Transcrição do Curso de introdução à linguagem C:

Olá! Na primeira parte do nosso curso de Linguagem C, trabalharemos em um **jogo de adivinhação**. Ao rodar o jogo, o computador perguntará um número e, cuja resposta poderia ser 10, por exemplo, e o computador poderia retrucar informando que pensou em outro número, maior que 10. Então, você tenta o número 50, e o computador diz que é menor que 50. E assim será a "brincadeira" de descobrir o número que o computador pensou.

Parece simples, mas o jogo tem muitos programas computacionais para resolvermos. Assim, aprenderemos a:

* lidar com entradas e saídas;
* como escrever no monitor;
* como ler do teclado;
* elaborar variáveis para guardar respostas e números que o computador pensar;
* gerar números randômicos (para o jogo ficar legal, toda vez que rodar, precisará de um número diferente);
* criar um programa que tome decisões - se acertar o número, o jogador vence e o jogo acaba. Se errar o número, o jogo precisa dizer se o número certo é maior ou menor
* a partir de *if*, gerar *loops* para fazer o programa repetir até que o jogador acerte o número e todas suas tentativas acabem - terá um número máximo de tentativas, caso contrário o jogo fica fácil.

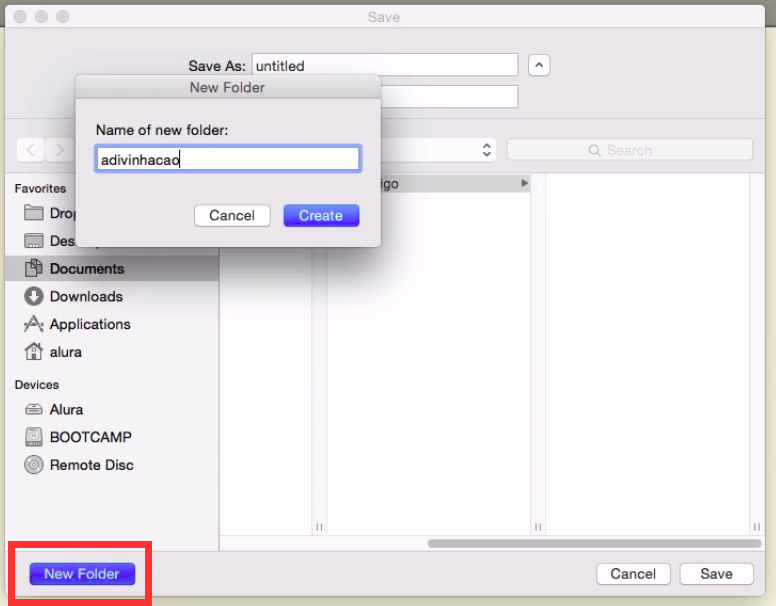
Pronto para desenvolver seu primeiro jogo? Vamos lá!

Desenvolver um programa de computador significa que faremos com que ele se comporte da maneira esperada, por meio de um conjunto de linhas e instruções escritas por nós, que o computador entenderá e executará. Nosso **primeiro passo** será abrir um editor de texto, no qual serão escritas as instruções.

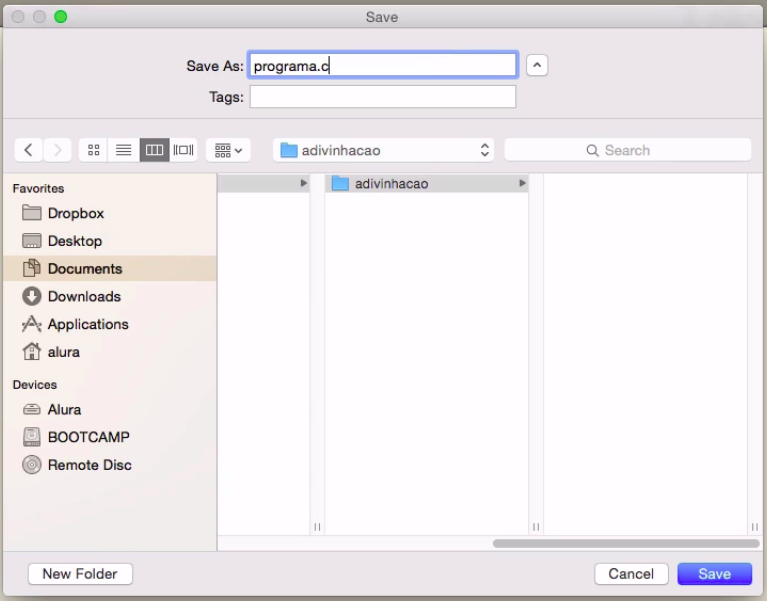
No entanto não trabalharemos com um editor de texto comum como o Word, geralmente utilizado para escrever cartas ou fazer trabalhos de faculdade, e sim um editor de texto especial para código. Se digitarmos "IDE" (nome utilizado para editores especiais para código) no Google, encontraremos diversos tipos que podem ser utilizados para Linguagem C.

