Arquitetura de Computadores 2018/19

TPC 1

Prazo de entrega: 23:59 de 5 de abril de 2019

Este trabalho de casa consiste em dois exercícios de programação **individual**. Pode esclarecer dúvidas gerais com colegas, mas a solução e a escrita do código deve ser estritamente individual. Todas as resoluções serão comparadas de forma automática e os casos de plágio serão punidos de acordo com os regulamentos em vigor.

A entrega será via Mooshak do DI (`http://mooshak.di.fct.unl.pt/~mooshak/`). Os programas serão compilados com o seguinte comando:

```
cc -m32 -Wall -std=c11 -o prog prog.c
```

Exercício 1

Pretende-se implementar a soma de números inteiros (>=0) de precisão arbitrária (i.e., com qualquer número de algarismos) sendo que cada número é representado por uma string contendo os caracteres dos algarismos e apenas algarismos. Exemplo: o número 3423 será representado na string de texto "3423" ou seja, um array de caracteres contendo '3', '4', '2', '3' e '\0'. Este exercício deve ser resolvido em dois passos, que deverão ser submetidos separadamente ao Mooshak.

a) [Prob A, 20%] Comece por implementar a função seguinte que soma dois algarismos (representados pelo respetivo caracter) e um eventual excesso (carry) anterior:

```
int addIntChar( char n1, char n2, int lastcarry, char *res );
```

o seu efeito será equivalente à avaliação de *res = n1+n2+lastcarry*, mas ficando em *res* só a unidade e devolvendo uma nova *carry*. Exemplo de utilização:

```
int c = 1;
char a = '8';
char b = '5';
char res;
c = addIntChar( a, b, c, &res );
```

terá como resultado: res=='4' e c==1 (já que 8+5+1=14).

b) [Prob B, 40%] Fazendo uso da função anterior, implemente agora a função seguinte que é capaz de somar dois números representados em strings, sendo o resultado colocado no *array* res, também como uma string.

```
void addIntStrings( char *n1, char *n2, char *res );
```

o seu efeito será equivalente à avaliação de *res = n1 + n2*. Admita sempre que o array res passado à função tem a dimensão suficiente para guardar todos os caracteres do resultado. Exemplo de utilização:

```
char *a = "35";
char *b = "91";
char res[100];
addIntStrings( a, b, res );
```

terá como resultado res = "126". Pode testar a sua função com o seguinte programa:

```
int main() {
  char a[100]; char b[100];
  char res[100];
  scanf("%s", a); getchar();
  scanf("%s", b); getchar();
  addIntStrings( a, b, res );
  printf("%s\n", res);
  return 0;
}
```

Nota: não se esqueça de garantir que a *string* resultado começa na posição 0 do *array* `res`. Pode construir a *string* resultado em qualquer posição do *array* e, no fim, mover esta para o início do mesmo usando a função `memmove`. Por exemplo, se a *string* começar na posição 5, pode movê-la para o início usando:

```
memmove( res, res+5, strlen(res+5)+1);
```

Exercício 2

[Prob C, 40%] Implemente em C uma função que efetue a multiplicação por 2 de números do tipo float. No código desta função (e das funções que esta possa chamar) não pode utilizar o operador de multiplicação nem funções de bibliotecas. Para resolver este exercício deve recordar a forma como um número com vírgula flutuante é representado na norma IEEE precisão simples. Garanta que compreende qual é o efeito na representação ao multiplicar o valor por dois. Lembre-se também das operações sobre bits.

A sua função deve ter o seguinte protótipo:

```
float mult2( float val );
```

Exemplo de utilização:

```
float g = mult2( 2.5 );
```

em g ficará o valor 5.0. Pode testar a sua função usando o programa seguinte:

```
int main( ) {
  float f;

  scanf( "%f", &f );
  float r = mult2( f );
  printf( "%f\n", r);
  return 0;
}
```

Em C, para obter numa variável `int` os bits que representam determinado `float` (e *vice versa*), para assim poder usar os operadores de bits, recorra a *casts* e *pointers* como nos seguintes exemplos:

```
float f = 2.5;
int i = *(int*)&f;       // obter em i os bits de f
float g = *(float*)&i;   // obter em g os bits de i (logo, g==f)
```