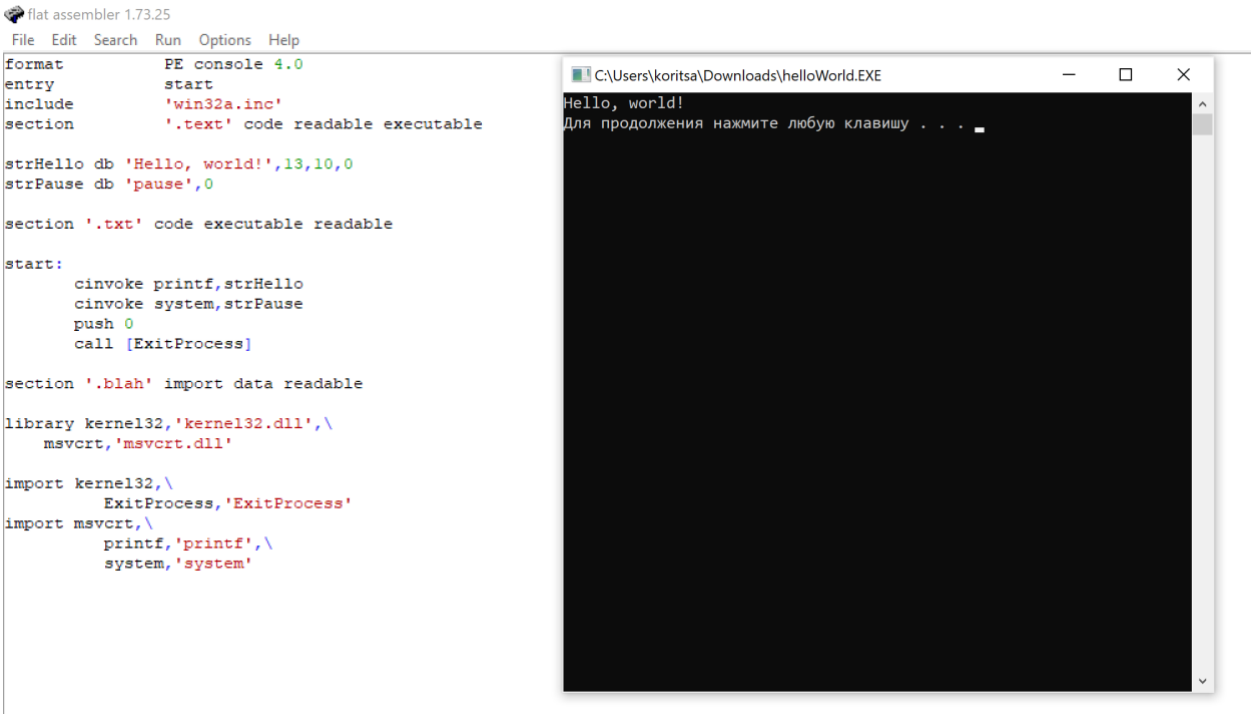
Корицкий Александр Юрьевич, БПИ196, ДЗ #1

Программа #1

Программа, которая просто выводит на экран строчку «Hello, world!». Посчитал, что это самая простая программа для начала освоения нового языка.



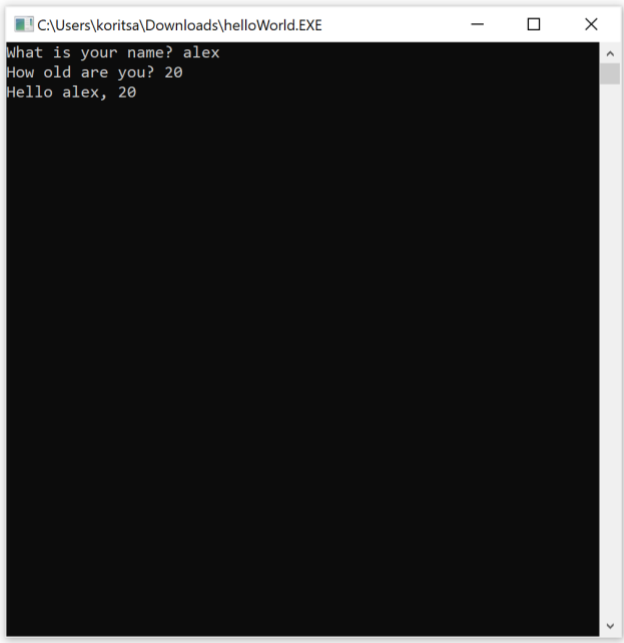
Программа #2

Вторая программа получает на вход от пользователя его имя и возраст и выводитт информацию с этими данными.
Например, как на скрине, первые две строчки – это запрос данных от пользователя, а третья – выводит программа с данными, которые ввел пользователь.

flat assembler 1.73.25

File Edit Search Run Options Help

```
section '.data' data readable writable

        formatStr db '%s', 0
        formatNum db '%d', 0

        name rd 2
        age rd 1

        wn db 'What is your name? ', 0
        ho db 'How old are you? ', 0

        hello db 'Hello %s, %d', 0

        NULL = 0

section '.code' code readable executable
        start:
                push wn
                call [printf]

                push name
                push formatStr
                call [scanf]

                push ho
                call [printf]

                push age
                push formatNum
                call [scanf]

                push [age]
                push name
                push hello
                call [printf]

                call[getch]

                push NULL
                call [ExitProcess]

section '.idata' import data readable
        library kernel, 'kernel32.dll', msvcrt, 'msvcrt.dll'

        import kernel, ExitProcess, 'ExitProcess'

        import msvcrt, printf, 'printf', getch, '_getch', scanf, 'scanf'
```



Программа #3-4-5

В качестве следующих задач я решил осветить простейшие математические операции. Однако после реализации этих задач подумал, что было бы здорово реализовать калькулятор на FASM. Калькулятор умеет принимать на вход две переменных и дальше необходимую нам математическую операцию, то есть +, -, * или /.

