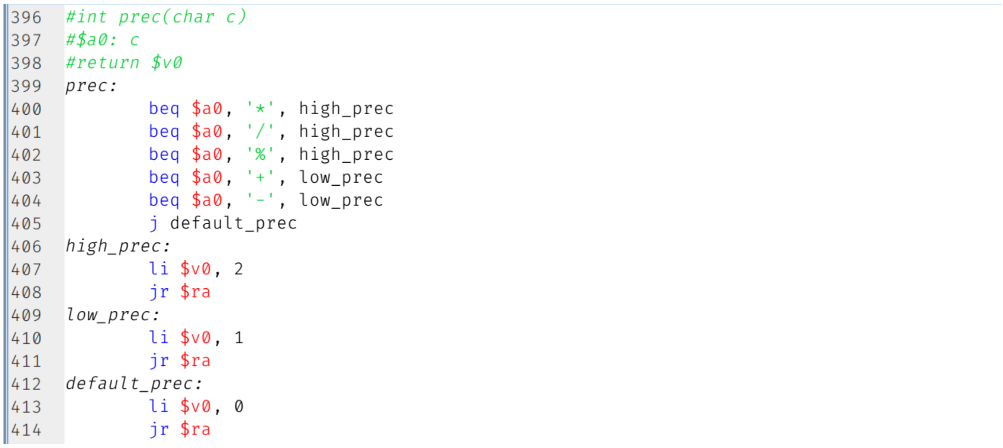


```
case_close_paren_in_loop_convert:
        loop_pop_until_open:
                jal operatorTop #v0 = opStack.top()
                move $t3, $v0 #t3: operator
                beq $t3, '(', end_loop_pop_until_open

                move $a0, $t3
                jal postfixAppend #postfix.append(operator)
                li $a0, ' '
                jal postfixAppend #postfix.append(' ')
```

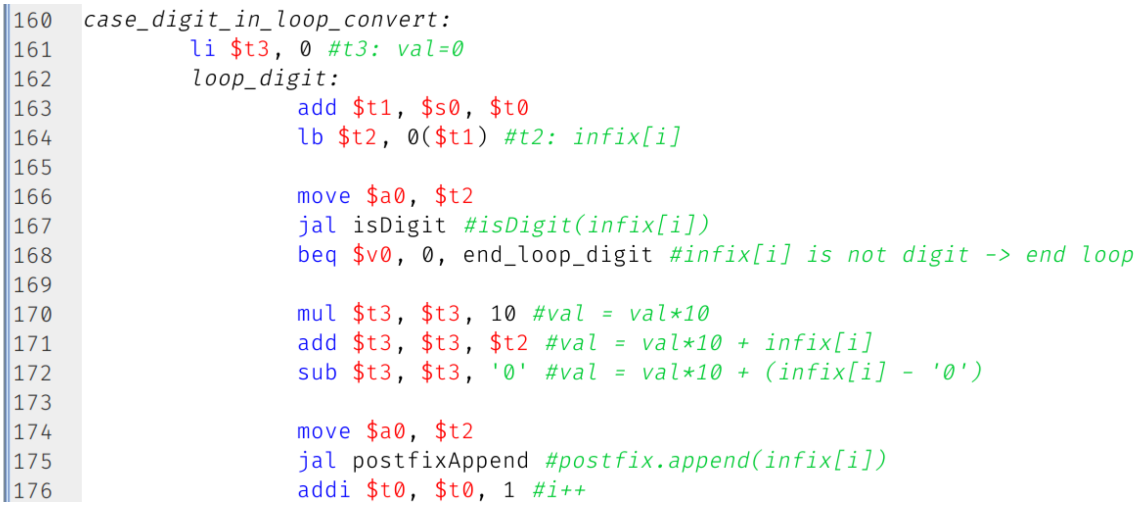


```
                jal valueTop
                move $a2, $v0 #a2: operand2 = valStack.top()
                addi $s6, $s6, -1 #valStack.pop()
                jal valueTop
                move $a1, $v0 #a1: operand1 = valStack.top()
                addi $s6, $s6, -1 #valStack.pop()
                move $a3, $t3 #a3: operator
                jal calculate
                move $a0, $v0 #a0: result = calculate(operand1, operand2, operator)
                jal valuePush #valStack.push(result)

                addi $s5, $s5, -1 #opStack.pop()
                j loop_pop_until_open
        end_loop_pop_until_open:
                addi $s5, $s5, -1 #opStack.pop(), pop '('
                j continue_loop_convert
```

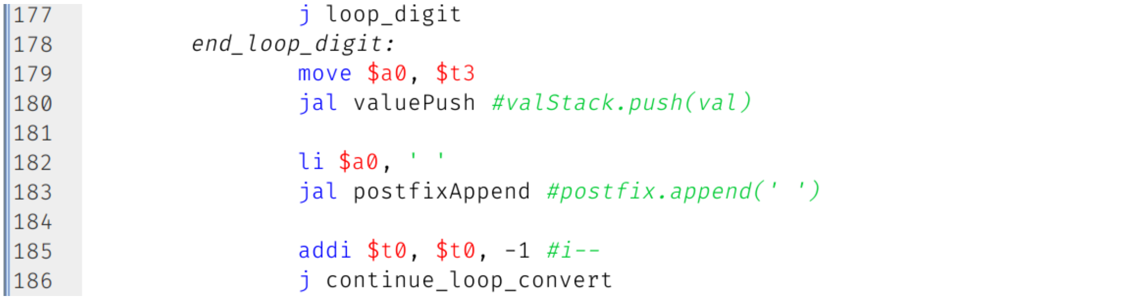
- ‘**case_digit_in_loop_convert**: handles the conversion of consecutive digits in the infix expression. It accumulates the numeric value of the digits, appends them to the postfix expression, and pushes the accumulated value onto the value stack.
- ‘**case_operator_in_loop_convert**’: ensures that operators with higher precedence are popped from the operator stack, appended to the postfix expression, and the corresponding calculations are performed. The loop continues until an operator with lower precedence is encountered or the operator stack becomes empty. Finally, the current operator in the infix expression is pushed onto the operator stack.



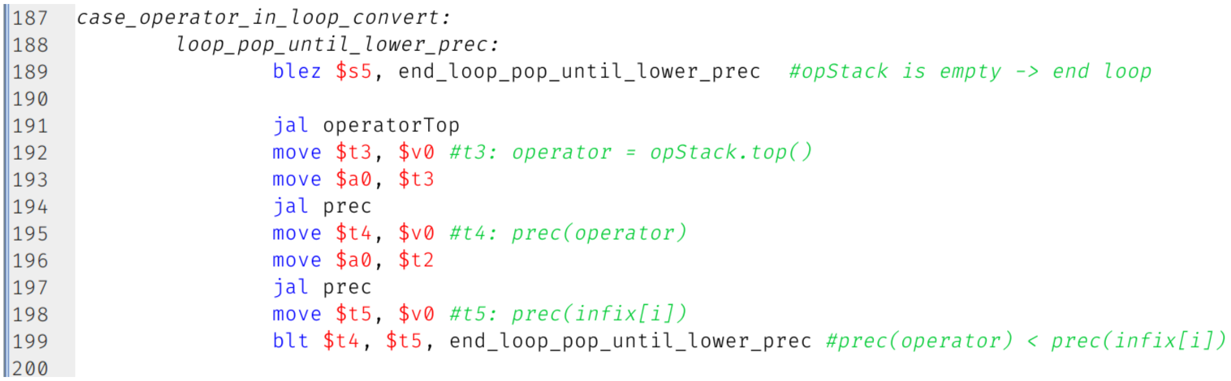
```
#int prec(char c)
#$a0: c
#return $v0
prec:
        beq $a0, '*', high_prec
        beq $a0, '/', high_prec
        beq $a0, '%', high_prec
        beq $a0, '+', low_prec
        beq $a0, '-', low_prec
        j default_prec
high_prec:
        li $v0, 2
        jr $ra
low_prec:
        li $v0, 1
        jr $ra
default_prec:
        li $v0, 0
        jr $ra
```



