Licenciatura em Engenharia Informática (FCT/UNL)
Ano Lectivo 2007/2008
Linguagens e Ambientes Programação – Época Normal

02 de Julho de 2008 às 17:00

Exame com consulta com 2 horas e 45 minutos de duração + 15 minutos de tolerância

Nome: Num:

Notas: *Nos problemas em ML mostre que sabe usar o método indutivo.*
Pode definir funções/métodos auxiliares sempre que precisar (muitas vezes é mesmo preciso)
Responda no próprio enunciado.
Fraude implica reprovação na cadeira.

1. [2 valores] Para cada frase, escreva na zona sublinhada se ela é VERDADEIRA ou FALSA:

- Se o verificador de tipos do C++ rejeitar um programa, isso significa que se o programa fosse executado, mesmo sendo inválido, se obteria um erro de execução. _____
- Determinados problemas particularmente complicados não podem ser resolvidos usando apenas a parte funcional do ML. Isso deve-se ao facto de não se poderem usar variáveis mutáveis. _____
- Em ML e em C++, é possível passar apenas parte dos argumentos para as funções. _____
- Ao contrário do C++, em ML a avaliação das funções que não terminam produz o valor especial *nil*. _____
- Dois parâmetros inteiros passados por valor para uma função em C++, no interior da função nunca podem referir a mesma localização de memória. _____
- Dois parâmetros inteiros passados por referência para uma função em C++, no interior da função nunca podem referir a mesma localização de memória. _____
- O algoritmo de inferência de tipos do ML consegue determinar o tipo de expressões que omitam declarações de tipo para as suas variáveis. _____
- Um sistema de tipos dinâmico tem a vantagem de manter uma linguagem flexível sem invalidar programas desnecessariamente. Contudo tem duas desvantagens: a perda da detecção precoce de erros e a perda da parte documental das anotações de tipos (quando a linguagem é explicitamente tipificada). _____
- Num compilador, a verificação de tipos vem antes da análise sintáctica pois isso evita que se perca tempo a fazer a análise sintáctica de expressões erradas. _____
- Ao contrário do que se passa com os programas interpretados, para correr um programa previamente compilado é preciso que o compilador esteja disponível na mesma máquina, assim como as bibliotecas dinâmicas de que o programa dependa. _____

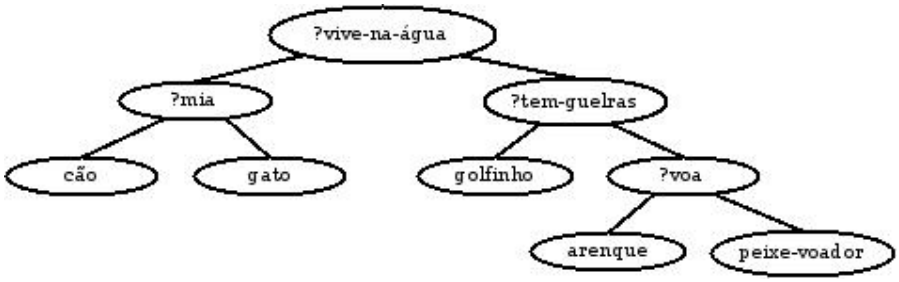
2. [2 valores] Diga qual o tipo da seguinte função em ML:

```
let f g x = g (fst x) ;;
```

[Ajuda: O tipo da função `fst` é `'a * 'b -> 'a` .]

3. **Árvores de decisão binárias em ML**. O conceito de árvore de decisão binária já é bem conhecido do projecto 1 de LAP. Uma árvore de decisão binária é constituída por **questões** (nós internos binários) e por **entidades** (as folhas da árvore). A árvore de decisão vazia não está prevista. Nesta pergunta vamos assumir que todas as funções se aplicam a árvores consistentes, ou seja, a árvores sem dados contraditórios.

