AULA PRÁTICA N.º 9

Objectivos:

Implementação de subrotinas recursivas.

Guião:

Neste guião vão ser apresentados vários exemplos de funções recursivas, tendo algumas delas sido já implementadas, na sua forma iterativa, em guiões anteriores. Note que, na maioria dessas funções, a solução recursiva não é a melhor em termos de eficiência. Essas funções são aqui usadas como exemplos por serem de fácil compreensão, o que permite praticar quer o conceito de recursividade quer a tradução de linguagem C para *assembly* desse tipo de funções.

1. A função seguinte apresenta a implementação, na forma recursiva, da função de contagem do número de caracteres de uma *string* já implementada anteriormente na forma iterativa.

```
int strlen(char *s)
{
    if(*s != '\0')
        return(1 + strlen(s + 1));
    else
        return 0;
}
```

a) Escreva, em linguagem C, a função **`main()`** com código que permita efectuar o teste da função **`strlen()`**.

b) Traduza as funções **`strlen()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

2. A função seguinte apresenta a implementação, na forma recursiva, da função de cópia de uma *string* (também já implementada anteriormente na forma iterativa).

```
char *strcpy(char *dst, char *src)
{
    if((*dst = *src) != '\0')
        strcpy(dst + 1, src + 1);
    return dst;
}
```

a) Escreva, em linguagem C, a função **`main()`** com código que permita efectuar o teste da função **`strcpy()`**.

b) Traduza as funções **`strcpy()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

3. A função seguinte obtém a soma dos elementos de um *array* de inteiros.

```
int soma(int *array, int nelem)
{
    if(nelem != 0)
        return *array + soma(array + 1, nelem - 1);
    else
        return 0;
}
```

~~a)~~ Traduza a função **soma()** para *assembly*.

~~b)~~ Escreva, em linguagem C, a função **main()** com código que permita efectuar o teste da função que escreveu na alínea a). Traduza esse programa para *assembly* e teste o funcionamento da função **soma()**.

~~4.~~ A função seguinte imprime no ecrã o valor inteiro "num" em qualquer base entre 2 e 16 (note que essa verificação não é efectuada no código). Esta função é uma implementação recursiva mais simples e elegante que a solução iterativa já apresentada no guião da aula 8 (que usava, recorde-se, a função **itoa()** para efectuar a conversão entre bases).

```
void print_int_ac1(unsigned int num, unsigned int base)
{
    if(num / base)
        print_int_ac1( num / base, base );
    print_char( toascii(num % base) );
}

char toascii(char v)
{
    v += '0';
    if( v > '9' )
        v += 7;  // 'A' - '9' - 1
    return v;
}
```

~~a)~~ Escreva, em linguagem C, a função **main()** com código que permita efectuar o teste da função **print_int_ac1()**.

~~b)~~ Traduza as funções **print_int_ac1()** e **main()** para *assembly*. Teste o funcionamento do conjunto.

~~5.~~ A função seguinte apresenta uma implementação iterativa de uma função de cálculo do factorial.

```
// Calculo do factorial de n - algoritmo iterativo

unsigned int fact_i(unsigned int n)
{
    unsigned int i;
    unsigned int res;

    for(res = 1, i=2; i <= n; i++)
        res = res * i;
    return res;
}
```

A mesma função escrita na forma recursiva poderá ter a seguinte implementação:

```
// Calculo do factorial de n - algoritmo recursivo

unsigned int fact(unsigned int n)
{
    if(n > 12)
        exit(1); // Overflow

    return (n > 1) ? n * fact(n-1) : 1;
}
```

~~a)~~ Escreva, em linguagem C, a função **`main()`** com código que permita efectuar o teste da função **`fact()`** (implementação recursiva).

~~b)~~ Traduza as funções **`fact()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

~~6.~~ A função seguinte apresenta uma implementação iterativa de uma função de cálculo do valor de x^y.

```
// Cálculo de xʸ – algoritmo iterativo

int xtoy_i(int x, unsigned int y)
{
    int i, result = 1;
    for(i=0; i < y; i++)
        result *= x;
    return result;
}
```

A mesma função escrita na forma recursiva poderá ter a seguinte implementação:

```
// Cálculo de xʸ – algoritmo recursivo

int xtoy(int x, unsigned int y)
{
    return (y != 0) ? x * xtoy(x, y-1) : 1;
}
```

~~a)~~ Escreva, em linguagem C, a função **`main()`** com código que permita efectuar o teste da função **`xtoy()`** (implementação recursiva).

~~b)~~ Traduza as funções **`xtoy()`** e **`main()`** para *assembly*. Teste o funcionamento do conjunto.

~~7.~~ A função seguinte apresenta a implementação recursiva da resolução do problema das torres de Hanoi. Apresenta-se também a implementação da função **`main()`**, bem como de uma função auxiliar para a impressão de uma mensagem.

```
void tohanoi(int n, int p1, int p2, int p3)
{
    static int count=0;

    if(n != 1)
    {
        tohanoi(n-1, p1, p3, p2);
        print_msg(p1, p2, ++count);
        tohanoi(n-1, p3, p2, p1);
    }
    else
        print_msg(p1, p2, ++count);
}
```

```
void print_msg(int t1, int t2, int cnt)
{
   print_str("\n");
   print_int_ac1(cnt, 10);
   print_str(" - Mover disco de topo de ");
   print_int_ac1(t1, 10);
   print_str(" para ");
   print_int_ac1(t2, 10);
}

int main(void)
{
   int ndiscs;

   print_str("\nIntroduza o numero de discos: ");
   ndiscs = read_int();
   if(ndiscs > 0)
       tohanoi(ndiscs, 1, 3, 2);

   return 0;
}
```

a) Traduza as funções anteriores para *assembly*. Teste o funcionamento do conjunto (pode encontrar um simulador deste problema em http://www.mazeworks.com/hanoi/).