Programação em C

Francisco de Assis Boldt

15 de março de 2018

Sumário

Capítulo 1

Programas em C

O menor programa completo [1] em C é apresentado no algoritmo 1.

Algoritmo 1: faznada.c

```
1  int main(){
2      return 0;
3  }
```

Apesar do programa gerado pelo código do algoritmo 1 não apresentar nada na tela quando executado, para o sistema operacional (SO) este programa faz alguma coisa. O SO precisa reservar um espaço de memória e tempo de uso do processador para este programa. Além disso, o SO também espera o fim da execução de qualquer programa e exige um código de erro, que é um número inteiro. Quando o programa executa sem erros o código retornado é 0 (zero). Este é o motivo pelo qual o algoritmo 1 inicia com `int`. O `return 0;` na linha 2 diz para o SO que o programa foi executado com sucesso. Em geral, os comandos em C terminam com um ponto e vírgula (;).

A palavra `main` indica que esta é a função principal do programa. No caso do algoritmo 1 é a única função do programa. Mas, um arquivo fonte escrito em C pode conter várias funções. Porém, a função `main` será a primeira a ser chamada pelo SO quando um programa escrito em C for executável. O início e o fim das funções em C são sinalizados por abertura ({) e fechamento (}) de chaves, respectivamente. A abertura e fechamento de parênteses após o nome da função também é obrigatória. Dentro dos parênteses são declarados os parâmetros da função.

1.1 printf

A linguagem C possui várias bibliotecas para ajudar os programadores. Uma delas é a biblioteca de entrada e saída padrão (`stdio.h` - STanDard Input and Output). Esta biblioteca oferece a função `printf`, que exibe uma cadeia de caracteres no terminal. Antes de usar uma biblioteca precisamos incluí-la no programa utilizando a diretiva de compilação `#include`, como pode ser visto no algoritmo 2. O programa gerado por este código imprime a frase "Hello World!" na tela do computador. A abertura e o fechamento das aspas na linha 3 indica que o conteúdo entre elas é uma cadeia de caracteres. O `\n` indica um quebra de linha no final da frase.

Algoritmo 2: hello.c

```
1  #include <stdio.h>
2  int main() {
3      printf("Hello World!\n");
4      return 0;
5  }
```

[1] Programas menores, que usam menos código, podem ser feitos retirando-se a palavra `int` e a linha `return 0;`. Porém, tal programa estaria fora do padrão aceito por qualquer arquitetura, sistema operacional e compilador.

1.2 Variáveis

Programas de computadores executam essencialmente operações matemáticas. Operações como soma podem ser executadas, como mostrado no algoritmo 3. O programa gerado com este código imprime "5 + 7 = 12" na tela. O %d representa um número inteiro que deve vir depois da vírgula, que neste caso é 12.

Algoritmo 3: soma5e7.c

```
1 #include <stdio.h>
2 int main() {
3     printf("5 + 7 = %d\n", 5+7);
4     return 0;
5 }
```

Podemos notar que se precisarmos alterar este programa, por exemplo trocando de 5 para 8, teremos que trocar em dois lugares. Isso não parece ser algo prático, principalmente se tivermos fórmulas mais complexas do que uma simples soma. Então, poderíamos fazer este programa de um forma um pouco mais reutilizável, como mostra o algoritmo 4. Com este algoritmo, caso queiramos mudar de 5 para 8, basta alterarmos a linha 3. Veja que neste caso temos `"%d + %d = %d\n"` ao invés de `"5 + 7 = %d\n"`. Agora, são necessários três números, um para cada `%d`. Os números são associados aos `%d`'s na ordem em que são apresentados.

Algoritmo 4: somaxy.c

```
1 #include <stdio.h>
2 int main() {
3     int x, y;
4     x = 5;
5     y = 7;
6     printf("%d + %d = %d\n", x, y, x+y);
7     return 0;
8 }
```

Para usarmos variáveis em C, precisamos declará-las antes. É assim que pedimos ao SO para reservar um espaço de memória para nossos programas. As variáveis em C possuem tipos com tamanhos diferentes. Então o tipo da variável influencia na quantidade de memória reservada para o programa. A declaração de um número inteiro é feita usando-se a palavra `int` seguida do nome da variável, que deve começar com uma letra. A linguagem C faz distinção entre letras maiúsculas e minúsculas.