Eis um exemplo duma árvore de decisão binária que representa algum conhecimento sobre animais:



O tipo das árvores de decisão binárias pode definir-se assim em ML:

```
type dtree = Entity of string | Question of string * dtree * dtree ;;
```

Resolva os seguintes três problemas sobre árvores de decisão. Tratam-se de problemas não relacionados, ou seja, nenhum deles ajuda a resolver os restantes.

a) [2 valores] Escreva em OCaml uma função

```
prefix: dtree -> string list
```

que, dada uma árvore de decisão binária, produza uma lista com todas as strings contidas na árvore, pela ordem prefixa. Repare que essa lista pode ser considerada uma representação alternativa da árvore de decisão. Exemplos:

```
prefix (Entity "gato") = ["gato"]
prefix (Question "?mia" (Entity cão) (Entity gato)) = ["?mia";"cão";"gato"]
```

b) [2.5 valores] Escreva em OCaml uma função

```
xiferp: string list -> dtree
```

que, dada uma lista de strings representando sob a forma prefixa uma árvore de decisão, reconstitua a árvore de decisão original. A função pedida é a função inversa da função prefix, da alínea anterior. Assuma que a lista-argumento não tem erros. Exemplos:

```
xiferp ["gato"] = (Entity "gato")
xiferp ["?mia";"cão";"gato"] = (Question "?mia" (Entity cão) (Entity gato))
```

[Ajuda: A seguinte função permite testar se uma string é uma questão:

```
let isQuestion s =
    String.length s > 0 && String.get s 0 = '?'
;;
```

]

c) [2.5 valores] Escreva em OCaml uma função

```
play: dtree -> dtree
```

que estabeleça um diálogo com o utilizador com vista a jogar um conhecido jogo de IA. A ideia é a seguinte: o utilizador escolhe um nome dum animal e a função `play` tenta descobrir qual é esse animal com base na árvore de decisão que lhe foi passada como argumento.

A função começa por colocar várias questões binárias ao utilizador, podendo este responder "não" ou "sim" a cada uma delas. Quando a função já não quiser fazer mais perguntas, ela dá um único palpite, o qual poderá estar correcto ou incorrecto.

No caso do palpite estar correcto, a função `play` retorna a árvore de entrada inalterada. Exemplo de diálogo com palpite correcto:

```
?vive na água  não
?mia  não
Está a pensar num cão? sim
Viva!! Acertei!!!
```

No caso do palpite estar incorrecto, a função tenta aprender com o seu erro. Nesse caso pergunta ao utilizador qual era o animal correcto e pede também uma questão binária que permita distinguir entre o animal certo e o palpite errado. Finalmente devolve a árvore de decisão enriquecida com a nova informação. Exemplo de diálogo com palpite incorrecto:

```
?vive na água  não
?mia  não
Está a pensar num cão? não
Desisto. Em que animal estava a pensar?  águia
Por favor, ensine-me uma pergunta cuja resposta seja SIM para águia e NÃO para cão: ?voa
Obrigado.
```

Assuma que o utilizador colabora e não tenta fazer batota.

[Ajudas: As escritas podem ser feitas usando a função `print_string` e as leituras usando a função `read_line`. Usando o método indutivo, é possível escrever a função `play` de forma compacta e elegante sem ser necessário usar qualquer função auxiliar; no entanto você tem a liberdade de definir funções auxiliares que quiser.]

4. Considere o seguinte programa escrito em GCC, uma variante do C que suporta aninhamento de funções:

```
int i ;

void X(int *a, int b) {
    void Y(int a, int b) {
        void Z(int a, int b) {
            X(&b, a) ;
        }
        Z(a, a+b) ;
    }
    Y(++*a, b) ;
}

int main(void) {
    i = 3 ;
    X(&i, i) ;
    return 0 ;
}
```

a) [0.5 valores] A linguagem C usa escopo estático. Mas será que a utilização de escopo dinâmico alteraria o comportamento deste programa particular? Responda, justificando de forma simples.

b) [1.5 valores] Mostre qual o estado da pilha de execução no momento em que é empilhado segundo registo de activação da função X.

Para efeitos da criação do registo de activação inicial, imagine que cada programa em GCC está embebido numa função sem argumentos chamada `start`. Depois trate todas as entidades globais do programa como sendo locais à função imaginária `start`. Assuma também que a primeira célula da pilha de execução é identificada como posição 0, a segunda célula da pilha de execução é identificada como posição 1, etc. Estas são as mesmas convenções que foram usadas nas aulas teóricas e práticas.

35	23	11
34	22	10
33	21	09
32	20	08
31	19	07
30	18	06
29	17	05
28	16	04
27	15	03
26	14	02
25	13	01
24	12	00

5. A forma mais natural de implementar um vector em C++ é usando um array primitivo uni-dimensional. Contudo há situações em que quase todos os valores guardados num vector são zeros e, sendo assim, bastaria tomar nota dos valores diferentes de zero, para poupar memória. Chama-se **vector esparso** a um vector no qual a grande maioria dos valores é igual a zero.

