Jean L. O. Walper

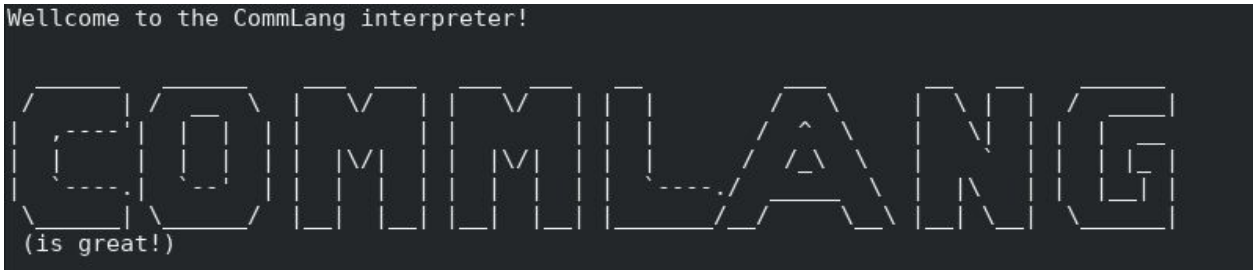
Manual da CommLang, a linguagem de programação comentada.

Ou a linguagem de programação comum, o que soar melhor.

Entrega APS Lógica da Computação.

Insper - 2019

Link do projeto: https://github.com/Lightclawjl/CommLang



Introdução

O nome CommLang vem do termo usado muito em RPGs: “Língua comum” falados pela maioria da população para simplificar as aventuras. Já que a ideia é ser funcional e simples o bastante para qualquer um entender.

A Ideia da APS era criar do zero um Interpretador para uma linguagem de programação idealizada pelo aluno, com foco em melhorar a legibilidade do código. Com isso em mente idealizei uma linguagem para ser usada como pseudo-código em situações onde não é necessário se preocupar com a sintaxe do código demais, situações como entrevistas de programação ou prototipagem e claro, ser usada em situações didáticas para iniciantes no mundo da programação, talvez com jogos e aplicativos Didáticos.

A linguagem foi baseada em python por sua simplicidade. O diferencial é a ideia de que, ao ler essa linguagem em voz alta, seja bem claro o que cada linha está fazendo. Por exemplo, para o código abaixo, não é necessário que o leitor tenha experiência com a linguagem para entender o comando.

```
put 10 divided by 3 in out.
print out.
```

Com isso em mente, algumas escolhas de design foram feitas para a linguagem, para evitar ao máximo o uso de símbolos não pronunciáveis. Um exemplo polêmico é a falta de parênteses entre as expressões, tipagem fraca, ou o uso de pontos “.” e da palavra “Then” para representar escopo.

Por sua natureza de ser para prototipagem, a CommLang copila apenas codigo escrito dentro de comentarios em C. Ou seja, apenas texto contornado por /* *caracteres de escape* */ ou em linhas iniciadas com //. Isso por que pode ser útil para iniciantes criarem um código em CommLang e no mesmo arquivo poderem construir o código em C ou similar.

Também é interessante para esconder fragmentos de código em arquivos .c q podem ser interpretados com o CommLang, e da um trocadilho com o nome, com “Comment Language”.

A CommLang ainda é um protótipo, e está longe de ser perfeita. Mas com o mindset de uma linguagem simples para uso prático e prototipagem, e com a ideia de ser completamente pronunciável, é possível ver usos da CommLang surgindo.

Uso Basico

Abaixo um exemplo de código em CL (CommLang):

```
/*
put input in a.
while a is higher than 0 do
    print a then
    put a minus 1 in a.
*/
```

Alguns pontos importantes, a indentação é opcional (mas recomendada claro), o código está dentro de um comentário, e a indentação de escopo

(normalmente formada por chaves ou palavras de escape) se baseia no uso de ponto final (.) e a palavra then.

Comandos básicos:

Put é o comando mais usado provavelmente, ele possibilita a atribuição de valores a variáveis. Usado como **Put** *Valor* **in** *endereço*.