Capítulo 2

Subprogramas: Funções e Procedimentos

Funções em C podem ser entendidas como pequenos programas e também podem ser chamadas de subprogramas. Normalmente as linguagens de programação fazem distinção entre funções e procedimentos, onde funções retornam algum valor enquanto procedimentos não.

2.1 Procedimentos

Um exemplo de procedimento é o subprograma que imprime "Hello world!" na tela, como mostra o algoritmo 5.

Algoritmo 5: hello_sub.c

```
1 #include <stdio.h>
2 void hello(){
3     printf("Hello World!\n");
4 }
5 int main() {
6     hello();
7     return 0;
8 }
```

Em C, a diferença entre função e procedimento está no retorno da função. No exemplo do algoritmo 5, a declaração do subprograma `hello` inicia com a palavra reservada `void`, indicando que este subprograma não retorna valor algum e, portanto, é um procedimento.

2.2 Parâmetros

Programas são mais versáteis quando geram saídas diferentes dependendo da entrada. Por exemplo, o procedimento `hello` no algoritmo 5 imprime sempre a mesma frase, o que o torna muito limitado. Muito mais interessante é o procedimento `imprime_soma` apresentado no algoritmo 6. Nota-se que, depois depois de criado, o subprograma pode ser reutilizado quantas vezes for necessário. Ele gera resultados diferentes dependendo dos parâmetros passados.

Ao definir um subprograma, seja ele uma função ou um procedimento, devemos incluir a lista de parâmetros. No caso do procedimento `hello` no algoritmo 5, a lista de parâmetros é vazia, pois não existe nada entre os parênteses colocados após o nome do procedimento. Por outro lado, o procedimento `imprime_soma` define uma lista com dois parâmetros inteiros, `int x` e `int y`. A definição de parâmetros de um subprograma se assemelha muito com a declaração de variáveis.

Algoritmo 6: somaxy_sub.c

```
1 #include <stdio.h>
2 void imprime_soma(int x, int y) {
3       printf("%d + %d = %d\n", x, y, x+y);
4 }
5 int main() {
6       imprime_soma(5, 7);
7       imprime_soma(8, 9);
8       return 0;
9 }
```

2.3 Funções

As funções de linguagens de programação estão intimamente ligadas às funções matemáticas. Tomemos como exemplo o gráfico da função do segundo grau apresentada na figura 2.1.

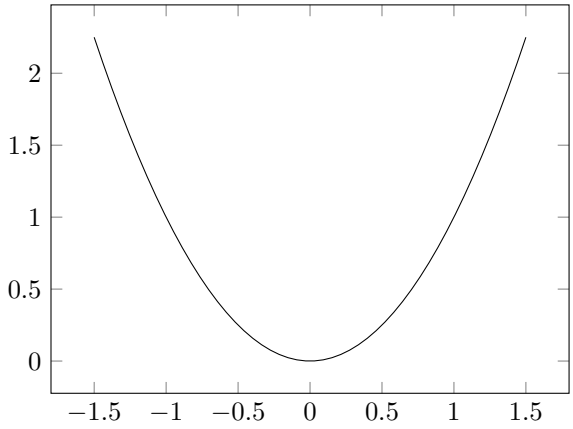

Figura 2.1: $f(x) = x^2$

Qualquer linguagem de programação possui recursos para implementar a função da figura 2.1. Em linguagem C esta implementação pode ser feita como mostra o algoritmo 7.

Algoritmo 7: quadrado.c

```
1 #include <stdio.h>
2 float quadrado(float x) {
3  return x*x;
4 }
5 int main() {
6  float x = -2;
7  printf("f(%f) = %f", x, quadrado(x));
8  return 0;
9 }
```

O algoritmo 7 apresenta algumas novidades. A primeira delas é a palavra reservada `float`, que aparece duas vezes na linha 2 e uma vez na linha 6. A palavra `float` é uma das palavras reservadas que diz ao SO que uma variável, um parâmetro ou o retorno de uma função é um número real, não um número inteiro como quando se usa a palavra `int`. Números inteiros e reais são processados em diferentes partes do processador do computador. Algumas linguagens fazem esta distinção automaticamente, mas este não é o caso da linguagem C. Então, o parâmetro `x` da função `quadrado` é usado para gerar o retorno desta função, que também é um valor do tipo `float`.