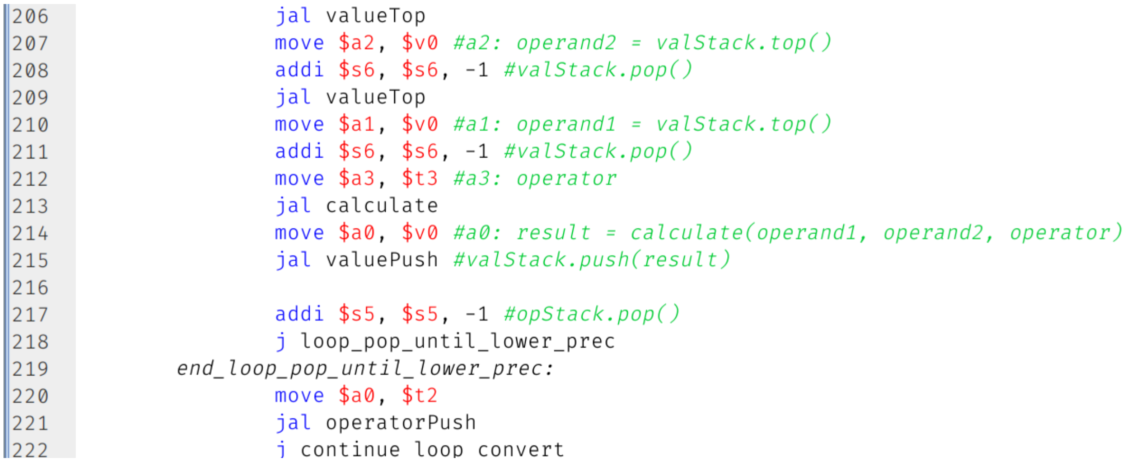
```
case_digit_in_loop_convert:
        li $t3, 0 #t3: val=0
        loop_digit:
                add $t1, $s0, $t0
                lb $t2, 0($t1) #t2: infix[i]

                move $a0, $t2
                jal isDigit #isDigit(infix[i])
                beq $v0, 0, end_loop_digit #infix[i] is not digit -> end loop

                mul $t3, $t3, 10 #val = val*10
                add $t3, $t3, $t2 #val = val*10 + infix[i]
                sub $t3, $t3, '0' #val = val*10 + (infix[i] - '0')

                move $a0, $t2
                jal postfixAppend #postfix.append(infix[i])
                addi $t0, $t0, 1 #i++
```



```
                j loop_digit
        end_loop_digit:
                move $a0, $t3
                jal valuePush #valStack.push(val)

                li $a0, ' '
                jal postfixAppend #postfix.append(' ')

                addi $t0, $t0, -1 #i--
                j continue_loop_convert
```

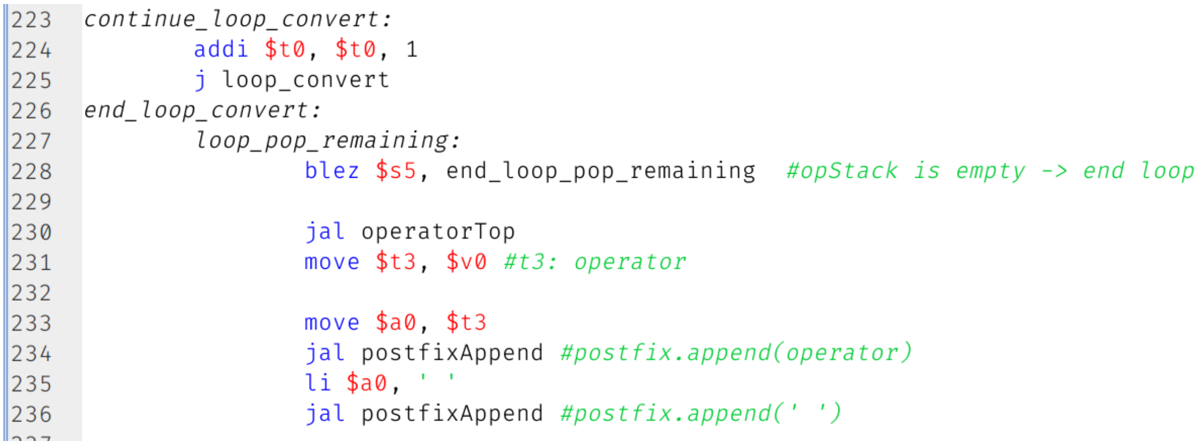


```
case_operator_in_loop_convert:
        loop_pop_until_lower_prec:
                blez $s5, end_loop_pop_until_lower_prec  #opStack is empty -> end loop

                jal operatorTop
                move $t3, $v0 #t3: operator = opStack.top()
                move $a0, $t3
                jal prec
                move $t4, $v0 #t4: prec(operator)
                move $a0, $t2
                jal prec
                move $t5, $v0 #t5: prec(infix[i])
                blt $t4, $t5, end_loop_pop_until_lower_prec #prec(operator) < prec(infix[i])

```

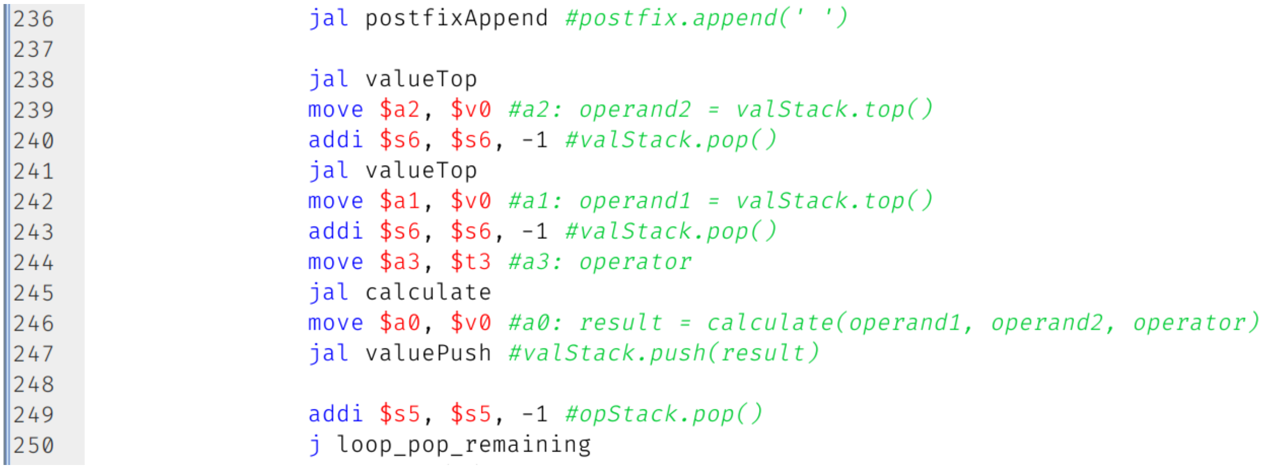


```
                jal valueTop
                move $a2, $v0 #a2: operand2 = valStack.top()
                addi $s6, $s6, -1 #valStack.pop()
                jal valueTop
                move $a1, $v0 #a1: operand1 = valStack.top()
                addi $s6, $s6, -1 #valStack.pop()
                move $a3, $t3 #a3: operator
                jal calculate
                move $a0, $v0 #a0: result = calculate(operand1, operand2, operator)
                jal valuePush #valStack.push(result)

                addi $s5, $s5, -1 #opStack.pop()
                j loop_pop_until_lower_prec
        end_loop_pop_until_lower_prec:
                move $a0, $t2
                jal operatorPush
                j continue_loop_convert
```

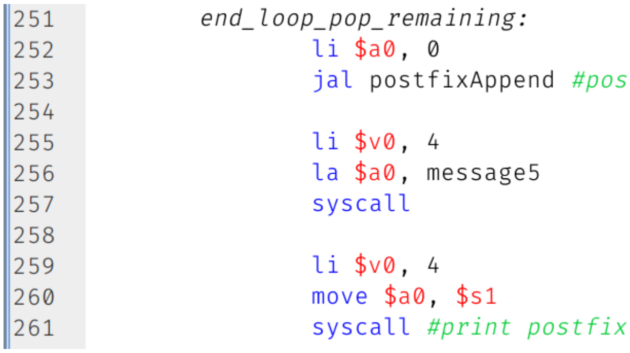
- ‘**end_loop_convert**’: is responsible for finalizing the conversion of the infix expression to postfix notation by popping any remaining operators from the operator stack, appending them to the postfix expression, and performing the corresponding calculations.



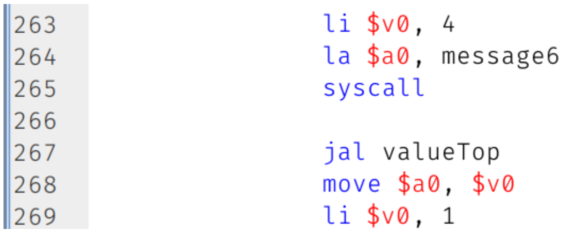
```
continue_loop_convert:
        addi $t0, $t0, 1
        j loop_convert
end_loop_convert:
        loop_pop_remaining:
                blez $s5, end_loop_pop_remaining  #opStack is empty -> end loop

                jal operatorTop
                move $t3, $v0 #t3: operator

                move $a0, $t3
                jal postfixAppend #postfix.append(operator)
                li $a0, ' '
                jal postfixAppend #postfix.append(' ')
```



```
                jal postfixAppend #postfix.append(' ')

                jal valueTop
                move $a2, $v0 #a2: operand2 = valStack.top()
                addi $s6, $s6, -1 #valStack.pop()
                jal valueTop
                move $a1, $v0 #a1: operand1 = valStack.top()
                addi $s6, $s6, -1 #valStack.pop()
                move $a3, $t3 #a3: operator
                jal calculate
                move $a0, $v0 #a0: result = calculate(operand1, operand2, operator)
                jal valuePush #valStack.push(result)

                addi $s5, $s5, -1 #opStack.pop()
                j loop_pop_remaining
```



```
        end_loop_pop_remaining:
                li $a0, 0
                jal postfixAppend #postfix.append('\0')

                li $v0, 4
                la $a0, message5
                syscall

                li $v0, 4
                move $a0, $s1
                syscall #print postfix
```

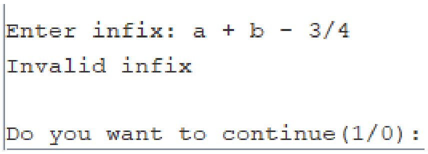


```
                li $v0, 4
                la $a0, message6
                syscall

                jal valueTop
                move $a0, $v0
                li $v0, 1
```

