

Funções

IPC2

1999/2000

F. Nunes Ferreira

Acetatos baseados no livro

C: How to Program (second edition)

H. M. Deitel

P. J. Deitel

Prentice Hall, 1994

Funções



- ◆ Utilizar, sempre que possível, as funções da *biblioteca standard ANSI C*
- ◆ Por outro lado, o programador escreverá funções para tarefas específicas
- ◆ Biblioteca matemática: `#include <math.h>`
- ◆ De seguida, algumas das funções matemáticas mais comuns

Funções

Funções matemáticas

Funções	Descrição	Exemplos
<code>sqrt(x)</code>	raiz quadrada de x	<code>sqrt(900.0)</code> é 30.0
<code>exp(x)</code>	função exponencial e^x	<code>exp(1.0)</code> é 2.718282
<code>log(x)</code>	logaritmo natural de x (base e)	<code>log(7.389056)</code> é 2.0
<code>log10(x)</code>	logaritmo de x (base 10)	<code>log10(100.0)</code> é 2.0
<code>fabs(x)</code>	valor absoluto de x	<code>fabs(-19.7)</code> é 19.7
<code>ceil(x)</code>	arredonda x para o inteiro imediatamente acima de x	<code>ceil(9.2)</code> é 10 <code>ceil(-9.8)</code> é -9

Funções

Funções matemáticas

Funções	Descrição	Exemplos
<code>floor(x)</code>	arredonda <code>x</code> para o inteiro imediatamente abaixo	<code>floor(9.2)</code> é 9 <code>floor(-9.8)</code> é -10
<code>pow(x, y)</code>	<code>x</code> elevado a <code>y</code> , x^y	<code>pow(2, 7)</code> é 128.0
<code>fmod(x, y)</code>	resto de <code>x/y</code> , em vírgula flutuante	<code>fmod(13.657, 2.333)</code> é 1.992
<code>sin(x)</code>	seno de <code>x</code> (<code>x</code> em radianos)	<code>sin(0.0)</code> é 0.0
<code>cos(x)</code>	cosseno de <code>x</code> (<code>x</code> em radianos)	<code>cos(0.0)</code> é 1.0
<code>tan(x)</code>	tangente de <code>x</code> (<code>x</code> em radianos)	<code>tan(0.0)</code> é 0.0

Funções



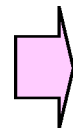
- ◆ Funções são utilizadas para modularizar programas

- ◆ O valor de retorno, o nome e a lista de parâmetros das funções...

- ◆ ... e as suas variáveis locais



- ◆ Cada função deve limitar-se a executar uma tarefa específica e o nome deve expressar aquela tarefa



- ◆ Quando é difícil escolher um nome adequado, talvez seja porque a função faz muitas coisas... O melhor é partir a função em funções mais pequenas

Funções

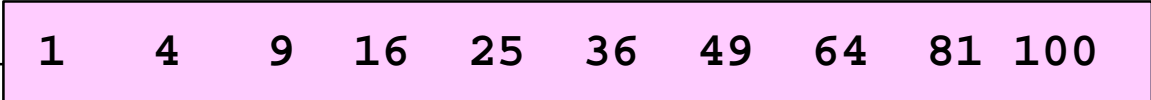
Exemplo

```
#include <stdio.h>
int quadrado(int);           /* protótipo da função */

int main(void)
{
    int x;

    for (x = 1; x <= 10; x++)
        printf("%4d  ", quadrado(x)); /* chamada da função */
    printf("\n");
    return 0;
}

int quadrado(int y)          /* definição de função */
{
    return y * y;
}
```



1 4 9 16 25 36 49 64 81 100

Funções

Definição

```
/* definição de função */  
  
tipo-do-valor-a-devolver nome-função (lista-parametros)  
{  
    declarações locais  
  
    instruções  
}
```

- ◆ *void* como *tipo-do-valor-a-devolver* significa que a função não tem valor de retorno
- ◆ Omitindo *tipo-do-valor-a-devolver* significa que o valor de retorno é *int* (mas não se aconselha...)
- ◆ Não definir uma função dentro de outra função