2.4 Funções da biblioteca math.h

Como já mencionado na seção 1.1, a linguagem C possui várias bibliotecas para facilitar a programação. Uma biblioteca muito importante é a `math.h`[1]. Esta biblioteca oferece várias funções matemáticas comummente necessárias. Veja o exemplo apresentado no algoritmo 8.

Algoritmo 8: coseno.c

```
1 #include <stdio.h>
2 #include <math.h>
3 int main() {
4   float x = -2;
5   printf("f(%f) = %f", x, cos(x));
6   return 0;
7 }
```

Neste programa nós usamos a função `cos`[2] da bilioteca `math.h` para calcular o coseno (figura 2.2) de um número real. Para isso, precisamos de incluir esta biblioteca com a diretiva de compilação `#include`, assim como feito para a biblioteca `stdio.h`. Depois de incluída a biblioteca `math.h`, tanto a função `cos`, quanto as demais funções oferecidas por essa biblioteca, podem ser usadas como se tivessem sido implementadas no mesmo programa.

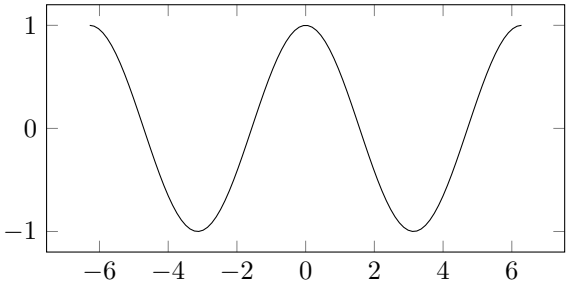

Figura 2.2: $f(x) = cos(x)$

2.5 Exercícios Resolvidos

2.5.1 Função rampa

Analise a função da figura 2.3 e a implemente em linguagem C. [3]

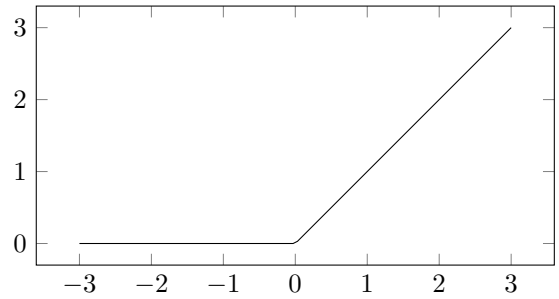

Figura 2.3: $f(x) = rampa(x)$

[1] http://www.cplusplus.com/reference/cmath/
[2] http://www.cplusplus.com/reference/cmath/cos/
[3] Dica: use a função `fabs` da biblioteca `math.h`.

Solução:

A função da figura 2.3 retorna zero para todos os números menores que zero e retorna o próprio número quando este é maior que zero. Podemos ver claramente que o padrão muda quando o domínio da função cresce acima de zero. Vamos solucionar este problema dividindo-o em duas partes.

A primeira para lida com números menores ou iguais a zero e a segunda, com números maiores que zero. Se somarmos números negativos com seus valores absolutos teremos sempre zero. Isso é o que queremos para números menores que zero. Então, o retorno da função em C poderia conter o código `return x+abs(x);`. Isso resolve o problema parcialmente, pois temos zero quando o domínio é menor que zero, mas o valor para domínios positivos é sempre igual a $2 \times x$. O que nos leva para segunda parte da solução.

Como a função para números positivos é $f(x) = 2x$, se aplicarmos a função inversa ($f^{-1}(x) = \frac{x}{2}$) na parte positiva do domínio teremos o valor desejado. Mas, zero dividido por dois é sempre zero. Então esta mudança não afeta a primeira parte da solução. Assim, a função pode ser escrita como $f(x) = \frac{(x+|x|)}{2}$. A solução final é apresentada no algoritmo 9. [4]

Algoritmo 9: rampa.c

```
1  #include <stdio.h>
2  #include <math.h>
3  float rampa(float x) {
4   return (x+fabs(x))/2;
5  }
6  int main() {
7   printf("f(%f) = %f\n", -1.0, rampa(-1));
8   printf("f(%f) = %f\n", 0.0, rampa(0));
9   printf("f(%f) = %f\n", 1.0, rampa(1));
10  return 0;
11 }
```

