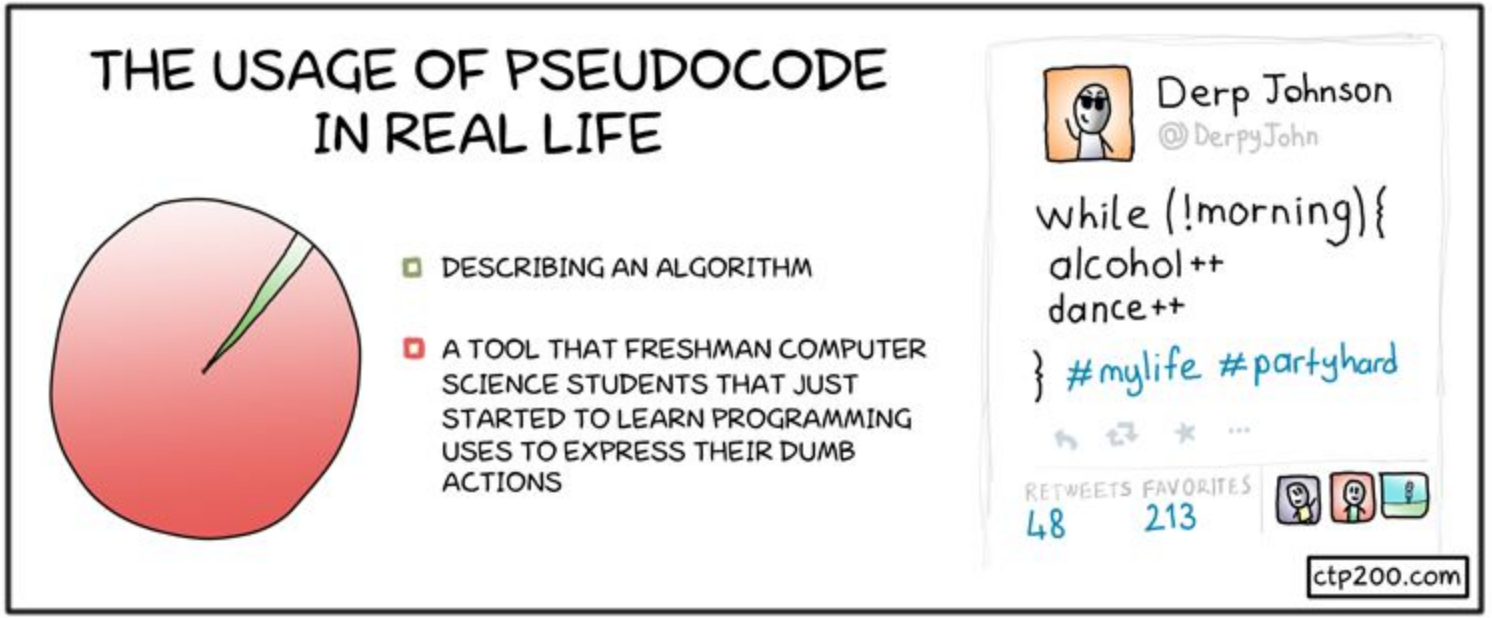
ePseudocode

Barbu Paul - Gheorghe



General belief about pseudocode

Pseudocode on the computer

- Aimed at beginner students
- Offers advanced features, too
- Motivation: I wanted to learn
- Interpreted, duck typed, clean syntax

Technologies

- Haskell
- Parsec, HUnit (Haskell libraries)
- x86 and x64 only

Influenced by C, Python and Haskell

Interpreter and REPL

- $ epseudocode example.epc
- $ epseudocode
  - 1+ 2 // 3
  - scrie("foobar") // foobar
  - daca x == 42 si !ok atunci

    scrie("true")

    altfel

    scrie("false")

    sfdaca
  - func pow(a, b) return a ** b sffunc

FizzBuzz

```
i = 1
cattimp i <= 100 executa
    daca i % 3 == 0 si i % 5 == 0 atunci
        scrie("fizzbuzz\n")
    altfel daca i % 3 == 0 atunci
        scrie("fizz\n")
    altfel daca i % 5 == 0 atunci
        scrie("buzz\n")
    altfel
        scrie(i,"\n")
    sfdaca sfdaca sfdaca
    i = i + 1
sfcattimp
```

```
i = 0
while i <= 100:
        if i % 3 == 0 and i % 5 == 0:
            print("fizzbuzz")
        elif i % 3 == 0:
            print("fizz")
        elif i % 5 == 0:
            print("buzz")
        else:
            print(str(i))
        i += 1
```

Callbacks

```
func applyToRange(a, b, step, f)
    pt i=a; i<=b; i=step(i) executa
        f(i)
    sfpt
sffunc
```

```
void applyToRange(int a, int b, int (*step)(int),
    void (*f)(int))
{
    for(int i=a; i<=b; i=step(i))
    {
        f(i);
    }
}
```

Closures

```
func plusN(n)
    ret func(x)
        ret n + x
    sffunc
sffunc
```

```
def plusN(n):
    def closure(x):
        return n + x
    return closure
```

User defined types - Point

```
struct point
    x = 0
    y = 0

    func translate(dx, dy)
        x = x+dx
        y = y+dy
    sffunc

    move = func(new_x, new_y)
        x = new_x
        y = new_y
    sffunc
sfstruct
```

User defined types - Dictionary

```
struct KeyVal
    key = ""
    val = ""
sfstruct

struct Dict
    _d = {}

    func insert(k, v)
        kv = KeyVal()
        kv.key = k
        kv.val = v
        _d = _d + kv
    sffunc
```

```
    func get(k, default)
        daca lung(_d) == 0 atunci
            ret default
        sfdaca

        pt i=lung(_d)-1; i>=0; i=i-1 executa
            daca _d[i].key == k atunci
                ret _d[i].val
            sfdaca
        sfpt

        ret default
    sffunc
```

```
    func min()
        m = _d[lung(_d)-1]

        pt i=lung(_d)-1; i>=0; i=i-1 executa
            daca get(_d[i].key, m.val) < m.val atunci
                m = _d[i]
            sfdaca
        sfpt

        ret m
    sffunc
sfstruct
```

Future work

- debugger
- optimize the interpreter
- tooling integration (e.g. Notepad++)
- REPL enhancements
- stdlib enrichment

References

- https://en.wikipedia.org/wiki/Haskell_%28programming_language%29
- https://en.wikipedia.org/wiki/C_%28programming_language%29
- https://en.wikipedia.org/wiki/Python_%28programming_language%29
- https://research.microsoft.com/en-us/um/people/daan/download/parsec/parsec.pdf
- https://wiki.haskell.org/Parsing_expressions_and_statements
- https://www.coursera.org/course/compilers
- https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

ePseudocode

Barbu Paul - Gheorghe