Funções

Definição: Lista de parâmetros

```
/* definição de função */  
  
tipo-do-valor-a-devolver nome-função (lista-parametros)  
{  
    declarações locais  
  
    instruções  
}
```

- ◆ *lista-parametros*: lista com a definição dos parâmetros separados por vírgula

int maximo(int primeiro, int segundo, int terceiro);

- ◆ *void* como *lista-parametros* significa ausência de parâmetros



- ◆ Erro comum: *float x, y* equivalente a *float x, int y*

Funções

Definição: Considerações gerais

- ◆ Uma função não deve ocupar mais do que uma página...
- ◆ O cabeçalho de uma função deve caber numa linha
- ◆ Uma função com uma longa lista de parâmetros deve estar associada a muitas tarefas...

Solução: Dividir a função em funções mais simples

- ◆ Saída de uma função

```
}      ou      return;
```

```
        }      sem valor de retorno
```

```
return expressão;
```

```
}      com valor de retorno
```

```
#include <stdio.h>
int maximo(int, int, int);
int main(void)
{
    int a, b, c;
    printf("Indicar 3 inteiros: ");
    scanf("%d%d%d", &a, &b, &c);
    printf("O maximo de %d, %d, e %d e': %d\n", a, b, c,
        maximo(a, b, c));
    return 0;
}

int maximo(int x, int y, int z)
{
    int max = x;
    if (y > max)
        max = y;
    if (z > max)
        max = z;
    return max;
}
```

Indicar 3 inteiros: 1 9 4
O maximo de 1, 9, e 4 e': 9

Funções

Protótipo da função

- ◆ Incluir o protótipo de todas as funções usadas, para controlo dos tipos... Controlo que se verifica na *definição* e na *chamada* das funções
- ◆ Usar também `#include` para obter os protótipos das funções da biblioteca standard

Ex: `#include <stdio.h>`

ou de funções próprias preparadas previamente...

Ex: `#include "maximo.h"`

Se assim não for, o compilador cria um protótipo através da primeira ocorrência da função - seja a definição ou uma chamada...

Funções

Protótipo da função

- ◆ O protótipo pode escrever-se:

`int maximo(int, int, int);` *ou também*

`int maximo(int primeiro, int segundo, int terceiro);`

Não esquecer o `;` no final do protótipo...

- ◆ O protótipo faz a *coerção dos argumentos*

– `sqrt(64)` dá 8.000

O parâmetro é *double* e *int* -> *double*

– Mas `quadrado(4.5)` dá 16 !!!








float -> *int*

```
int quadrado(int y)
{
    return y * y;
}
```

Funções

Bibliotecas standard

- ◆ Cada biblioteca standard tem um ficheiro com os protótipos das funções e a definição de tipos e constantes utilizados pelas funções

Ficheiros	Explicação
 <assert.h>	apoiar o <i>debugging</i>
<ctype.c>	manipular caracteres
<errno.h>	reportar condições de erro
<float.h>	limites da vírgula flutuante do sistema
 <limits.h>	limites inteiros do sistema
<locale.h>	adaptar programas a regiões do mundo
 <math.h>	funções matemáticas
<setjmp.h>	funções com esquemas anormais de chamada e retorno
<signal.h>	assinalar várias condições que ocorram na execução
<stddef.h>	definições de tipos para certos cálculos
 <stdio.h>	funções de entrada/saída standard
 <stdlib.h>	funções de conversão números/texto, alocação memória, números aleatórios e outras funções utilitárias
 <string.h>	funções para cadeias de caracteres
 <time.h>	manipulação de tempo e datas

Funções

Chamadas da função

◆ Normalmente, ***chamadas por valor***

- São passadas cópias dos valores dos argumentos
- Não há ***efeitos laterais...***

`quadrado(x);`

◆ ***Chamadas por referência*** em C

- Quando os argumentos são arrays

`preencheValores(valLidos, comprimento);`



- ou simulando com o ***operador endereço*** (&) e
operador de indireção (*)

`scanf("%d" , &num);`

Funções

Exemplo

- ◆ Escrever um programa que põe questões do seguinte tipo, em que os valores numéricos são determinados aleatoriamente.

Quantos são 3 vezes 8?

Se a resposta for correcta responde:

Resposta certa...

se não, responde:

Resposta errada...

