Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada

Disciplina: DIM0437 — Linguagens de Programação: Conceitos e Paradigmas
Docente: Umberto Souza da Costa
Discentes: Dogival Ferreira da Silva Junior
Felipe Cortez de Sá
Gabriel Sebastian von Conta
Phellipe Albert Volkmer
Vinícius Araújo Petch

Subproblema 7 — Subprogramas

1 Problema

1.1 Produto do problema

Definição da sintaxe e semântica intuitiva dos mecanismos que regem as formas de abstração de processamento (procedimentos e funções) da linguagem de programação a ser definida pelo grupo. Incluir a representação destas abstrações, assim como os mecanismos de passagem de parâmetros e a implementação sugerida.

1.2 Questões

1. Como serão definidos os procedimentos e funções de sua linguagem? Note que estes conceitos são diferentes, embora sejam tratados de forma unificada por algumas linguagens de programação. Quais são as diferenças entre esses conceitos?
2. Como serão definidos os parâmetros da linguagem? Quais dados poderão ser colocados como argumentos em chamadas a procedimentos e funções? Nomes de subprogramas poderão ser utilizados como parâmetros?
3. Quais serão as formas de passagem de parâmetros e como serão implementadas?
4. Sua linguagem verificará os tipos de parâmetros dos subprogramas?
5. A linguagem terá subprogramas sobrecarregados ou genéricos?
6. A linguagem deverá ter compilação separada ou independente?
7. Sua linguagem dará suporte a co-rotinas?

2 Resoluções

1. Na nossa linguagem daremos suporte tanto a funções quanto procedimentos, procurando explicitar sua diferença através da sintaxe com o objetivo de deixar claros ambos os conceitos facilitando o aprendizado dos nossos usuários.

Tanto funções quanto procedimentos são sub-rotinas, ou seja, sequências de comandos reutilizáveis que podem ser alteradas sem a necessidade de sua repetição no código-fonte. A diferença conceitual entre procedimento e função reside na presença de variáveis de retorno para funções. Os procedimentos, por outro lado, são úteis apenas quando geram efeitos colaterais.

Em C, por exemplo, procedimentos têm a mesma sintaxe de funções, com `void` no lugar do tipo:

```
int funcao(int a, int b) {
    return a + b;
}

void procedimento() {
    puts("Este comando gera um efeito colateral");
}
```

A sintaxe da nossa linguagem, portanto, estará na seguinte forma

```
// A sintaxe vai aqui
```

2. Os parâmetros em nossa linguagem poderão ser de qualquer tipo da linguagem, inclusive os tipos criados pelo usuário. A utilização de parâmetros será de forma posicional, uma vez que, como a maior parte dos programas será curto, não são necessários keyword parameters, que diminuiriam a capacidade de escrita do nosso código e o tornaria muito extenso. Subprogramas não poderão ser utilizados como argumentos, uma vez que acarretaria em uma queda de legibilidade para o programa. Além disso, novamente, como o caráter dos subprogramas é simples, pode-se facilmente atribuir o valor de um subprograma a uma variável e então usar essa variável como argumento.
3. Nós teremos passagem por valor, implementado por cópia, e passagem por referência.
4. Sim, ela utilizará o método de protótipo.
5. Não teremos subprogramas genéricos, mas teremos subprogramas com sobrecarga.
6. (Compilação separada: as unidades de compilação podem ser compiladas em tempos diferentes, mas elas não são independentes uma da outra se qualquer uma delas acessar ou usar quaisquer entidades da outra. Tal interdependência é necessária se precisar ser feita verificação de interface.)

   (Compilação independente: unidades de programa podem ser compiladas sem informações sobre quaisquer outras unidades de programa. Unidades compiladas separadamente não são verificadas quanto à coerência de tipos)

   (Algumas linguagens não oferecem nem compilação separada, nem compilação independente, significando que somente a unidade de compilação é um programa completo. Isso a torna virtualmente inútil para aplicações industriais)

   Nossa linguagem terá compilação separada, pois essa compilação é bastante prática já que caso ocorra uma alteração no código, nem sempre será necessário compilar todo o código. A escolha da compilação separada ao invés da compilação independente se dá na falta de verificação de coerência de tipo desta.

7. Uma co-rotina é um tipo especial de subprograma. Ao invés de possuir uma relação mestre-escravo entre o subprograma que chama e o subprograma chamada, ambos estão em uma relação mais justa.

   Apesar de co-rotinas serem uma funcionalidade relevante à uma linguagem, como a nossa linguagem possui um escopo educativo elas não serão utilizadas, e portanto desnecessárias.