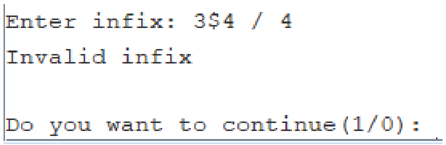
3. Results demonstration

a. Exception cases

- Invalid digits/characters error

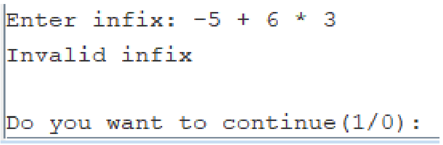


```
Enter infix: a + b - 3/4
Invalid infix

Do you want to continue(1/0):
```



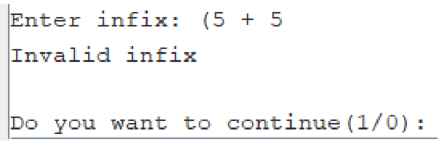
```
Enter infix: 3$4 / 4
Invalid infix

Do you want to continue(1/0):
```

- Negative numbers error

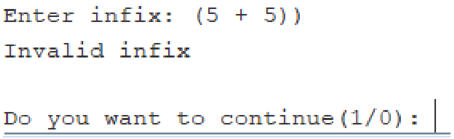


```
Enter infix: -5 + 6 * 3
Invalid infix

Do you want to continue(1/0):
```

- Parentheses error

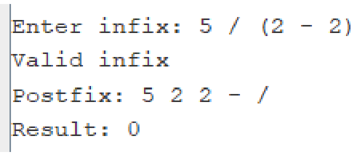


```
Enter infix: (5 + 5
Invalid infix

Do you want to continue(1/0):
```



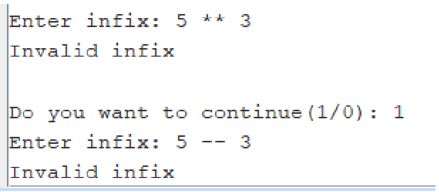
```
Enter infix: (5 + 5))
Invalid infix

Do you want to continue(1/0):
```

- Divide by 0 error (the result is always 0 by default)



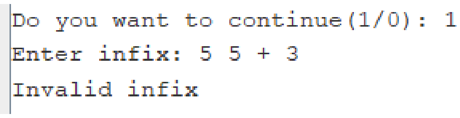
```
Enter infix: 5 / (2 - 2)
Valid infix
Postfix: 5 2 2 - /
Result: 0
```

- Two consecutive operators error



```
Enter infix: 5 ** 3
Invalid infix

Do you want to continue(1/0): 1
Enter infix: 5 -- 3
Invalid infix
```

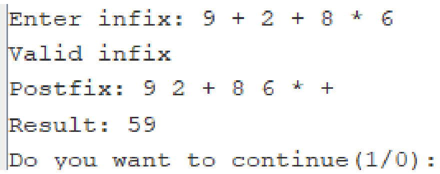
- Two consecutive operands error



```
Do you want to continue(1/0): 1
Enter infix: 5 5 + 3
Invalid infix
```

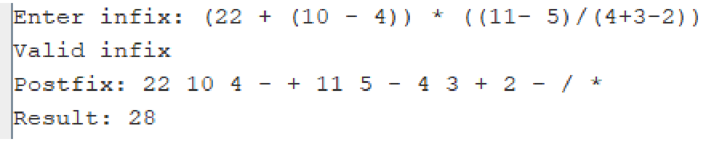
b. Various results

9 + 2 + 8 * 6



```
Enter infix: 9 + 2 + 8 * 6
Valid infix
Postfix: 9 2 + 8 6 * +
Result: 59
Do you want to continue(1/0):
```

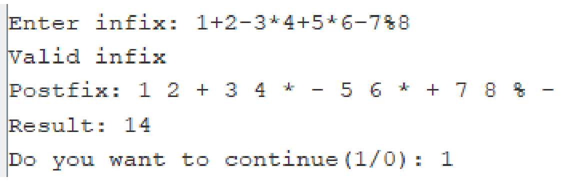
(22 + (10 - 4)) * ((11- 5)/(4+3-2))



```
Enter infix: (22 + (10 - 4)) * ((11- 5)/(4+3-2))
Valid infix
Postfix: 22 10 4 - + 11 5 - 4 3 + 2 - / *
Result: 28
```

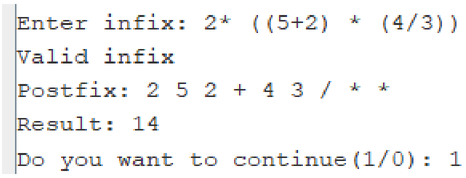
(*Note:* Because the expression is rounded from the division (11- 5)/(4+3-2), the result will be different compared to the exact calculation)

1+2-3*4+5*6-7%8



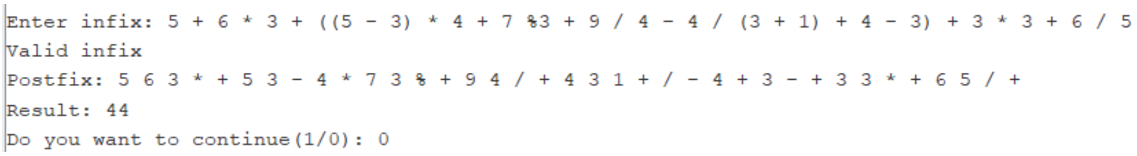
```
Enter infix: 1+2-3*4+5*6-7%8
Valid infix
Postfix: 1 2 + 3 4 * - 5 6 * + 7 8 % -
Result: 14
Do you want to continue(1/0): 1
```

2* ((5+2) * (4/3))



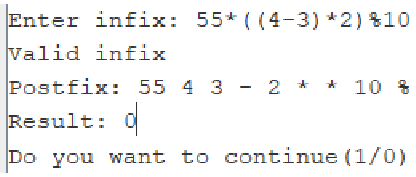
```
Enter infix: 2* ((5+2) * (4/3))
Valid infix
Postfix: 2 5 2 + 4 3 / * *
Result: 14
Do you want to continue(1/0): 1
```

5 + 6 * 3 + ((5 - 3) * 4 + 7 %3 + 9 / 4 - 4 / (3 + 1) + 4 - 3) + 3 * 3 + 6 / 5



```
Enter infix: 5 + 6 * 3 + ((5 - 3) * 4 + 7 %3 + 9 / 4 - 4 / (3 + 1) + 4 - 3) + 3 * 3 + 6 / 5
Valid infix
Postfix: 5 6 3 * + 5 3 - 4 * 7 3 % + 9 4 / + 4 3 1 + / - 4 + 3 - + 3 3 * + 6 5 / +
Result: 44
Do you want to continue(1/0): 0
```

55*((4-3)*2)%10



```
Enter infix: 55*((4-3)*2)%10
Valid infix
Postfix: 55 4 3 - 2 * * 10 %
Result: 0
Do you want to continue(1/0): 1
```

II. Task 9: Drawing shape using ASCII characters

1. Problem description

Given a picture translated to ASCII characters as follows, this is the shapes of DCE with border * and colors are digits.

- Show this picture in the console window.
- Change the picture so that DCE has only a border without color inside.
- Change the order of DCE to ECD.
- Enter the new color number from the keyboard, update the picture with new colors.

Note: *Except the memory used to store the picture in source code, do not use any extra memory space.*

```
                                           *************
**************                            *3333333333333*
*222222222222222*                         *33333********
*22222******222222*                       *33333*
*22222*      *22222*                      *33333********
*22222*       *22222*      *************  *3333333333333*
*22222*       *22222*    **11111*****111* *33333********
*22222*       *22222*  **1111**       **  *33333*
*22222*      *222222*  *1111*             *33333********
*22222*******222222*  *11111*             *3333333333333*
*2222222222222222*    *11111*              *************
***************       *11111*
      ---              *1111**
    / o o \             *1111****   *****
    \   > /              **111111***111*
     -----                 ***********    dce.hust.edu.vn
```

2. Project implementation

a. Algorithm

- **Show picture with color**

  The program traverses through all 16 lines of the picture and print them in the console window.

- **Show picture without color**

  The program goes through all 16 lines of the picture. In each line, it traverses through every character, checks whether they are digit or not, and print them in the console window. If the character is digit, it will be printed as blank space.

- **Change the order of DCE to ECD**

  The program goes through all 16 lines of the picture. It is easily noticed that words ‘D’, ‘C’, ‘E’ are separated by blank spaces, so we can temporarily replace blank spaces by null characters. In each line, we will print character sequences of ‘E’ first, then ‘C’ and ‘D’, and finally a line feed character. After printing all characters in the order we want in each line, we need to restore the blank spaces to their previous states.