Este algoritmo possui algumas partes que precisam ser explicadas. Na linha 4 foi usada a função `fabs` da biblioteca `math.h`. A biblioteca `math.h` também possui a função `abs`, mas esta só funciona para números inteiros (`int`). Para números reais (`float`), precisamos usar a função `fabs`. Nas linhas 7, 8 e 9, colocamos os números o `.0` no final. Em C, por padrão, constantes sem o `.0` são consideradas números inteiros. Então, dependendo do compilador ou arquitetura que você usa, o `%f` do `printf` pode não entender a constante inteira e imprimir zero ao invés do número desejado. Por outro lado, quando passamos os valores para a função `rampa` como parâmetros, o compilador já entende que este é um número real pois já foi declarado como tal na linha 3. O retorno da função `rampa` é um número real e portanto compreendido pela função `printf`.

2.5.2 Rampa invertida

Analise a função na figura 2.4 e implemente-a em C. Siga o processo de desenvolvimento mostrado no exercício resolvido anterior. A solução desse exercício é basicamente aquela apresentada no algoritmo 9, alterando as linhas 3 e 4, como mostra o algoritmo 10.

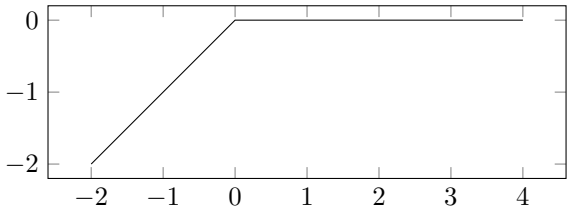

Figura 2.4: $f(x) = rampainv(x)$

[4] A solução do exercício está apenas nas linhas 3,4 e 5. Porém, é importante testar as funções.

Algoritmo 10: rampainv.c

```
1  #include <stdio.h>
2  #include <math.h>
3  float rampainv(float x) {
4   return (x-fabs(x))/2;
5  }
6  int main() {
7   printf("f(%f) = %f\n", -1.0, rampainv(-1));
8   printf("f(%f) = %f\n", 0.0, rampainv(0));
9   printf("f(%f) = %f\n", 1.0, rampainv(1));
10  return 0;
11 }
```

2.5.3 Rampa invertida com máximo 1

A função da figura 2.5 é muito parecida com a função da figura 2.4. Use a função `rampainv` para implementar a função da figura 2.5. Uma possível solução é apresentada no algoritmo 11.

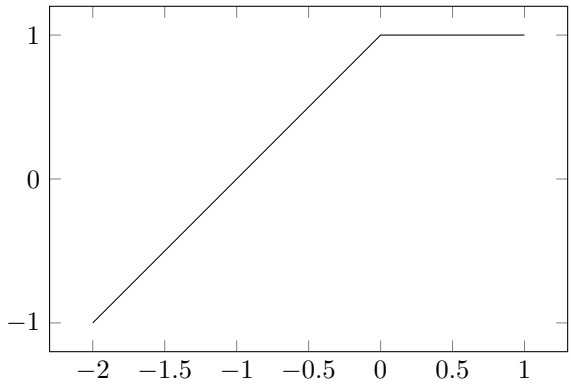

Figura 2.5: $f(x) = rampainvmax1(x)$

Algoritmo 11: rampainvmax1.c

```
1  #include <stdio.h>
2  #include <math.h>
3  float rampainv(float x) {
4   return (x-fabs(x))/2;
5  }
6  float rampainvmax1(float x) {
7   return rampainv(x)+1;
8  }
9  int main() {
10  printf("f(%f) = %f\n", -1.0, rampainvmax1(-1));
11  printf("f(%f) = %f\n", 0.0, rampainvmax1(0));
12  printf("f(%f) = %f\n", 1.0, rampainvmax1(1));
13  return 0;
14 }
```

2.5.4 Rampa invertida com máximo 1 e mínimo 0

Implemente a função da figura 2.6 usando as funções `rampa` e `rampinv`. Baseie-se na implementação da função `rampainvmax1` apresentada no algoritmo 11. Em linguagem C, assim como a maior parte das linguagens de programação existentes, pode-se colocar o retorno de uma função diretamente como parâmetro de outra.

