Métodos e Técnicas de Programação
.:: Avaliação continuada P4 ::.

Prof. Igor Peretta

ENTREGA: até 07/out/2018

Contents

1 Introdução

1.1 Declaração de funções

Dentro da perspectiva de sequência de instruções na computação, um dos conceitos mais importantes é o da definição prévia do que se vai utilizar. Em linguagens fortemente tipadas, como no caso do C, uma variável precisa ser instânciada (definida e, portanto, com espaço reservado de memória) antes de seu uso. O mesmo vale para funções.

Um exemplo trivial dentro de C é o uso das funções `printf()`, `scanf()` etc. Elas são definidas dentro da biblioteca `stdio.h`, ou seja, a biblioteca deve ser inserida antes do uso.

Para funções definidas pelo usuário, é comum quando se começa a programar que as mesmas estejam definidas antes da função `main()`, como por exemplo:

```
#include <stdio.h>
int soma(int x, int y) {
    return x+y;
}
int main() {
    int a = 3, b = 4;
    printf("%d + %d = %d\n", a, b, soma(a,b));
    return 0;
}
```

Mas as boas práticas de programação nos indica uma outra estratégia:

```
#include <stdio.h>
int soma(int, int);
int main() {
    int a = 3, b = 4;
    printf("%d + %d = %d\n", a, b, soma(a,b));
    return 0;
}
int soma(int x, int y) {
    return x+y;
}
```

Como pode ser visto no exemplo, a linguagem C permite que se faça apenas a descrição de uma função antes de `main()`, que inclui o tipo de retorno, o nome e os tipos dos argumentos da função desejada. Já a própria função pode ser definida em qualquer lugar, mesmo que depois de `main()`. Esse é o princípio das bibliotecas, objeto de aulas futuras. O tipo de prática aqui descrito permite com que duas funções possam utilizar uma na outra, como no exemplo:

```
#include <stdio.h>
int soma(int, int);
int multiplica(int, int);
int main() {
    int a = 3, b = 4;
    printf("%d + %d = %d\n", a, b,
soma(a,b));
    printf("%d * %d = %d\n", a, b,
multiplica(a,b));
```

```
    return 0;
}
int soma(int x, int y) {
    return (x == y) ? multiplica(x, 2) : x + y;
}
int multiplica(int x, int y) {
    int i, res = 0;
    if(y == 2) res = 2*x;
    else
for(i = 0; i < y; i++)
    res = soma(res, x);
    return res;
}
```

Isso é possível porque a cada menção a uma das funções, o compilador C já sabe a descrição da mesma (conhece o endereço de memória da função), mesmo que a definição formal de sua sequência de instruções ainda não tenha sido implementada da primeira vez que percorre seu código.

1.2 Recursão

Fonte: `https://panda.ime.usp.br/pensepy/static/pensepy/12-Recursao/recursionsimple-ptbr.html`

Recursão é um método de resolução de problemas que envolve quebrar um problema em subproblemas menores e menores até chegar a um problema pequeno o suficiente para que ele possa ser resolvido trivialmente. Normalmente recursão envolve uma função que chama a si mesma. Embora possa não parecer muito, a recursão nos permite escrever soluções elegantes para problemas que, de outra forma, podem ser muito difíceis de programar.

Imaginem por um minuto que você não tem laços while ou for. Como você calcularia a soma de uma lista de números? Se você fosse um matemático poderia começar recordando que a adição é uma função definida para dois parâmetros, um par de números. Para redefinir o problema da adição de uma lista para a adição de pares de números, podemos reescrever a lista como uma expressão totalmente entre parênteses. Tal expressão poderia ser algo como:

$$((((1 + 3) + 5) + 7) + 9)$$

Poderíamos também colocar os parênteses na ordem reversa, até o último elemento

$$(1 + (3 + (5 + (7 + (9)))))$$

Observe que o par de parênteses mais interno, (9), é um problema que podemos resolver sem qualquer construção especial. Na verdade, pode-se utilizar a seguinte sequência de simplificações para calcular uma soma final.

- total= $(1 + (3 + (5 + (7 + (9)))))$
- total= $(1 + (3 + (5 + (7 + 9))))$
- total= $(1 + (3 + (5 + 16)))$
- total= $(1 + (3 + 21))$
- total= $(1 + 24)$
- total= 25

A implementação desse exemplo em C:

```
#include <stdio.h>
int soma(int*,int*);
int main() {
    int vetor[] = {1, 3, 5, 7, 9};
    int tamanho = sizeof(vet)/sizeof(int);
    int *pprimeiro = vet, *pultimo = vet+tamanho-1;
    printf("Soma dos elementos = %d\n",
soma(pprimeiro, pultimo));
    return 0;
}
int soma(int *ref, int *px) {
    return (*px) + ((px == ref) ? 0 : soma(ref, px-1));
}
```

Fique à vontade para modificar os elementos do vetor, em valor e em quantidade, e testar esse código.

Definições matemáticas recursivas, como

$$\text{fat(n)} = \begin{cases} 1 & , n = 1 \\ n \cdot \text{fat(n} - 1) & , \text{caso contrario} \end{cases}$$

podem ser traduzidas para C quase que diretamente:

```
#include <stdio.h>
long long int fat(long long int);
int main() {
    int n = 5;
    printf("%d! = %lld\n", n, fat(n));
    return 0;
}
long long int fat(long long int n) {
    if(n == 1) return 1;
    else return n*fat(n-1);
}
```

2 Programas a serem entregues

Os programas a serem entregues precisam seguir o nome da seção em que são descritos, não sendo aceitos programas com outros nomes.

2.1 P4.c

Implemente a seguinte recursão:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Esta é a função de Ackermann (fonte: `https://en.wikipedia.org/wiki/Ackermann_function`). Peça ao usuário para quais `m` e `n` (nesta ordem) e mostre o resultado da função.

2.1.1 Dicas

1. A **única** biblioteca que deverá ser usada é a `stdio.h`

2.1.2 Testes

- "0" + "0" resulta em "1"
- "2" + "3" resulta em "9"
- "3" + "3" resulta em "61"
- "3" + "4" resulta em "125"

3 Informações importantes

É necessário criar em sua conta do github um repositório com o nome 'MTP-2018-2'. É nesse repositório que você dar *upload* do(s) seu(s) código-fonte(s) (ex. arquivos `P1.c`, `P2.c` etc.), não sendo desejado nenhum executável ou arquivo de apoio de projetos.

Em todo programa que você fizer, comece com seu nome e matrícula como comentários. Se não constar essas informações nos arquivos enviados para seu repositório no Github, os programas serão **desconsiderados**.

Mantenha seu código limpo. Não use comandos como `system(pause)` ou `#include<conio.h>` pois são específicos do sistema operacional Windows. Se usá-los, seu código-fonte poderá não compilar, invalidando sua entrega.