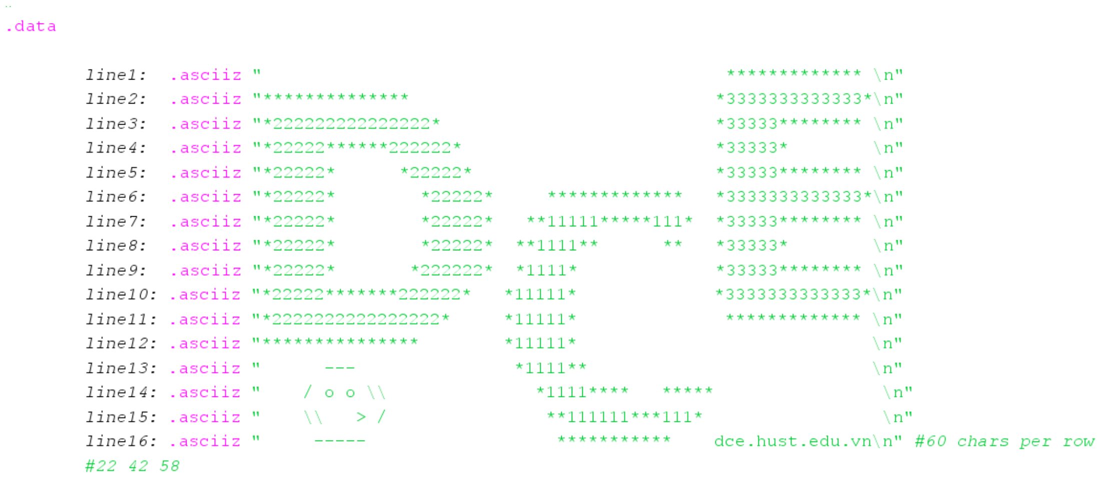
- **Update the picture with new colors**

  Firstly, we ask user to enter new colors for ‘D’, ‘C’, ‘E’. If the colors are not valid, the program asks the user to reenter until receiving valid colors.

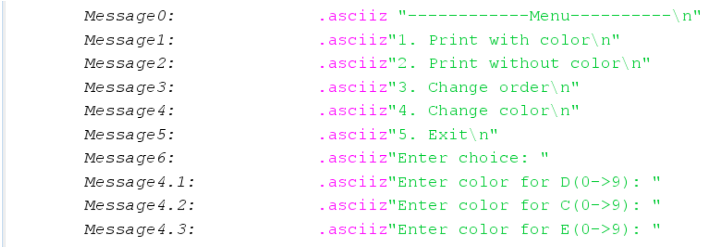
  We need to track both old and new colors of ‘D’, ‘C’, ‘E’. We go through all 16 lines of the picture. In each line, we sequentially check 3 words. If any characters are old colors, we update them to new colors corresponding to each word by storing new values in the memory, so that in the future, we can see the updated picture. While updating, we also print the new picture to see the effect. After finishing updating the picture, we update the current color values of ‘D’, ‘C’, ‘E’.

b. Code explanation

- Data section includes all lines of the picture and necessary message strings

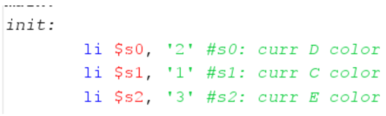


```
.data

        line1:  .asciiz "                                             ************* \n"
        line2:  .asciiz "**************                              *3333333333333*\n"
        line3:  .asciiz "*222222222222222*                           *33333******** \n"
        line4:  .asciiz "*22222******222222*                         *33333*        \n"
        line5:  .asciiz "*22222*      *22222*                        *33333******** \n"
        line6:  .asciiz "*22222*        *22222*     *************    *3333333333333*\n"
        line7:  .asciiz "*22222*        *22222*   **11111*****111*   *33333******** \n"
        line8:  .asciiz "*22222*        *22222*  **1111**      **    *33333*        \n"
        line9:  .asciiz "*22222*       *222222*  *1111*              *33333******** \n"
        line10: .asciiz "*22222*******222222*   *11111*              *3333333333333*\n"
        line11: .asciiz "*2222222222222222*     *11111*               ************* \n"
        line12: .asciiz "***************        *11111*                             \n"
        line13: .asciiz "      ---               *1111**                            \n"
        line14: .asciiz "    / o o \\              *1111****    *****                \n"
        line15: .asciiz "    \\   > /               **111111***111*                  \n"
        line16: .asciiz "     -----                  ***********     dce.hust.edu.vn\n" #60 chars per row
        #22 42 58
```



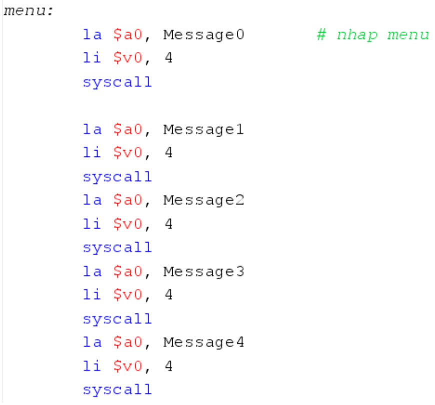
```
        Message0:               .asciiz "------------Menu----------\n"
        Message1:               .asciiz"1. Print with color\n"
        Message2:               .asciiz"2. Print without color\n"
        Message3:               .asciiz"3. Change order\n"
        Message4:               .asciiz"4. Change color\n"
        Message5:               .asciiz"5. Exit\n"
        Message6:               .asciiz"Enter choice: "
        Message4.1:             .asciiz"Enter color for D(0->9): "
        Message4.2:             .asciiz"Enter color for C(0->9): "
        Message4.3:             .asciiz"Enter color for E(0->9): "
```

- Initialize current color values of 3 words

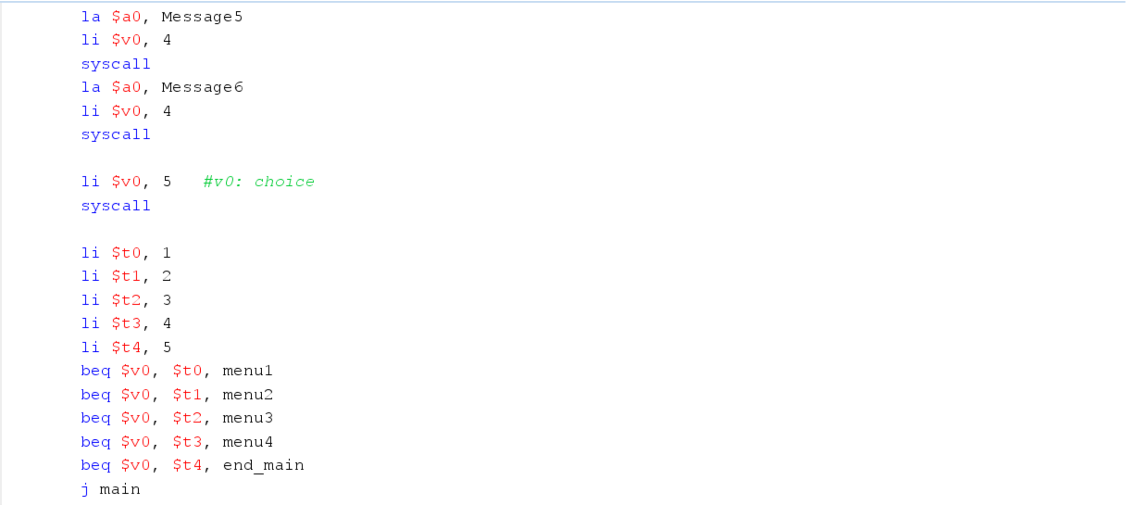


```
init:
        li $s0, '2' #s0: curr D color
        li $s1, '1' #s1: curr C color
        li $s2, '3' #s2: curr E color
```

- Show the menu and ask the user to enter a choice. If the choice is invalid, ask the user to enter again.

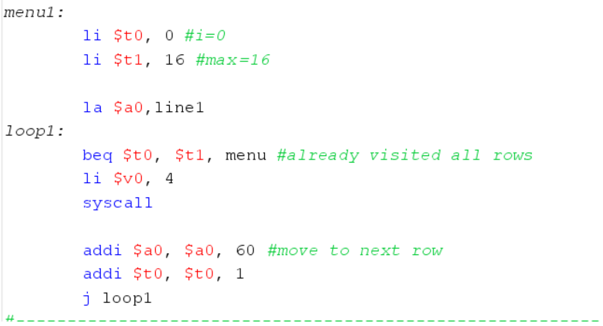


```
menu:
        la $a0, Message0        # nhap menu
        li $v0, 4
        syscall

        la $a0, Message1
        li $v0, 4
        syscall
        la $a0, Message2
        li $v0, 4
        syscall
        la $a0, Message3
        li $v0, 4
        syscall
        la $a0, Message4
        li $v0, 4
        syscall
```