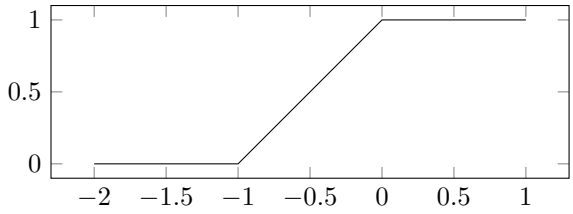

Figura 2.6: $f(x) = rampainvmax1min0(x)$

Algoritmo 12: rampainvmax1min0.c

```
1  #include <stdio.h>
2  #include <math.h>
3  float rampa(float x) {
4   return (x+fabs(x))/2;
5  }
6  float rampainv(float x) {
7   return (x-fabs(x))/2;
8  }
9  float rampainvmax1min0(float x) {
10  return rampa(rampainv(x)+1);
11 }
12 int main() {
13  printf("f(%f) = %f\n", -1.0, rampainvmax1min0(-1));
14  printf("f(%f) = %f\n", -0.5, rampainvmax1min0(-0.5));
15  printf("f(%f) = %f\n", 0.0, rampainvmax1min0(0));
16  return 0;
17 }
```

2.5.5 Rampa íngreme

Enquanto a rampa da função na figura 2.6 tem 45° de inclinação a rampa da função na figura 2.7 tem 60° de inclinação. Implemente a função da figura 2.7.

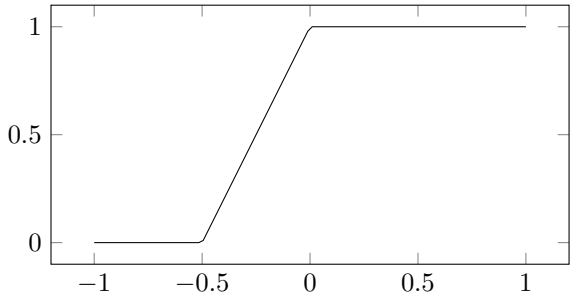

Figura 2.7: $f(x) = rampaingreme(x)$

Algoritmo 13: rampaingreme.c

```
1  #include <stdio.h>
2  #include <math.h>
3  float rampa(float x) {
4   return (x+fabs(x))/2;
5  }
6  float rampainv(float x) {
7   return (x-fabs(x))/2;
8  }
9  float rampaingreme(float x) {
10  const float aproxim = 0.5;
11  return rampa(rampainv(x)+aproxim)/aproxim;
12 }
13 int main() {
14  printf("f(%f) = %f\n", -1.0, rampaingreme(-1));
15  printf("f(%f) = %f\n", -0.5, rampaingreme(-0.5));
16  printf("f(%f) = %f\n", 0.0, rampaingreme(0));
17  return 0;
18 }
```

2.5.6 Função degrau

Baseado na função `rampaingreme` do algoritmo 13, implemente a função `degrau` mostrada na figura 2.8

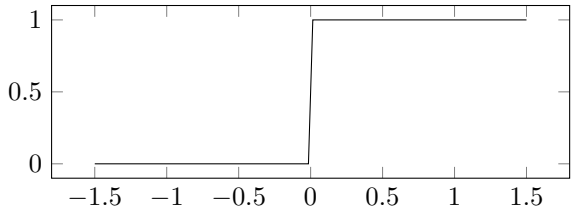

Figura 2.8: $f(x) = degrau(x)$

Algoritmo 14: degrau.c

```
1  #include <stdio.h>
2  #include <math.h>
3  float relu(float x) {
4   return (x+fabs(x))/2;
5  }
6  float degrau(float x) {
7   return relu(x)/(relu(x)+0.000000000000000000001);
8  }
9  int main() {
10  printf("f(%f) = %f\n", -1.0, degrau(-1));
11  printf("f(%f) = %f\n", 0.0, degrau(0));
12  printf("f(%f) = %f\n", 1.0, degrau(1));
13  printf("f(%f) = %f\n", 0.00000000001, degrau(0.00000000001));
14  return 0;
15 }
```