O diálogo integra 10 perguntas e, no final, o estudante é aconselhado a estudar a tabuada se tiver errado 2 ou mais perguntas.

- ◆ ***Como gerar números aleatórios?***

Funções

Geração de números aleatórios

- ◆ Correndo os 2 últimos programas, a solução é sempre a mesma!!!
- ◆ A função *srand* ...

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i;
    unsigned voltas_previas;
    printf("Voltas_previas: ");
    scanf("%u", &voltas_previas);
    srand(voltas_previas);

    for (i = 1; i <= 10; i++) {
        printf("%3d", 1 + rand() % 6);
        if (i % 5 == 0)
            printf("\n");
    }
    return 0;
}
```

Voltas-previas: 35

3	6	1	2	4
5	2	2	5	3

Voltas-previas: 10

6	4	3	3	6
3	3	1	3	4

Voltas-previas: 35

3	6	1	2	4
5	2	2	5	3

Funções

Geração de números aleatórios

- ◆ Uma forma de fornecer o argumento a ***srand***

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
...
srand( time( NULL ) )
```

- ◆ ***time (NULL)*** fornece a data e hora do dia corrente, em segundos.

Funções

Exemplo

- ◆ Escrever um programa que põe questões do seguinte tipo, em que os valores numéricos são determinados aleatoriamente.

Quantos são 3 vezes 8?

Se a resposta for correcta responde:

Resposta certa...

se não, responde:

Resposta errada...

O diálogo integra 10 perguntas e, no final, o estudante é aconselhado a estudar a tabuada se tiver errado 2 ou mais perguntas.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int pergunta_resposta(void);
int main(void)
{
    unsigned int i, erradas = 0;
    srand(time(NULL));
    for (i = 1; i <= 10; i++){
        if (pergunta_resposta())
            printf("Resposta certa...\n");
        else{
            printf("Resposta errada...\n");
            ++erradas;
        }
    }
    if (erradas >= 2)
        printf("\n\nE' preciso estudar a tabuada.\n");
    else
        printf("\n\nParabens. Tabuada bem estudada.\n");
    return 0;
}
```

```
int pergunta_resposta(void)
{
    int resposta, n1, n2;

    n1 = rand() % 10;
    n2 = rand() % 10;
    printf("Quantos sao %d vezes %d? ", n1, n2);
    scanf("%d", &resposta);

    return resposta == n1 * n2;
}
```

Funções

Desenvolvimento de programas de média/grande complexidade

- ◆ Supondo exemplo anterior de média complexidade

tabua.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N_PERGUNTAS 10
#define LIM_MIN 2

int pergunta_resposta(void);
```

tabuaMain.c

```
#include "tabua.h"
int main(void)
{
    ...
}
```

pergTabua.c

```
#include "tabua.h"
int pergunta_...
{
    ...
}
```

- *Compilações separadas...*

Funções

Ficheiro tabuaMain.c

```
#include "tabua.h"

int main(void)
{
    unsigned int i, erradas = 0;

    for (i = 1; i <= N_PERGUNTAS; i++){
        if (pergunta_resposta())
            printf("Resposta certa...\n");
        else{
            printf("Resposta errada...\n");
            ++erradas;
        }
    }
    if (erradas >= LIM_MIN)
        printf("\n\nE' preciso estudar a tabuada.\n");
    else
        printf("\n\nParabens. Tabuada bem estudada.\n");
    return 0;
}
```

Funções

Ficheiro pergTabua.c

```
#include "tabua.h"

int pergunta_resposta(void)
{
    int resposta, n1, n2;

    srand(time(NULL));

    n1 = rand() % 10;
    n2 = rand() % 10;
    printf("Quantos sao %d vezes %d? ", n1, n2);
    scanf("%d", &resposta);

    return resposta == n1 * n2;
}
```

Funções

Classes de armazenamento

- ◆ Para além do nome, tipo e valor... classe de armazenamento, duração das entidades...