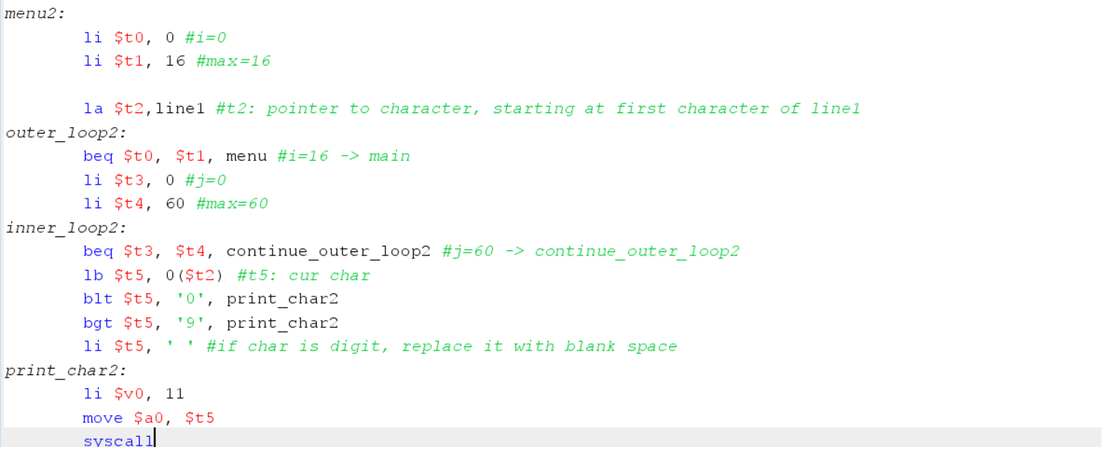
```
        la $a0, Message5
        li $v0, 4
        syscall
        la $a0, Message6
        li $v0, 4
        syscall

        li $v0, 5   #v0: choice
        syscall

        li $t0, 1
        li $t1, 2
        li $t2, 3
        li $t3, 4
        li $t4, 5
        beq $v0, $t0, menu1
        beq $v0, $t1, menu2
        beq $v0, $t2, menu3
        beq $v0, $t3, menu4
        beq $v0, $t4, end_main
        j main
```

- The first option is to show the picture with color. We go through all 16 lines, use $a0 as the pointer to base address of each line and print them. After an iteration, we increase $a0 by 60 because of 60 characters in each line.

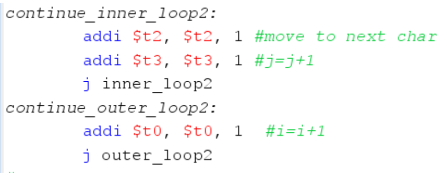


```
menu1:
        li $t0, 0 #i=0
        li $t1, 16 #max=16

        la $a0,line1
loop1:
        beq $t0, $t1, menu #already visited all rows
        li $v0, 4
        syscall

        addi $a0, $a0, 60 #move to next row
        addi $t0, $t0, 1
        j loop1
#----------------------------------------------------------
```

- The second option is to show the picture with only a border. We use nested loops to go through every character of every line. If the character is not digit, just print it. Otherwise, print it as a blank space.

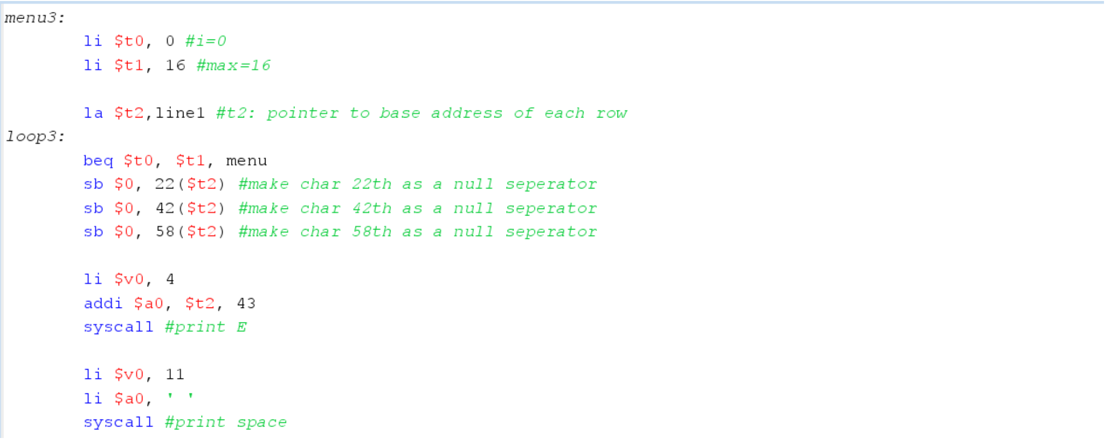


```
menu2:
        li $t0, 0 #i=0
        li $t1, 16 #max=16

        la $t2,line1 #t2: pointer to character, starting at first character of line1
outer_loop2:
        beq $t0, $t1, menu #i=16 -> main
        li $t3, 0 #j=0
        li $t4, 60 #max=60
inner_loop2:
        beq $t3, $t4, continue_outer_loop2 #j=60 -> continue_outer_loop2
        lb $t5, 0($t2) #t5: cur char
        blt $t5, '0', print_char2
        bgt $t5, '9', print_char2
        li $t5, ' ' #if char is digit, replace it with blank space
print_char2:
        li $v0, 11
        move $a0, $t5
        syscall
```

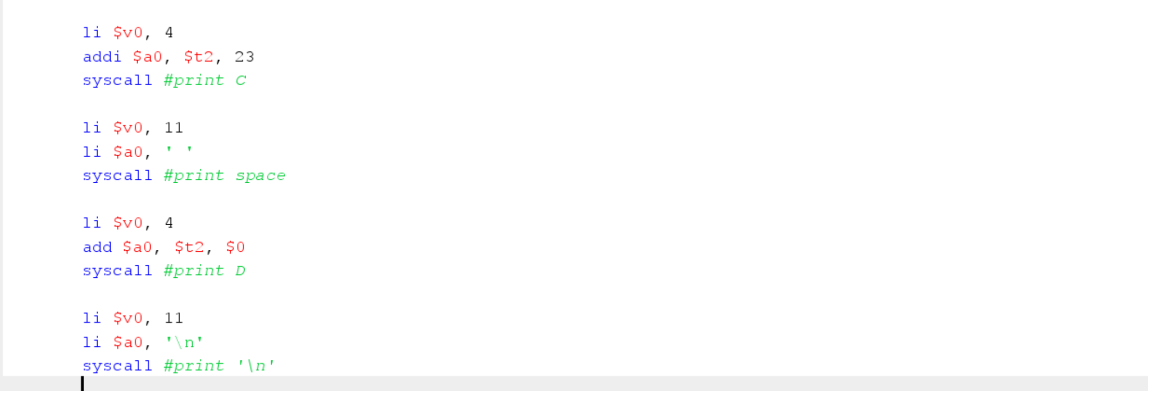


```
continue_inner_loop2:
	addi $t2, $t2, 1 #move to next char
	addi $t3, $t3, 1 #j=j+1
	j inner_loop2
continue_outer_loop2:
	addi $t0, $t0, 1  #i=i+1
	j outer_loop2
```

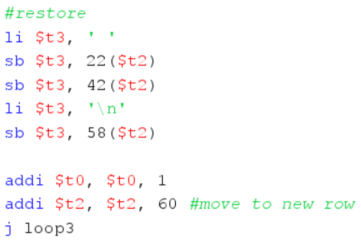
- The third option is to print the picture in the order ‘E’, ‘C’, ‘D’. We use 1 loop to traverse through every line. In each line, we temporarily update the 22nd, 42nd, 58th characters as null terminated characters in the memory because these positions are the boundaries among words. Then, we can easily print words ‘E’, ‘C’, ‘D’ with the help of null terminated characters. We also print spaces between words, and line feed at the end for the purpose of clear demonstration. Finally, we need to restore the state of the picture in the memory.



```
menu3:
	li $t0, 0 #i=0
	li $t1, 16 #max=16

	la $t2,line1 #t2: pointer to base address of each row
loop3:
	beq $t0, $t1, menu
	sb $0, 22($t2) #make char 22th as a null seperator
	sb $0, 42($t2) #make char 42th as a null seperator
	sb $0, 58($t2) #make char 58th as a null seperator

	li $v0, 4
	addi $a0, $t2, 43
	syscall #print E

	li $v0, 11
	li $a0, ' '
	syscall #print space
```

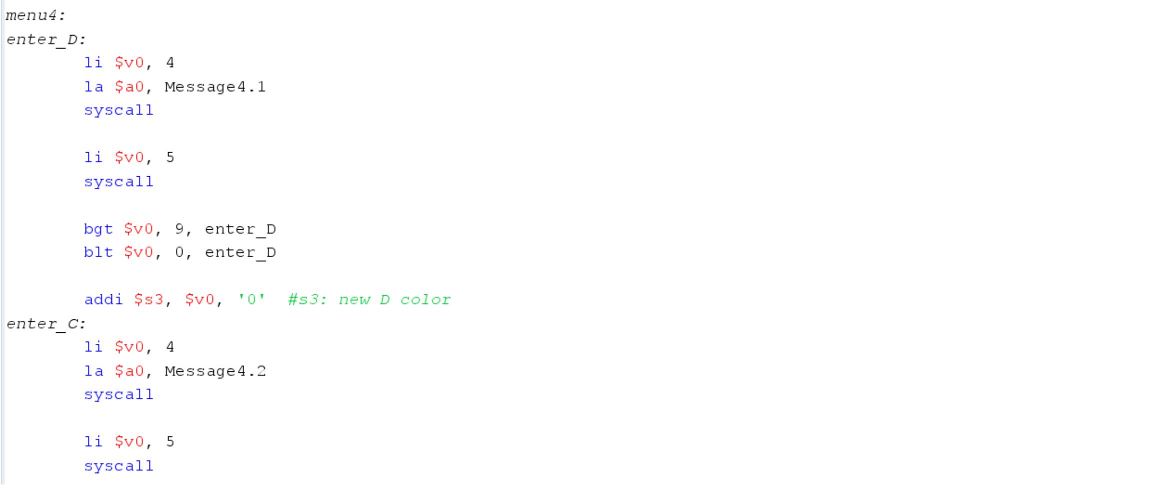


```
	li $v0, 4
	addi $a0, $t2, 23
	syscall #print C

	li $v0, 11
	li $a0, ' '
	syscall #print space

	li $v0, 4
	add $a0, $t2, $0
	syscall #print D

	li $v0, 11
	li $a0, '\n'
	syscall #print '\n'
```