O objectivo deste problema é desenvolver um **sistema de classes** que permita ao utilizador representar vectores normais e vectores esparsos, estes últimos com uma implementação interna especial que poupa memória. Para se obter flexibilidade, deve ser possível usar os dois tipos de vector de forma intermutável, tirando partido do polimorfismo da linguagem C++.

a) [5 valores] O utilizador do sistema de classes deve ter a liberdade de declarar variáveis dum tipo geral `Vector*` e através delas manipular objectos de duas classes chamadas `NormalVector` e `SparseVector`, como no seguinte exemplo de utilização:

```
void Proc() {
    Vector *v1 = new NormalVector(100) ;
    Vector *v2 = new SparseVector(100) ;
    for(int i = 0 ; i < 20 ; i++ ) {
        v1->Set(i, i*2.1) ;
        v2->Set(i, i*3.123) ;
    }
    …
}
```

Nesta alínea, pedimos-lhe para programar integralmente as classes `Vector`, `NormalVector` e `SparseVector`. Vejamos os detalhes:

Os vectores armazenam valores de tipo `double`. Os índices começam em zero, como de costume. Os construtores têm um único argumento inteiro que indica a capacidade dos vectores a criar. As funções públicas, a definir, são as seguintes três:

`double Get(int i)` – Retorna o valor guardado na posição i. É lançada uma excepção se o índice i estiver fora dos limites.
`void Set(int i, double v)` – Guarda o valor v na posição i. É lançada uma excepção se o índice i estiver fora dos limites (no caso de `NormalVector`) ou se o valor v já não couber no vector (no caso de `SparseVector`).
`int Find(double v)` – Produz o índice do valor procurado no vector. Retorna -1 se o valor v não ocorrer no vector.

A classe `NormalVector` é implementada usando um array primitivo uni-dimensional de reais, onde se guardam os valores. Nesta classe os índices têm um limite superior determinado pela capacidade indicada no construtor.

A classe `SparseVector` é implementada usando dois arrays paralelos com o mesmo comprimento: um array primitivo uni-dimensional de inteiros onde se guardam os índices, mais um array primitivo uni-dimensional de reais onde se guardam os valores. Os índices não têm limite superior nesta classe, apesar da capacidade de armazenamento ser limitada.

Escreva código simples e correcto e não se preocupe muito com a velocidade de execução.

b) [2 valores] Adicione ao seu sistema dos vectores uma nova classe concreta de **vectores inteligentes** chamada `CleverVector`, na qual a representação interna se ajusta automaticamente aos dados: se a grande maioria dos valores guardados forem zeros, usa-se a representação esparsa; caso contrário usa-se a representação normal; além disso deve poder transitar-se entre as duas representações de forma automática. Dentro de cada vector inteligente basta guardar um SparseVector ou um NormalVector, consoante o que for conveniente. Também deve ser possível manipular vectores inteligentes através de variáveis de tipo `Vector*`.

Quais são os limiares para mudança de representação dentro dos vectores inteligentes? Se a representação corrente for esparsa, a representação muda para normal se o número de não-zeros subir até aos 50%. Se a representação corrente for normal, a representação muda para esparsa se o número de não-zeros baixar até aos 25%. Para tomar as suas decisões, a classe `CleverVector` tem arranjar alguma forma de saber o número de não-zeros guardados; não se recomenda a adição de mais funções às classes da alínea anterior pois há pelo menos uma boa solução que respeita a integralidade das classes existentes.

Nesta classe, os índices dos vectores inteligentes têm as mesmas restrições dos vectores normais.

Nesta alínea, pedimos-lhe para programar integralmente a classe `CleverVector`.

[Agora que o exame já terminou, uma observação:

A restrição indicada na penúltima linha da alínea b) torna a classe `CleverVector` bastante menos útil do que poderia ser, embora mais fácil de implementar. Para que a classe `CleverVector` se tornasse verdadeiramente útil, ela deveria ser sujeita apenas às restrições da classe `SparseVector` e ser mais frequentemente implementada usando essa classe. A transição interna para a classe `NormalVector` só deveria acontecer quando se verificassem as duas seguintes condições: (1) o número de não-zeros sobe até aos 50%; (2) todos os índices do vector esparso se situam no intervalo de inteiros aberto à direita [0, capacidade[.]