Kompilátory - Assignment 1

Peter Schmidt

1 Špecifikácia jazyka

1.1 Syntax

Syntax je podobná jazyku C. Každý riadok obsahuje najviac jeden príkaz (priradenie, volanie funkcie alebo niektorú z riadiacich štruktúr), a príkazy sa neoddeľujú bodkočiarkami. Identifikátory (názvy funkcií a mená premenných) musia spĺňať regulárny výraz `[_a-zA-Z][_a-zA-Z0-9]*`

1.2 Premenné

Jazyk podporuje 32 bitový celočíselný typ int, polia typu int a booleovský typ bool. Jazyk podporuje stringové literály iba pre výstup. Premenné majú statický typ, deklarované sú ako v jazyku C. Neinicializované premenné majú hodnotu 0 resp. `false`.

```
int i, j
int n = 5
```

Premenné sú platné po koniec bloku v ktorom boli definované. Nie je povolené deklarovať premennú s názvom rovnakým ako názov funkcie alebo meno inej platnej premennej.

1.3 Operátory

Jazyk podporuje operátory pre bežné aritmetické a logické operácie

```
+ - * / % & | ^
```

a relácie porovnania

```
< > <= >= == !=
```

1.4 Polia

Jazyk podporuje iba polia typu `int`. Polia sú indexované od 0, veľkosť poľa je možné zistiť vstavanou funkciu `size()`. Veľkosť poľa je definovaná v runtime a pole je zrušené pri návrate z funkcie.

```
int pole[5]
int pole[5][2]
n = pole[i]
```

1.5 Funkcie

Funkcie sú deklarované a definované rovnako ako v jazyku C.

```
int fn(int i, int pole[]) {
    return pole[i]
}
```

Tiež je možné používať externé funkcie, tie je potrebné deklarovať na začiatku zdrojového kódu.

```
extern int fn(int, int)
```

1.6 Vstup a výstup

Pre štandardný vstup a výstup slúžia kľúčové slová read a write. Načítať je možné iba do premennej typu int, pre výpis je možné navyše používať stringové literály.

```
read n                  // ak n je typu int, nacita cele cislo
write "vysledok: ", n   // vypise string a cislo n
```

1.7 Riadiace štruktúry

Jazyk podporuje for a while pre cykly a if a unless pre podmienené vetvenie. Iteračná premenná for-cyklu je vždy typu int, čo nie je potrebné deklarovať. Pre potreby cyklu sa používa range (ako v Ruby), inkluzívny 1..n, resp. exkluzívny 0...size(pole)

```
for i in 0...size(pole) {
    // do something with pole[i]
}

if n < m {
    // ...
}
else {
    // ...
}
```

2 Príklady

2.1 Triedenie

```
void merge(int[] in1, int[] in2, int[] out) {
    int p1 = 0, p2 = 0
    for i in 0...size(out) {
        if in1[p1] <= in2[p2] {
            out[i] = in1[p1]
            ++p1
        }
        else {
```

```
            out[i] = in2[p2]
            ++p2
        }
    }
}

void merge_sort(int[] pole) {
    if size(pole) == 1 {
        return
    }

    int odd = (size(pole) % 2 == 0) ? 0 : 1
    int sub1[size(pole) / 2 + odd]
    int sub2[size(pole) / 2]

    for i in 0...size(sub1) {
        sub1[i] = pole[i]
    }
    for i in 0...size(sub2) {
        sub2[i] = pole[i + size(sub1)]
    }

    merge_sort(sub1)
    merge_sort(sub2)
    merge(pole, sub1, sub2)
}

void main() {
    int n
    read n

    int pole[n]
    for i in 0...n {
        read pole[i]
    }

    merge_sort(pole)

    for i in 0...n {
        write pole[i]
    }
}
```

2.2 Prvočísla

```
void main() {
    int n
    read n

    int pole[n-2]
    for i in 0...size(pole) {
        pole[i] = i+2
    }

    for i in 0...size(pole) {
        unless pole[i] == 0 {
            for j in (i+1)...size(pole) {
```

```
                if pole[j] % i == 0 {
                    pole[j] = 0
                }
            }
        }
    }

    for i in 0...size(pole) {
        unless pole[i] == 0 {
            write pole[i]
        }
    }
}
```

2.3 Súvislosť grafu

```
int findset(int[] pole, int i) {
    int iset = i
    while pole[iset] != iset {
        iset = pole[iset]
    }
    while pole[i] != iset {
        pole[i] = iset
        i = pole[i]
    }
    return iset
}

int union(int[] pole, int i, int j) {
    int iset = findset(pole, i)
    pole[iset] = findset(pole, j)
    return pole[iset]
}

void main() {
    int n, m
    read n, m

    int sets[n]
    for i in 0...n {
        sets[i] = i
    }

    for i in 0...m {
        int u, v
        read u, v
        union(sets, u, v)
    }

    bool connected = true
    for i in 1...n {
        unless sets[i-1] == sets[i] {
            connected = false
        }
    }

    if connected {
```

```
        write "ANO\n"
    }
    else {
        write "NIE\n"
    }
}
```