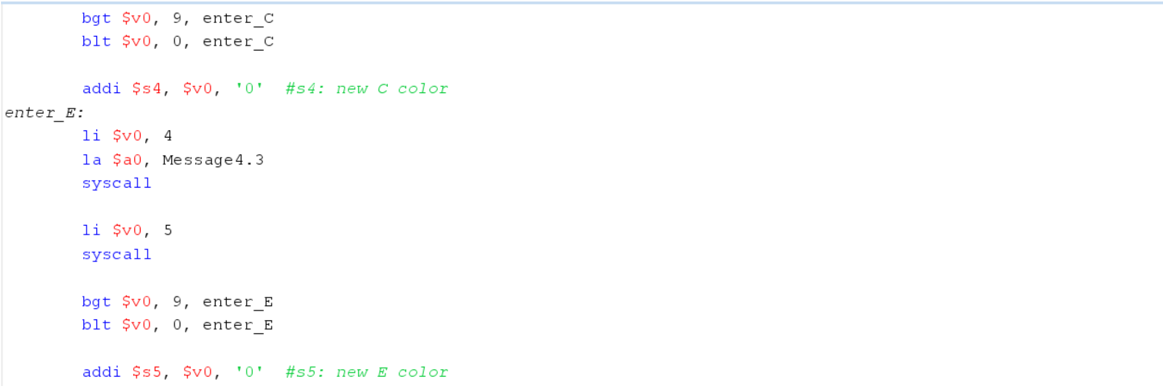


```
	#restore
	li $t3, ' '
	sb $t3, 22($t2)
	sb $t3, 42($t2)
	li $t3, '\n'
	sb $t3, 58($t2)

	addi $t0, $t0, 1
	addi $t2, $t2, 60 #move to new row
	j loop3
```

- The fourth option is to update the picture with the new color entered by user. We ask the user to enter 3 new colors, store them in $s3, $s4, $s5, and ask to reenter if any color is invalid.

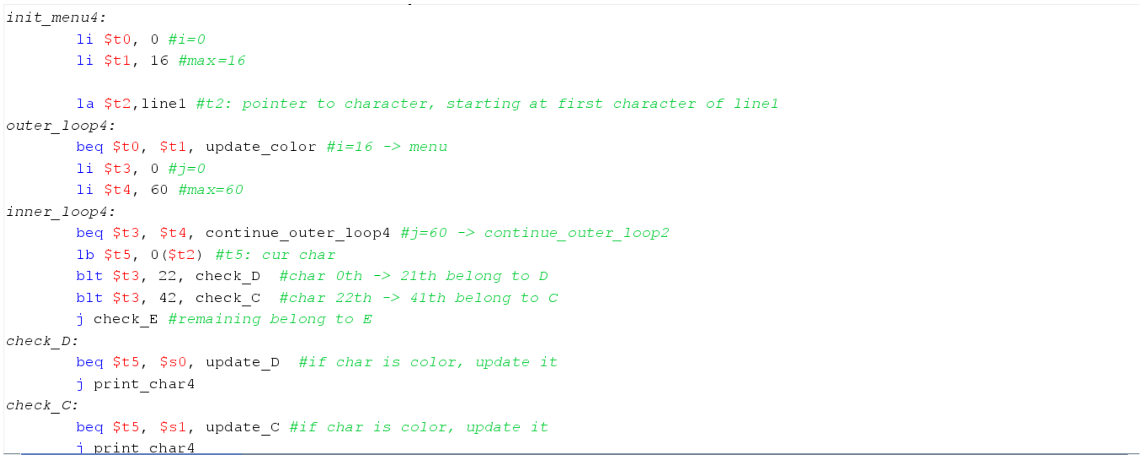


```
menu4:
enter_D:
        li $v0, 4
        la $a0, Message4.1
        syscall

        li $v0, 5
        syscall

        bgt $v0, 9, enter_D
        blt $v0, 0, enter_D

        addi $s3, $v0, '0'  #s3: new D color
enter_C:
        li $v0, 4
        la $a0, Message4.2
        syscall

        li $v0, 5
        syscall
```

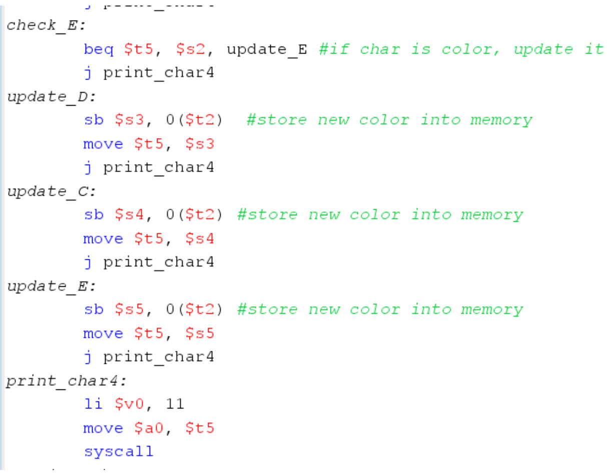


```
        bgt $v0, 9, enter_C
        blt $v0, 0, enter_C

        addi $s4, $v0, '0'  #s4: new C color
enter_E:
        li $v0, 4
        la $a0, Message4.3
        syscall

        li $v0, 5
        syscall

        bgt $v0, 9, enter_E
        blt $v0, 0, enter_E

        addi $s5, $v0, '0'  #s5: new E color
```

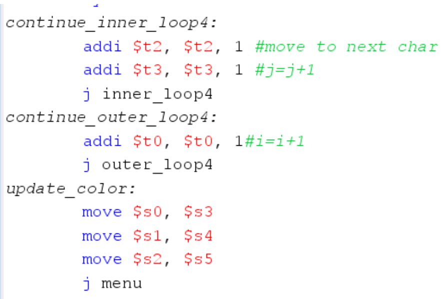
We use nested loops to traverse through every character of every line. In each line, the first 22 characters belong to ‘D’, the next 20 characters belong to ‘C’ and the remaining belong to ‘E’. We handle characters of each word with 3 branches ‘check_D’, ‘check_C’ and ‘check_E’. In each branch, if the character is digit, we update it with new corresponding color in the memory with 1 of 3 branchs ‘update_D’, ‘update_C’, ‘update_E’. We also print all characters to the console to see the updated effect. After looping through all lines, we update current color values ($s0, $s1, $s2) with new values ($s3, $s4, $s5)



```
init_menu4:
	li $t0, 0 #i=0
	li $t1, 16 #max=16

	la $t2,line1 #t2: pointer to character, starting at first character of line1
outer_loop4:
	beq $t0, $t1, update_color #i=16 -> menu
	li $t3, 0 #j=0
	li $t4, 60 #max=60
inner_loop4:
	beq $t3, $t4, continue_outer_loop4 #j=60 -> continue_outer_loop2
	lb $t5, 0($t2) #t5: cur char
	blt $t3, 22, check_D  #char 0th -> 21th belong to D
	blt $t3, 42, check_C  #char 22th -> 41th belong to C
	j check_E #remaining belong to E
check_D:
	beq $t5, $s0, update_D  #if char is color, update it
	j print_char4
check_C:
	beq $t5, $s1, update_C #if char is color, update it
	j print_char4
```

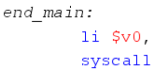


```
check_E:
	beq $t5, $s2, update_E #if char is color, update it
	j print_char4
update_D:
	sb $s3, 0($t2)  #store new color into memory
	move $t5, $s3
	j print_char4
update_C:
	sb $s4, 0($t2) #store new color into memory
	move $t5, $s4
	j print_char4
update_E:
	sb $s5, 0($t2) #store new color into memory
	move $t5, $s5
	j print_char4
print_char4:
	li $v0, 11
	move $a0, $t5
	syscall
```



```
continue_inner_loop4:
	addi $t2, $t2, 1 #move to next char
	addi $t3, $t3, 1 #j=j+1
	j inner_loop4
continue_outer_loop4:
	addi $t0, $t0, 1#i=i+1
	j outer_loop4
update_color:
	move $s0, $s3
	move $s1, $s4
	move $s2, $s5
	j menu
```

