Primeira Prova de Linguagens de Programação
- DCC024B -
Sistemas de Informação

Nome: _____
"Eu dou minha palavra de honra que não trapacearei neste exame."

Número de matrícula: _____

As regras do jogo:

- A prova é sem consulta.
- Quando terminar, não entregue nada além do caderno de provas para o instrutor.
- Quando escrever código, a sintaxe correta é importante.
- Cada estudante tem direito a fazer uma pergunta ao instrutor durante a prova. Traga o caderno de provas quando vier à mesa do instrutor.
- A prova termina uma hora e quarenta minutos após seu início.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

Alguns conselhos:

- Escreva sempre algo nas questões, a fim de ganhar algum crédito parcial.
- Se não entender a questão, e já tiver gasto sua pergunta, escreva a sua interpretação da questão junto à resposta.
- A prova não é difícil, ela é divertida, então aproveite!

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5

Questão Extra (0.5 Pontos): qual é a propriedade invariante que se aplica a qualquer ponto de uma parábola?

1. O programa `gcc`, usado em Unix para invocar o compilador de C desenvolvido pela gnu, é na verdade um *script* que invoca vários outros programas. É possível saber quais os programas são invocados por `gcc` adicionando-se o parâmetro `-v` à sua linha de invocação. Cada uma das questões abaixo contém um dos passos adotados por `gcc` para produzir um arquivo binário a partir de um programa fonte `hello.c`, no sistema Linux. Descreva o que cada uma destas linhas faz.

   (a) `gcc -E hello.c > hello.p.c`

   (b) `gcc -S hello.p.c -o hello.p.s`

   (c) `as hello.p.s -o hello.o`

   (d) `/usr/bin/collect2 hello.o -o a.out`

2. Considere a gramática lógica abaixo, que reconhece cadeias de zero ou mais ocorrências dos símbolos `+` e `-`:

```
mp --> [+], mp.
mp --> [-], mp.
mp --> [].
```

   (a) (2 Pontos) Essa gramática não é ambígua. Escreva, usando a notação BNF, uma gramática que reconheça a mesma linguagem, mas que seja ambígua.

   (b) (2 Pontos) Demonstre que a gramática que você criou na questão anterior é, de fato, ambígua.

   (c) (6 Pontos) Modifique a gramática lógica mostrada no início desta questão, adicionando-lhe um atributo `D`. Esse atributo deve informar a diferença entre símbolos `+` e símbolos `-`. Por exemplo:

```
?- mp(D, [], []).
D = 0.

?- mp(D, [-, +, +], []).
D = 1 ;
false.

?- mp(D, [-, -, +], []).
D = -1 ;
false.
```

3. (10 Pontos) Escreva uma função `toDig`, de tipo `string -> string list`, que converta uma *string* em uma lista de dígitos. Por exemplo:

```
- toDig "123";
val it = ["um","dois","tres"] : string list
- toDig "";
val it = [] : string list
- toDig "001001";
val it = ["zero","zero","um","zero","zero","um"] : string list
```

Essa função deve ser capaz de lidar com qualquer *string* de entrada. Se ela receber uma *string* contendo caracteres que não são números, então ela deve imprimir `???` para cada um desses caracteres. Por exemplo:

```
- toDig "A23";
val it = ["???","dois","tres"] : string list
```

É possível que você queira usar a função `ord`, de tipo `char -> int` em sua solução. Essa função converte um caracter para o seu código ASCII. Por exemplo:

```
- ord #"0";
val it = 48 : int
- ord #"1";
val it = 49 : int
- ord #"2";
val it = 50 : int
```

Outra função que você pode achar interessante é `explode`, de tipo `string -> char list`, que recebe uma *string* e a converte para uma lista de caracteres:

```
- explode "dcc024";
val it = [#"d",#"c",#"c",#"0",#"2",#"4"] : char list
```

4. O *Link de Aninhamento* é uma estrutura usada para permitir que uma função aninhada possa fazer referência a variáveis declaradas no corpo da função aninhadora. Como um exemplo, considere o programa abaixo, implementado em **Gnu C**:

```
int sum(int x, int limit) {
  int prod(int begin) {
    if (begin >= limit)
      return 1;
    else
      return begin * prod(begin + 1);
  }
  return prod(x);
}
int main(int argc, char** argv) {
  printf("sum(2, 1) = %d\n", sum(2, 1));
}
```

A função `prod` precisa utilizar um link de aninhamento para encontrar o valor da variável `limit`, pois essa variável não é local a `prod`.

(a) Em cada uma das situações abaixo, determine como é descoberto o endereço para onde aponta o link de aninhamento da função g, logo que ela é ativada:

i. (2 Pontos) A função g não está aninhada dentro de alguma outra função.

ii. (2 Pontos) g está aninhada dentro de uma função f, e g é chamada pela primeira vez.

iii. (2 Pontos) A função g está aninhada dentro de uma função f, e g é chamada recursivamente.

(b) (4 Pontos) O link de aninhamento é necessário em linguagens de programação que permitem a declaração de funções aninhadas. Por outro lado, mesmo linguagens que não possuem esse recurso, como **ANSI C**, ainda permitem que variáveis livres existam. Isso ocorre devido às variáveis globais:

```
int counter;
void incCounter() {
  counter++; // counter eh variavel livre no escopo de incCounter
}
int main() {
  counter = 0;
  incCounter();
  printf("Counter = %d\n", counter);
}
```

Por que não é necessário um link de aninhamento para encontrar-se o valor de variáveis globais?

5. (2.5 pontos cada) Uma linguagem é estaticamente tipada quando o tipo de cada expressão pode ser resolvido em tempo de compilação. Uma linguagem é dinamicamente tipada quando o tipo da variável é resolvido em tempo de execução.

   (a) Dê um exemplo de uma linguagem estaticamente tipada. Como o compilador consegue descobrir o tipo das variáveis, no caso desta linguagem?

   (b) Dê um exemplo de uma linguagem dinamicamente tipada. Escreva um programa, muito simples, que evidencie o caráter dinâmico desta linguagem.

   (c) Cite uma vantagem da tipagem estática sobre a tipagem dinâmica.

   (d) Agora, cite uma vantagem da tipagem dinâmica sobre a tipagem estática.