Input faz a requisição de um input numérico do usuário.

Print imprime a variavel.

Branch:

If e **Else** são simples, if requer um condicional, e no fim do condicional um token **Do**.

Else não precisa de nenhum token extra.

```
/*
put input in a.
if a is equal to 1 do
    print 1 then
    put a plus 1 in a.
else
    print 0.
*/
```

Escopo:

Como dito, o escopo é ditado por uso de ponto e then.

O **If** acima tem duas linhas, o programa sabe disso por conta do token **Then** usado após o comando. Quando é necessário sair, um ponto faz com que o escopo acabe.

O uso de múltiplos níveis de código faz com que sejam necessários alguns pontos adicionais no código para fazer sentido.

```
put false in flag.
put input in coisa.
if flag is not true do
    put coisa in counter then
    while counter is not equal to 0 do
        print counter then
        put counter minus 1 in counter.
    .
.
print 9999.  (fim do codigo)
```

Por exemplo esses pontos no fim da rotina de **while** e de **if** desse código.

Comentarios:

Comentários no código podem ser feito em qualquer lugar com o uso dea parênteses.

Operadores:

Operadores de soma, subtração, divisão e multiplicação são dados dentro das expressões, com: *valor* **plus** *valor*

valor **minus** *valor*

valor **divided by** *valor*

valor **Times** *valor*

Condicionais:

Usamos **and, or, not, higher than, lower than, equals**.

O equals tem uma qualidade especial, ele pode ser escrito de varias formas:

```
put input in boo.
if boo is equal to 10 do
  print 10.
if boo equals 20 do
  print 20.
if boo is 30 do
  print 30.
```

As três formas de escrever a comparação de igualdade são idênticas. A ideia de algo assim é deixar a legibilidade da CommLang melhor, já que variar entre as diferentes formas de expressar essa comparação faz com que os comandos pareçam mais naturais. Como a APS é apenas um protótipo, esse tratamento foi dado apenas ao **Equals**, mas com o tempo as outras expressões podem ter expressões equivalentes também.

E também alguns atalhos para prototipagem, como um comando **Swap** *endereço* **and** *endereço* para fazer uma troca rápida de variáveis, ou **Increment** *value* para substituir o **put** comum.

Esses atalhos são importantes para a natureza da CommLang, mas não foram implementados para o Escopo da entrega.

Funções:

Funções são declaradas com o token **To**, seguidas por um identificador e depois um token **Do** para indicar o início da rotina. É altamente encorajado que os nomes das funções sejam verbos no infinitivo.

```
to succed_in_life do
    print 000000600 then
    print 111000111.

do succed_in_life.
```

Para invocar funções, usa-se o token **Do** no início de um comando.

Para passar argumentos, usa-se a seguinte sintaxe.

```
to sum_and_print ask for a,b then do
    print a plus b.

do sum_and_print with 16, 25.
```

Após a declaração da função, usar os tokens **Ask** e **For** para declarar os argumentos, separados por vírgula. Ao chamar uma função, o token **With** indica a passagem de argumentos, também separados por vírgula.

Para uma função retornar um valor, ela deve atribuir um valor a variavel reservada **Return**. O Token **Do** não é capaz de capturar esse retorno. Para isso é usado o comando **Get**.

```
to sum_and_return ask for a,b then do
    put a plus b in Return.

print(get sum_and_return with 16, 25).
```

Get deve ser usado em expressões como um prefixo, assim como um **Not**.

Recursão:

Com esses features é possível implementar códigos recursivos como o exemplo abaixo. Comparado com um código em Python3.

```
1   def fat(a):                    //to fat ask for a then do
2       if a > 0:                  //  if a is higher than 0 do
3           out = a * rec(a - 1)   //      put a times get fat with a minus 1 in out then
4           #print (a)             //      (print a then)
5           print (out)            //      print out then
6           return out             //      put out in Return.
7       else:                      //  else
8           return 1               //      put 1 in Return.
9                                  //
10  fat(10)                        //do fat with 10.
11
```