- ◆ Classes de armazenamento:

auto, *register*, *extern* e *static*



- ◆ **Duração de armazenamento automático**

- Criadas no início de um bloco, retiradas no final
- Ex: variáveis locais de funções

```
auto float x, y;           /* auto é considerada por omissão,  
                           e, por isso, raramente utilizado */
```

```
register int counter = 1;  /* desnecessário, pois os  
                           compiladores já tentam fazer  
                           isto o melhor possível */
```


Funções

Classes de armazenamento e ambientes de acção

- ◆ Classes de armazenamento:

auto, *register*, *extern* e *static*



- ◆ Duração de armazenamento estático

- Existem sempre... Mas podem não estar sempre acessíveis...
- Variáveis globais e nomes de funções são **extern** por omissão
- **static**: para variáveis locais que duram para além do bloco em que são criadas,
... mas só são acessíveis dentro dele

Funções

Classes de armazenamento e ambientes de acção

```
#include <stdio.h>
void a(void);
void b(void);
void c(void);
int x = 1;
int main(void)
{
    int x = 5;
    printf("x local no ambiente mais exterior de main: %d\n", x);
    {
        int x = 7;
        printf("x local no ambiente mais interior de main: %d\n", x);
    }
    printf("x local no ambiente mais exterior de main: %d\n", x);
    a();
    b();
    c();
    a();
    b();
    c();
    printf("x local no ambiente mais exterior de main: %d\n", x);
    return 0;
}
```

Funções

Classes de armazenamento e ambientes de acção

```
void a(void)
{
    int x = 25;
    printf("\nx local e' %d ao entrar em a\n", x);
    ++x;
    printf("x local e' %d ao sair de a\n", x);
}

void b(void)
{
    static int x = 50;
    printf("\nx local e' %d ao entrar em b\n", x);
    ++x;
    printf("x local e' %d ao sair de b\n", x);
}

void c(void)
{
    printf("\nx global e' %d ao entrar em c\n", x);
    x *=10;
    printf("x global e' %d ao sair de c\n", x);
}
```

Funções

Classes de armazenamento e ambientes de acção

```
x local no ambiente mais exterior de main: 5
x local no ambiente mais interior de main: 7
x local no ambiente mais exterior de main: 5

x local e' 25 ao entrar em a
x local e' 26 ao sair de a

x local e' 50 ao entrar em b
x local e' 51 ao sair de b

x global e' 1 ao entrar em c
x global e' 10 ao sair de c

x local e' 25 ao entrar em a
x local e' 26 ao sair de a

x local e' 51 ao entrar em b
x local e' 52 ao sair de b

x global e' 10 ao entrar em c
x global e' 100 ao sair de c
x local no ambiente mais exterior de main: 5
```

static

Funções

Recursividade

◆ Exemplo: Factorial

$n! = n * (n - 1) * (n - 2) * \dots * 1$ - *por iteração*

$n! = n * (n - 1)!$ - *por recursividade*

```
#include <stdio.h>

long factorial(long);
int main(void)
{
    int i;
    for (i = 1; i <= 10; i++)
        printf("%2d! = %ld\n", i, factorial(i));

    return 0;
}

long factorial(long num)
{
    if (num == 1)
        return 1;
    else
        return num * factorial (num - 1);
}
```

1!	=	1
2!	=	2
3!	=	6
4!	=	24
5!	=	120
6!	=	720
7!	=	5040
8!	=	40320
9!	=	362880
10!	=	3628800

Funções

Recursividade vs iteração

- ◆ *Iteração* usa uma estrutura de repetição
- ◆ *Recursividade* usa uma estrutura de selecção

- ◆ *Iteração* termina com a **condição de repetição falsa**
- ◆ *Recursividade* termina com o **caso base**

- ◆ *Recursividade* tem o *overhead* das sucessivas chamadas da função
- ◆ Custa em tempo e espaço

- ◆ Soluções baseadas na **iteração** são mais difíceis de encontrar

Funções

Erros mais comuns

- ◆ Esquecer algum `#include ...`
- ◆ Na definição de uma função, esquecer a indicação do tipo do valor devolvido
- ◆ Esquecer o valor de retorno no corpo da função
- ◆ Especificar um valor de retorno numa função `void`
- ◆ Declarar parâmetros de forma errada. Por exemplo, escrever `float x, y` em vez de `float x, float y`
- ◆ ...