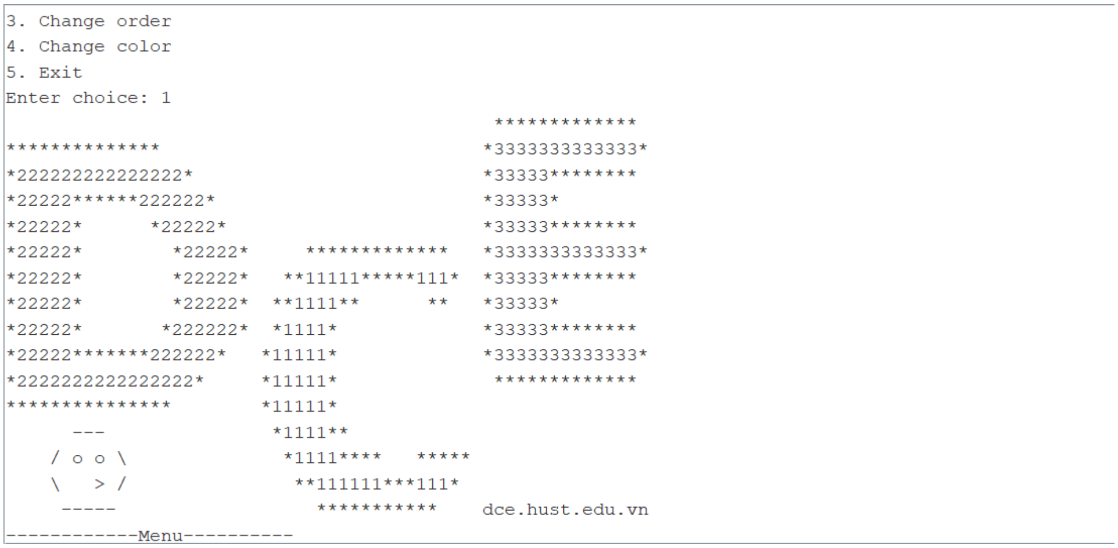
- The last option is to exit. It jumps to end_main.



```
end_main:
	li $v0, 10
	syscall
```

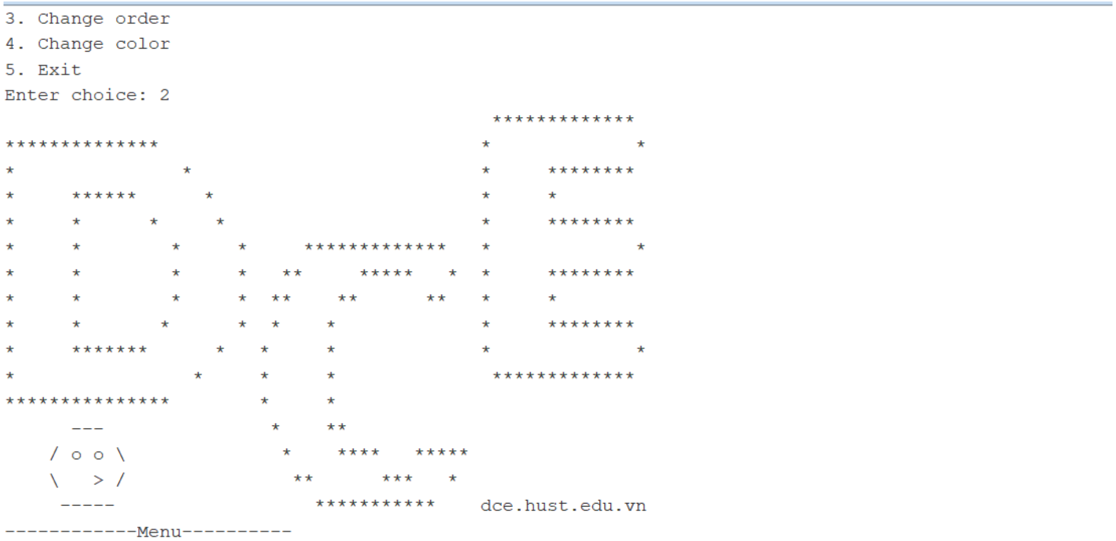
3. Results demonstration

- **Show picture with color**



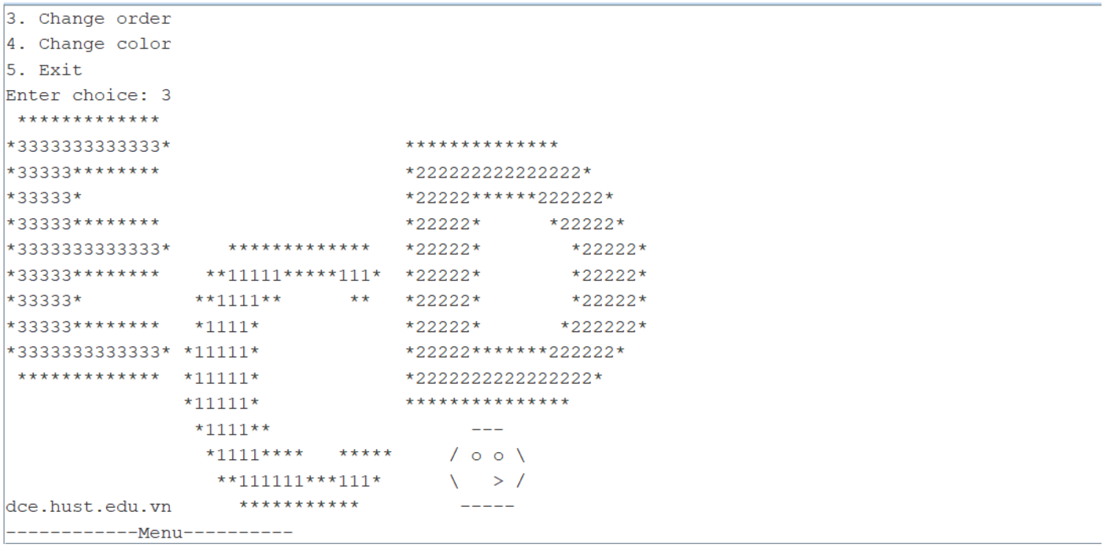
```
3. Change order
4. Change color
5. Exit
Enter choice: 1
                                            *************
**************                             *3333333333333*
*222222222222222*                          *33333********
*22222******222222*                        *33333*
*22222*      *22222*                       *33333********
*22222*        *22222*     *************   *3333333333333*
*22222*        *22222*   **11111*****111*  *33333********
*22222*        *22222*  **1111**      **   *33333*
*22222*       *222222*  *1111*             *33333********
*22222*******222222*   *11111*             *3333333333333*
*2222222222222222*     *11111*              *************
***************        *11111*
      ---               *1111**
    / o o \              *1111****   *****
    \   > /               **111111***111*
     -----                  ***********    dce.hust.edu.vn
------------Menu----------
```

- **Show picture without color**



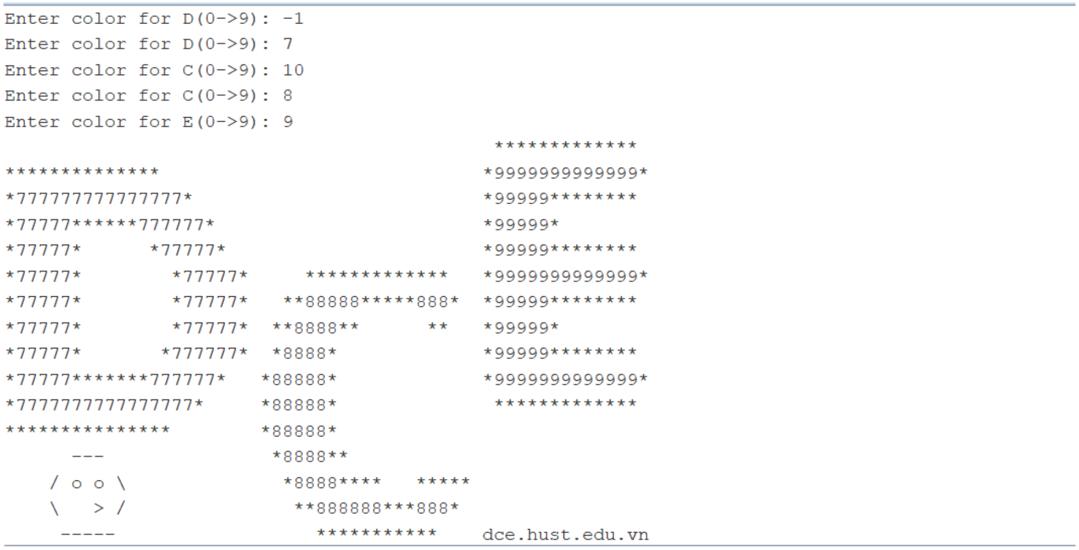
```
3. Change order
4. Change color
5. Exit
Enter choice: 2
                                            *************
**************                             *             *
*               *                          *     ********
*     ******      *                        *     *
*     *      *     *                       *     ********
*     *        *     *     *************   *             *
*     *        *     *   **     *****   *  *     ********
*     *        *     *  **    **      **   *     *
*     *       *      *  *    *             *     ********
*     *******      *   *     *             *             *
*                *     *     *              *************
***************        *     *
      ---               *    **
    / o o \              *    ****   *****
    \   > /               **      ***   *
     -----                  ***********    dce.hust.edu.vn
------------Menu----------
```