Aqui, utilizaremos um editor mais simples, o ***Sublime***, similar ao UltraEdit ou o Bloco de Notas do Windows. Ele é simples, visualmente agradável, e tem a função de colorir o código, o que facilita na hora de lidarmos com códigos extensos. Fique à vontade para escolher o melhor editor para você. Dependendo do sistema operacional, pode-se usar o *Note Pad* também.

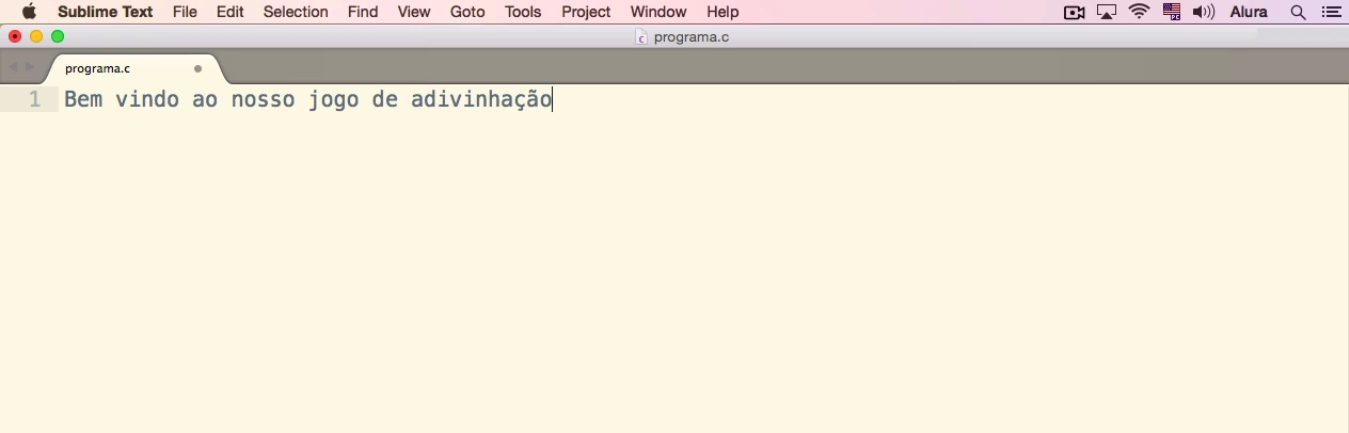
Começaremos escrevendo nosso programa, o qual se encontra em um arquivo que iremos salvar ainda em branco. Para isso, podemos utilizar o atalho "Command + S" ou "Ctrl + S". Escolha um diretório para arquivá-lo e crie uma pasta em "New Folder" para o projeto. Neste caso, usaremos o nome de adivinhacao para o jogo.



Salvaremos o arquivo como programa.c, pois sempre que fizermos um programa em C, devemos salvá-lo com a extensão **.c**, assim como arquivos do Word terminam com .doc e os do Excel, que terminam com .xls.



Ao ser iniciado, nosso programa exibirá uma mensagem de boas-vindas ao usuário, escreveremos simplesmente Bem-vindo ao nosso jogo de adivinhação.



Porém, ainda não é um código em C. Por enquanto, é um texto simples que o computador não entende como comando para imprimir o texto de boas-vindas. Para fazê-lo, colocaremos a mensagem entre aspas.



Agora, diremos para o programa "exiba na tela o texto 'Bem-vindo ao nosso jogo de adivinhação'". Infelizmente, programar não é tão simples assim, em português. Usaremos diversas palavras reservadas da Linguagem C. Se quisermos escrever na tela, usaremos a palavra printf, abrindo um parênteses para o nosso texto. Em inglês, *print* significa imprimir.

printf("Bem vindo ao nosso jogo de adivinhação")

Entenderemos adiante a função dos parênteses. Por enquanto, lembrem-se que sempre que usarmos printf, em seguida abriremos parênteses para inserir um texto entre aspas. No vocabulário de Linguagem C, falaremos ***string*** quando quisermos nos referir a um texto, e tudo que for aberto terá de ser fechado - parênteses, chaves e colchetes, por exemplo. E toda linha termina com ponto e vírgula ou chaves.

printf("Bem-vindo ao nosso jogo de adivinhação");