Funções

Erros mais comuns

- ◆ ...
- ◆ Colocar ; imediatamente após o cabeçalho de uma função
- ◆ Esquecer o protótipo da função causa um erro de sintaxe se a função não devolver um *int*. Se não for detectado o erro de sintaxe, outros erros poderão ocorrer, bem mais graves
- ◆ Utilizar um nome para um identificador interno, quando, de facto, se pretende aceder a um identificador externo com o mesmo nome

Funções

Regras de boa programação

- ◆ Tomar conhecimento e utilizar sempre que necessário as funções das bibliotecas standard
- ◆ Apesar de, por omissão, o tipo do valor de retorno das funções ser *int*, convém explicitá-lo. Assim não se costuma fazer com *main*
- ◆ Escolher nomes sugestivos para as funções e parâmetros
- ◆ Variáveis utilizadas numa certa função devem ser declaradas como variáveis locais e não como variáveis globais
- ◆ ...

Funções

Regras de boa programação

- ◆ ...
- ◆ Variáveis globais só devem ser utilizadas em situações em que existam preocupações de desempenho
- ◆ Cada função deve ser limitada a uma simples tarefa
- ◆ Uma função não deve ocupar mais do que uma página. Mas o melhor é não ultrapassar a meia página
- ◆ Uma função com muitos parâmetros indicia muitas tarefas...
- ◆ Uma função cujo protótipo se encontra no interior de uma função, só poderá ser chamada desta função

Exercícios

- ◆ Escrever a função ***quadrado*** que tem apenas um parâmetro, ***lado***.

Ex: ***quadrado(3)***

```
***  
***  
***
```

- ◆ Escrever a função ***quadrado_caract*** com dois parâmetros, ***lado*** e ***character***

Ex: ***quadrado_caract(2, 'p')***

```
pp  
pp
```

- ◆ Escrever um programa que pede um inteiro e escreve os seus dígitos separados por um espaço

Ex: Indicar um inteiro: 2456

2 4 5 6

Exercícios

- ◆ Foi apresentado como exemplo o seguinte exercício:
Escrever um programa que põe questões do seguinte tipo, em que os valores numéricos são determinados aleatoriamente. **Quantos são 3 vezes 8?**

Se a resposta for correcta responde **Resposta certa...**, se não, responde **Resposta errada...**

O diálogo integra 10 perguntas e, no final, o estudante é aconselhado a estudar a tabuada se tiver errado 2 ou mais perguntas.

Agora pretende-se algo de semelhante, mas, aleatoriamente, surgem as seguintes apreciações às respostas do estudante:

Muito bem, Excelente, Boa resposta, Continua assim.

Errou. Tenta de novo, Calma. Tenta de novo, Pensa melhor, Tenta outra vez.

```
#include <studio.h>
#include <stdlib.h>
#include <time.h>
int pergunta_resposta(void);
int main(void)
{
    unsigned int i, erradas = 0;
    for (i = 1; i <= 10; i++){
        if (pergunta_resposta())
            resposta_certa();
        else{
            resposta_errada();
            ++erradas;
        }
    }
    if (erradas >= 2)
        printf("\n\nE' preciso estudar a tabuada.\n");
    else
        printf("\n\nParabens. Tabuada bem estudada.\n");
    return 0;
}
```

```
int pergunta_resposta(void)
{ ... }

void resposta_certa(void);
{
    switch (rand() % 4){
        case 0:
            printf("Muito bem.\n");
            break;
        ...
        case 3:
            printf("Continua assim.\n");
            break;
    }
}

void resposta_errada(void);
{
    switch (rand() % 4){
        case 0:
            printf("Errou. Tenta de novo.\n");
            ...
    }
}
```

Exercícios

- ◆ O máximo divisor comum de dois inteiros positivos é o maior inteiro que é divisor desses positivos.
- ◆ Uma solução recursiva de uma função que calcula o máximo divisor comum:

```
int mdc (int n1, int n2)
{
    int resto;

    if ((resto = n1 % n2) == 0)
        return n2;
    else
        return mdc (n2, resto);
}
```

- ◆ Escrever um programa para testar a função *mdc*
- ◆ Escrever e testar a função *mdc_iter*, uma solução iterativa para este problema

Exercícios

- ◆ n_0 é um inteiro positivo à escolha e

$$n_{i+1} = \begin{cases} n_i / 2 & \text{se } n_i \text{ for par} \\ 3n_i + 1 & \text{se } n_i \text{ for ímpar} \end{cases}$$

- ◆ Uma sequência começa em n_0 evolui de acordo com o que se indicou e termina quando n_i alcançar o valor 1.
- ◆ Sabe-se que

```
void sequencia (int num)
{...
}
```

calcula e visualiza a sequência que começa em *num*.