Плюс:

```
Enter A: 10
Enter B: 5
Enter operation: Result: 15
```

Минус:

```
Enter A: 20
Enter B: 6
Enter operation: Result: 14
```

Умножение:

```
Enter A: 10
Enter B: 10
Enter operation: Result: 100
```

Деление:

То есть я не просто сделал простейшие задачи на вычисление, но и научился определять введенную математическую операцию, который ввел пользователь. Перед появлением результата пользователь вводит эту операцию. В качестве доказательства привожу листинг программы и ее скрины в среде разработки.

Листинг:

```
format PE console

entry Start

include 'win32a.inc'

section '.data' data readable writable

        strA db 'Enter A: ', 0
        strB db 'Enter B: ', 0
        strOp db 'Enter operation: ', 0

        resStr db 'Result: %d', 0
        resMod db '/%d', 0

        spaceStr db ' %d', 0
        emptyStr db '%d', 0

        infinity db 'infinity', 0
        point db ',', 0

        A dd ?
        B dd ?
        C dd ?
        NULL = 0

section '.code' code readable executable

    Start:
        push strA
        call [printf]

        push A
        push spaceStr
        call [scanf]

        push strB
```

```
call [printf]

push B
push spaceStr
call [scanf]

push strOp
call [printf]

call [getch]

cmp eax, 43
jne notAdd
   mov ecx, [A]
   add ecx, [B]

   push ecx
   push resStr
   call [printf]

   jmp finish
 notAdd:

 cmp eax, 45
 jne notSub
     mov ecx, [A]
     sub ecx, [B]

     push ecx
     push resStr
     call [printf]

     jmp finish
 notSub:

 cmp eax, 42
 jne notMul
    mov ecx, [A]
    imul ecx, [B]

    push ecx
    push resStr
    call[printf]

    jmp finish
 notMul:

cmp eax, 47
jne notDiv
```

```
  mov eax, [A]
  mov ecx, [B]
  mov edx, 0

  cmp[B], 0
  jne notNullDiv1
     push infinity
     call[printf]

     jmp finish
  notNullDiv1:

  div ecx
  mov [C], edx

  push eax
  push resStr
  call [printf]

  push point
  call [printf]

  mov ebx, 0
  lp:

       mov eax, [C]
       mov ecx, [B]
       imul eax, 10

       mov edx, 0
       div ecx
       mov [C], edx

       push eax
       push emptyStr
       call [printf]


  add ebx, 1
  cmp ebx, 3
  jne lp
  jmp finish
notDiv:



 finish:

 call [getch]
```

```
        push NULL
        call [ExitProcess]



section '.idata' import data readable

        library kernel, 'kernel32.dll',\
              msvcrt, 'msvcrt.dll'

        import kernel,\
             ExitProcess, 'ExitProcess'

        import msvcrt,\
             printf, 'printf',\
             scanf, 'scanf',\
             getch, '_getch'
```

Скрины:

```
format PE console

entry Start

include 'win32a.inc'

section '.data' data readable writable

            strA db 'Enter A: ', 0
            strB db 'Enter B: ', 0
            strOp db 'Enter operation: ', 0

            resStr db 'Result: %d', 0
            resMod db '/%d', 0

            spaceStr db ' %d', 0
            emptyStr db '%d', 0

            infinity db 'infinity', 0
            point db ',', 0

            A dd ?
            B dd ?
            C dd ?
            NULL = 0

section '.code' code readable executable

        Start:
                push strA
                call [printf]

                push A
                push spaceStr
                call [scanf]

                push strB
                call [printf]

                push B
                push spaceStr
                call [scanf]

                push strOp
                call [printf]

                call [getch]
```

```
cmp eax, 43
jne notAdd
    mov ecx, [A]
    add ecx, [B]

    push ecx
    push resStr
    call [printf]

    jmp finish
 notAdd:

 cmp eax, 45
 jne notSub
     mov ecx, [A]
     sub ecx, [B]

     push ecx
     push resStr
     call [printf]

     jmp finish
 notSub:

 cmp eax, 42
 jne notMul
    mov ecx, [A]
    imul ecx, [B]

    push ecx
    push resStr
    call[printf]

    jmp finish
 notMul:

cmp eax, 47
jne notDiv
   mov eax, [A]
   mov ecx, [B]
   mov edx, 0

   cmp[B], 0
   jne notNullDivl
       push infinity
       call[printf]
```

```
        jmp finish
    notNullDiv1:

    div ecx
    mov [C], edx

    push eax
    push resStr
    call [printf]

    push point
    call [printf]

    mov ebx, 0
    lp:

          mov eax, [C]
          mov ecx, [B]
          imul eax, 10

          mov edx, 0
          div ecx
          mov [C], edx

          push eax
          push emptyStr
          call [printf]


    add ebx, 1
    cmp ebx, 3
    jne lp
    jmp finish
notDiv:



 finish:

 call [getch]

 push NULL
 call [ExitProcess]
```

```
section '.idata' import data readable

            library kernel, 'kernel32.dll',\
                    msvcrt, 'msvcrt.dll'

            import kernel,\
                   ExitProcess, 'ExitProcess'

            import msvcrt,\
                   printf, 'printf',\
                   scanf, 'scanf',\
                   getch, '_getch'
```