No entanto, da maneira em que está, o texto ainda não é um programa. Acrescentaremos a função int main() para que a Linguagem C entenda que, ao rodar o programa, a primeira função a ser executada deve ser o printf:

int main() {

printf("Bem-vindo ao nosso jogo de adivinhação");

}

O importante é saber que para começar a escrever um programa em C, é necessário escrever int main(). No exemplo, fechei as chaves depois do printf, pois queremos passar a ideia de que ele está dentro desse bloco de código, que está separado pelas chaves. Falta só um detalhe. Para usarmos o printf precisamos avisar o programa em C que o jogo escreverá, e depois vai ler pelo teclado. Para avisá-lo que a função printf será utilizada, incluiremos a biblioteca (conjunto de funções que podem ser utilizadas no programa) em que ela se encontra por meio de #include <stdio.h>.

#include <stdio.h>

int main() {

printf("Bem-vindo ao nosso jogo de adivinhação");

}

Agora, sim, temos um programa em C! Precisamos rodá-lo para ver o computador imprimir o que foi feito. O computador não entende diretamente essa linguagem, binária, em que usamos apenas os números 0 e 1. No entanto, nós, seres humanos, não a entendemos, e por isto transformaremos o código que desenvolvemos em código de máquina por meio do **compilador**, que transforma a Linguagem C em dígitos 0 e 1.

Para usar o compilador é necessário tê-lo instalado na máquina. A linguagem C tem diversos compiladores. Utilizaremos o **GCC**, disponível para Linux e MAC. Se for utilizar o Windows, basta procurar no Google por "GCC download Windows" e seguir o instalador padrão. Aberto o **terminal** (no Windows, é o *Prompt* de comando), vamos inserir o código, que localizará o programa de acordo com o diretório em que salvamos o arquivo:

Alura: codigo alura$ pwd

/Users/alura/Documents/aniche/curso-c/codigo

Alura:codigo alura $ ls

adivinhacao

Alura: codigo alura$ cd adivinhacao/

Alura: adivinhacao alura$

programa.c

Alura:adivinhacao alura$

O programa não roda ainda; é necessário compilá-lo, utilizando o comando gcc programa.c -o programa.out ou gcc programa.c -o programa.exe no Windows:

Alura: codigo alura$ pwd

/Users/alura/Documents/aniche/curso-c/codigo

Alura:codigo alura $ ls

adivinhacao

Alura: codigo alura$ cd adivinhacao/

Alura: adivinhacao alura$

programa.c

Alura:adivinhacao alura$ gcc programa.c -o programa.out

Isto significa que:

* gcc é o compilador;
* programa.c é o nome do nosso programa;
* -o é o local onde o código binário, que só a máquina entende, será escrito;
* programa.out ou programa.exeé o outro nome.

Após apertarmos "Enter", será indicado que o programa está inativo, mostrando que o nosso programa.c foi compilado com sucesso.

Alura: codigo alura$ pwd

/Users/alura/Documents/aniche/curso-c/codigo

Alura:codigo alura $ ls

adivinhacao

Alura: codigo alura$ cd adivinhacao/

Alura: adivinhacao alura$

programa.c

Alura:adivinhacao alura$ gcc programa.c -o programa.out

Alura:adivinhacao alura$

Acrescentaremos um ls e teremos o programa.out:

Alura: codigo alura$ pwd

/Users/alura/Documents/aniche/curso-c/codigo

Alura:codigo alura $ ls

adivinhacao

Alura: codigo alura$ cd adivinhacao/

Alura: adivinhacao alura$

programa.c

Alura:adivinhacao alura$ gcc programa.c -o programa.out

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$

Se tentarmos imprimir o programa.c com instrução cat, ele o fará na tela e conseguiremos entender.

Alura: codigo alura$ pwd

/Users/alura/Documents/aniche/curso-c/codigo

Alura:codigo alura $

Alura:adivinhacao alura$ ls

programa.c

Alura:adivinhacao alura$ gcc programa.c -o programa.exe

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ cat programa.c

#include <stdio.h>

int main() {

printf("Bem-vindo ao nosso jogo de adivinhação");

}Alura:adivinhacao alura$

Se tentarmos imprimir o programa.out:

Alura: codigo alura$ pwd

/Users/alura/Documents/aniche/curso-c/codigo

Alura:codigo alura $

Alura:adivinhacao alura$ ls

programa.c

Alura:adivinhacao alura$ gcc programa.c -o programa.exe

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ cat programa.c

#include <stdio.h>

int main() {

printf("Bem-vindo ao nosso jogo de adivinhação");

}Alura:adivinhacao alura$ cat programa.out

Em seguida, a tela exibirá códigos que apenas o seu sistema operacional consegue entender. Vamos limpar a tela com o comando clear. Encontraremos o programa.out.