- ◆ Escrever ...

Exercícios

- ◆ Escrever o programa que se baseia na função *sequencia* e gera, por exemplo, o que se segue, quando o número inicial é 77:

Sequencia gerada a partir de 77

77	232	116	58	29	88
44	22	11	34	17	52
26	13	40	20	10	5
16	8	4	2	1	

Comprimento da sequencia gerada: 23

Exercícios

- ◆ Escrever o programa que permite estabelecer o seguinte tipo de diálogo:

Indicar inteiro a factorizar (EOF termina): 9

9 = 3 x 3

Indicar inteiro a factorizar (EOF termina): 17

17 = primo

Indicar inteiro a factorizar (EOF termina): 52

52 = 2 x 2 x 13

Exercícios

- ◆ Um número é perfeito se a soma dos seus factores (retirando o próprio número) é igual a esse número. 6 é perfeito, pois $1+2+3 = 6$.

- ◆ `int perfeito (int numero)`

é o cabeçalho de uma função que recebe um número inteiro e devolve 1 se ele for perfeito e 0 se não for.

-Escrever a função *perfeito*

-Escrever um programa que sustenta diálogos do

tipo:

Gama de valores: 1 10

Os numeros perfeitos entre 1 e 10:

6

Gama de valores: 100 10000

Os numeros perfeitos entre 100 e 1000:

...

Exercícios

- ◆ O estacionamento num parque tem um custo fixo, até 3 horas, de 200\$00. Por cada hora adicional (ou parte) paga-se 100\$00. O custo máximo, num período de 24 horas, é de 2.000\$00. Não são permitidos períodos de estacionamento superiores a 24 horas.

Escrever o programa que mantém um diálogo do tipo:

Tempo de estacionamento (> 24 horas, termina): 2.5
custo: 200\$00

Tempo de estacionamento (> 24 horas, termina): 3.1
custo: 300\$00

Tempo de estacionamento (> 24 horas, termina): 4.1
custo: 400\$00

Tempo de estacionamento (> 24 horas, termina): 24
custo: 2000\$00

Exercícios

- ◆ Escrever a função *inverte* que toma um valor inteiro e devolve um número correspondente ao número dado, mas com os dígitos em posição invertida. Se o número dado for 12345, devolve 54321.
- ◆ Escrever a função *inverte*
- ◆ Escrever um programa que permita testar a função *inverte*.

Exercícios

◆ Um jogo de sorte - Regras do jogo

Um jogador lança 2 dados.

Se a soma é 7 ou 11, o jogador ganha a soma.

Se a soma é 2, 3, ou 12, o jogador perde a soma.

Se a soma é 4, 5, 6, 8, 9, ou 10... o ponto é a soma e a jogada continua...

... até obter uma soma que é igual ao ponto, e ganha.

... ou até obter uma soma igual a 7, e perde.

jogar (lançar os dados: 1; terminar: 0)

O jogador obteve $6 + 5 = 11$

O jogador ganhou 11 pontos e o saldo e': 11

jogar (lançar os dados : 1; terminar: 0)

O jogador obteve $6 + 6 = 12$

O jogador perdeu 12 pontos e o saldo e': -1

7 ou 11 ...

2, 3 ou 12 ...

4, 5, 6, 8, 9, ou 10 ...

Exercícios

jogar (lançar os dados: 1; terminar: 0)

O jogador obteve $4 + 6 = 10$

O ponto é 10 e continua a jogada.

jogar (lançar dados: 1)

O jogador obteve $2 + 4 = 6$

jogar (lançar dados: 1)

O jogador obteve $6 + 5 = 11$

jogar (lançar dados: 1)

O jogador obteve $6 + 4 = 10$

O jogador ganhou 10 pontos e o saldo é: 9

7 ou 11 ...