- **Show picture in the order “ECD”**



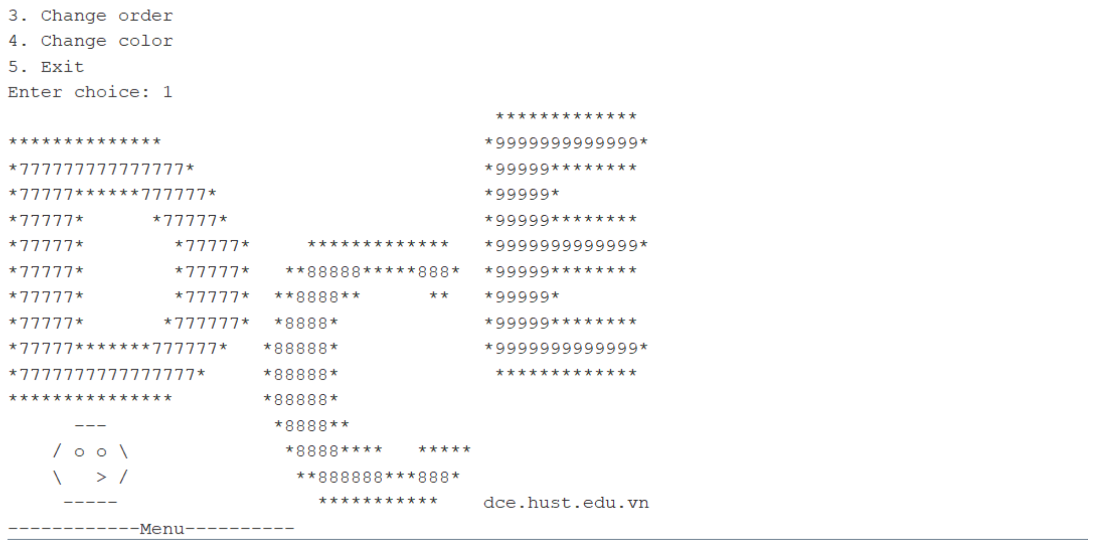
```
3. Change order
4. Change color
5. Exit
Enter choice: 3
 *************
*3333333333333*                     **************
*33333********                      *222222222222222*
*33333*                             *22222******222222*
*33333********                      *22222*      *22222*
*3333333333333*     *************   *22222*        *22222*
*33333********    **11111*****111*  *22222*        *22222*
*33333*          **1111**      **   *22222*        *22222*
*33333********   *1111*             *22222*       *222222*
*3333333333333* *11111*             *22222*******222222*
 *************  *11111*             *2222222222222222*
                *11111*             ***************
                 *1111**                  ---
                  *1111****   *****     / o o \
                   **111111***111*      \   > /
dce.hust.edu.vn      ***********         -----
------------Menu----------
```

- Update the picture with new colors

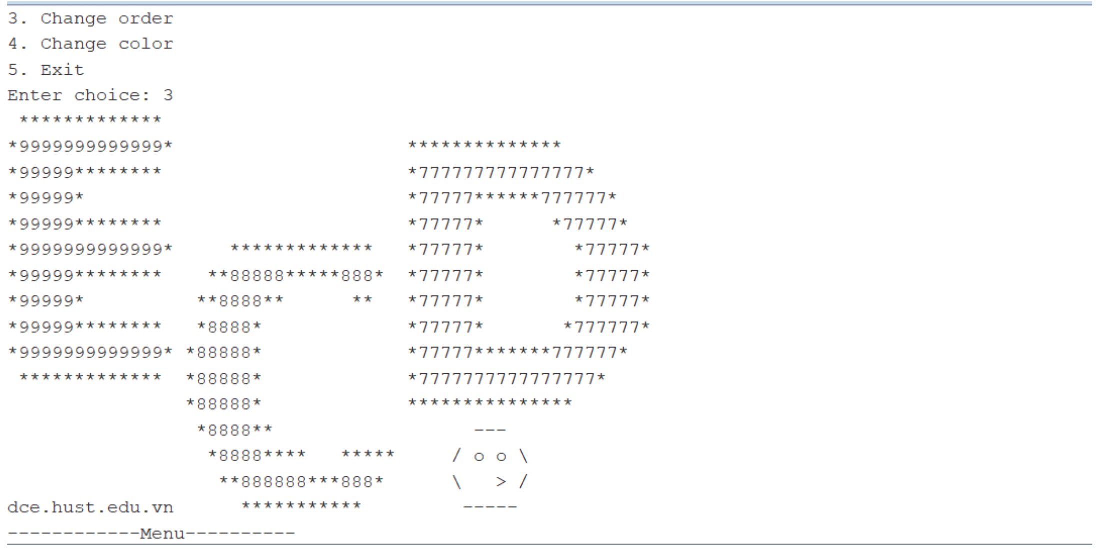


```
Enter color for D(0->9): -1
Enter color for D(0->9): 7
Enter color for C(0->9): 10
Enter color for C(0->9): 8
Enter color for E(0->9): 9
                                            *************
**************                             *9999999999999*
*777777777777777*                          *99999********
*77777******777777*                        *99999*
*77777*      *77777*                       *99999********
*77777*        *77777*     *************   *9999999999999*
*77777*        *77777*   **88888*****888*  *99999********
*77777*        *77777*  **8888**      **   *99999*
*77777*       *777777*  *8888*             *99999********
*77777*******777777*   *88888*             *9999999999999*
*7777777777777777*     *88888*              *************
***************        *88888*
      ---               *8888**
    / o o \              *8888****   *****
    \   > /               **888888***888*
     -----                  ***********    dce.hust.edu.vn
```

The picture is updated and applied to all choices in the future.

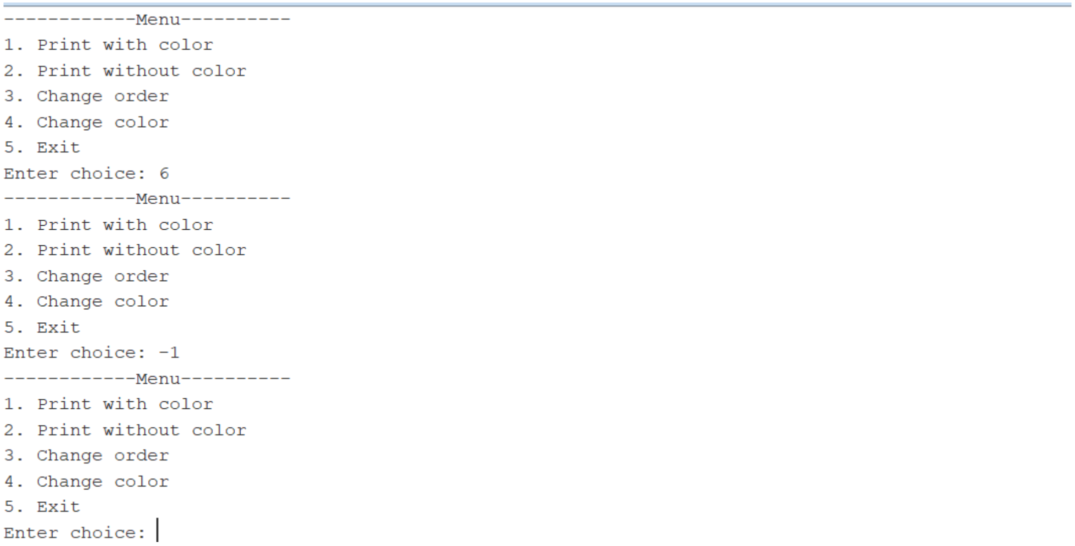


```
3. Change order
4. Change color
5. Exit
Enter choice: 1
                                            *************
**************                             *9999999999999*
*777777777777777*                          *99999********
*77777******777777*                        *99999*
*77777*      *77777*                       *99999********
*77777*        *77777*     *************   *9999999999999*
*77777*        *77777*   **88888*****888*  *99999********
*77777*        *77777*  **8888**      **   *99999*
*77777*       *777777*  *8888*             *99999********
*77777*******777777*   *88888*             *9999999999999*
*7777777777777777*     *88888*              *************
***************        *88888*
      ---               *8888**
    / o o \              *8888****   *****
    \   > /               **888888***888*
     -----                  ***********    dce.hust.edu.vn
------------Menu----------
```



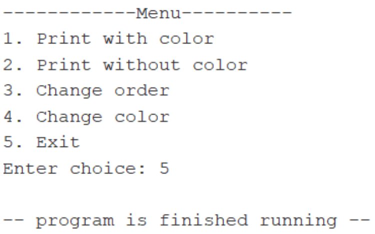
```
3. Change order
4. Change color
5. Exit
Enter choice: 3
 *************
*9999999999999*                     **************
*99999********                      *777777777777777*
*99999*                             *77777******777777*
*99999********                      *77777*      *77777*
*9999999999999*     *************   *77777*        *77777*
*99999********    **88888*****888*  *77777*        *77777*
*99999*          **8888**      **   *77777*        *77777*
*99999********   *8888*             *77777*       *777777*
*9999999999999* *88888*             *77777*******777777*
 *************  *88888*             *7777777777777777*
                *88888*             ***************
                 *8888**                  ---
                  *8888****   *****     / o o \
                   **888888***888*      \   > /
dce.hust.edu.vn       ***********        -----
------------Menu----------
```

- **Invalid option**



```
------------Menu----------
1. Print with color
2. Print without color
3. Change order
4. Change color
5. Exit
Enter choice: 6
------------Menu----------
1. Print with color
2. Print without color
3. Change order
4. Change color
5. Exit
Enter choice: -1
------------Menu----------
1. Print with color
2. Print without color
3. Change order
4. Change color
5. Exit
Enter choice: 
```

- **Exit the program**



```
------------Menu----------
1. Print with color
2. Print without color
3. Change order
4. Change color
5. Exit
Enter choice: 5

-- program is finished running --
```