2, 3 ou 12 ...

4, 5, 6, 8, 9, ou 10 ...

Exercícios

jogar (lançar dados: 1; terminar: 0)

O jogador obteve $1 + 3 = 4$

O ponto é 4 e continua a jogada.

jogar (lançar dados: 1)

O jogador obteve $1 + 4 = 5$

jogar (lançar dados: 1)

O jogador obteve $5 + 4 = 9$

jogar (lançar dados: 1)

O jogador obteve $5 + 2 = 7$

O jogador perdeu 4 pontos e o saldo é: 5


```
int main(void)
{
    /* lançar dados, enquanto utilizador desejar */

    /* tratamento do resultado do lançamento */

        /* ganhou */

        /* perdeu */

        /* jogada de ponto */

}
```

```
int jogar(void)
{
    /* mensagem para lançar os dados */

    /* mensagem sobre o resultado do lançamento */

    /* devolve soma ou zero ... */
}
```

```
int jogarPonto(int saldo, int soma)
{
    /* mensagem para lançar os dados */

    /* mensagem sobre o resultado do lançamento */

    /* tratamento do resultado do lançamento */

    /* ganhou */

    /* perdeu */

    /* repete */
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int jogar(void);
int jogarPonto(int saldo, int soma);

int main(void)
{
    int soma, saldo = 0;
    srand(time(NULL));

    /* lançar dados, enquanto o utilizador desejar */

    soma = jogar();

    /* tratamento do resultado do lançamento */

    while (soma != 0){
        switch (soma) {

            ...
```

```

switch (soma) {
    case 7: case 11:                                /* ganhou */
        saldo += soma;
        printf("%s%d%s%d\n",
            "O jogador ganhou ", soma,
            " pontos e o saldo e': ", saldo);
        break;

    case 2: case 3: case 12:                        /* perdeu */
        saldo -= soma;
        printf("%s%d%s%d\n",
            "O jogador perdeu ", soma,
            " pontos e o saldo e': ", saldo);
        break;

    default:                                        /* jogada de ponto */
        printf("%s%d%s\n",
            "O ponto e' ", soma,
            " e continua a jogada. ");

        saldo = jogarPonto(saldo, soma);
}
soma = jogar();
}
return 0;
}

```

```

int jogar(void)
{
    int d1, d2, tecla, soma_obtida;
    d1 = rand() % 6 + 1;
    d2 = rand() % 6 + 1;
    soma_obtida = d1 + d2;

    /* mensagem para lançar os dados */

    printf("\njogar (lançar dados: 1; terminar: 0)");
    scanf("%d", &tecla);
    if (tecla == 0)
        return 0;
    else {
        /* mensagem sobre resultado lançamento */

        printf("%s%d%s%d%s%d\n", "O jogador obteve ",
            d1, " + ", d2, " = ", soma_obtida);
        return soma_obtida;
    }
    /* devolve soma ou zero ... */
}

```

```
int jogarPonto(int saldo, int soma)
{
    int d1, d2, tecla, soma_ponto;

    do {
        d1 = rand() % 6 + 1;
        d2 = rand() % 6 + 1;
        soma_ponto = d1 + d2;

        /* mensagem para lançar os dados */

        printf("\njogar (lançar dados: 1)");
        scanf("%d", &tecla);

        /* mensagem sobre o resultado do lançamento */

        printf("%s%d%s%d%s%d\n", "O jogador obteve ",
            d1, " + ", d2, " = ", soma_ponto);

        /* tratamento do resultado do lançamento */

        ...
    }
```

```

if (soma_ponto == soma) {                                     /* ganhou */
    saldo += soma;
    printf("%s%d%s%d\n", "O jogador ganhou ",
           soma, " pontos e o saldo e': ",
           saldo);
}
else if (soma_ponto == 7) {                                    /* perdeu */
    saldo -= soma;
    printf("%s%d%s%d\n", "O jogador perdeu ",
           soma, " pontos e o saldo e': ",
           saldo);
}

/* repete */
} while (soma_ponto != 7 && soma_ponto != soma);

return saldo;
}

```