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ ./programa.out

No Linux ou no MAC, executaremos ./programa.out e, no caso do Windows, será necessário digitarmos ./programa.exe. Ao rodá-lo, teremos como retorno Bem-vindo:

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ ./programa.out

Bem-vindo ao nosso jogo de adivinhaçãoAlura:adivinhacao alura$

Esse é nosso primeiro programa! Aprendemos a compilar, escrevemos um programa em C no *Sublime Text*. Após a compilação, é gerado um executável, e depois rodamos e tudo funcionou. Adiante, aprimoraremos o texto.

Anteriormente, estudamos:

* como iniciar um programa;
* como o código deve ficar dentro do int main();
* a necessidade de compilação para que a máquina entenda e execute o programa.

Começaremos esta aula renomeando nosso programa, pois programa.c é muito genérico. Usaremos adivinhacao.c, e para isso acrescentaremos no terminal os comandos:

Alura:adivinhacao alura$ rm programa.out

Alura:adivinhacao alura$ mv programa.c adivinhacao.c

Assim, o nome programa.c será alterado para adivinhacao.c.

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ ./programa.out

Bem-vindo ao nosso jogo de adivinhaçãoAlura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ rm programa.out

Alura:adivinhacao alura$ mv programa.c adivinhacao.c

Alura:adivinhacao alura$ ls

adivinhacao.c

Alura:adivinhacao alura$

Adiante, veremos a importância da nomeação de arquivos. De volta ao editor de texto Sublime, aprimoraremos o texto que estamos desenvolvendo acrescentando funções printf().

#include <stdio.h>

int main() {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

Deixaremos nosso texto com aparência de *banner*, e em seguida compilaremos no nosso console.

Alura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ ./programa.out

Bem-vindo ao nosso jogo de adivinhaçãoAlura:adivinhacao alura$ ls

programa.c programa.out

Alura:adivinhacao alura$ rm programa.out

Alura:adivinhacao alura$ mv programa.c adivinhacao.c

Alura:adivinhacao alura$ ls

adivinhacao.c

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ls

adivinhacao.c adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bem-vindo ao nosso jogo de adivinhação \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Alura:adivinhacao alura$

O texto atualizado é apresentado, porém sem o alinhamento planejado. Para que a visualização do texto seja executada da forma planejada, será necessário adicionarmos um comando \n, que na Linguagem C equivale a "Enter". Ao adicionarmos esse comando no Sublime, ele ficará na cor vermelha.

#include <stdio.h>

int main() {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

Voltaremos ao terminal para compilar o código atualizado e, com o comando clear limparemos a tela e inseriremos o código do programa:

Alura:adivinhacao alura$ ls

adivinhacao.c adivinhacao.out

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Alura:adivinhacao alura$

Agora que adicionamos \n, o texto ficou da maneira que planejamos. É necessário compilar para verificarmos o código, considerando que se escrevermos algo errado, o compilador não funcionará. Testaremos isso retirando o ponto e vírgula do terceiro printf() do código no Sublime.

#include <stdio.h>

int main() {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

}

Salvo o arquivo no editor de texto, compilaremos o programa no terminal, e por meio do clique da seta para cima, o comando gcc adivinhacao.c -o adivinhacao.out será acrescentado automaticamente:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Como retorno, teremos o seguinte erro:

adivinhacao.c:6:56: error: expected ';' after expression

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

^

;

1 error generated.

Alura:adivinhacao alura$

O compilador indica como localizar o erro:

* Localização: adivinhacao.c:6:56 – linha 6, coluna 56. Lembrando que no Sublime, a contagem de espaços ou colunas é maior, porque ele não considera o "Tab".
* Erro: error: expected ';' after expression – o compilador esperava um ponto e vírgula após a expressão.
* Expressão que deveria ter ponto e vírgula no final: printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n").

Cada item que esquecemos gera um erro diferente. Se não fecharmos a chave do código no Sublime, por exemplo:

#include <stdio.h>

int main() {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

O erro comunicado no terminal será diferente:

adivinhacao.c:7:1: error: expected '}'

^

adivinhacao.c:3:12: note: to match this '{'

int main() {

^

1 error generated.

O computador aponta onde estão os erros, mas não os corrige. Nesse caso, ele apontou que a chave não foi fechada, então temos que fazê-lo. Corrigido o erro no editor de texto:

#include <stdio.h>

int main() {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

Compilaremos no terminal e o programa é executado sem problemas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Alura:adivinhacao alura$

Por meio de comentários no C com o uso de **//**, antes que o programa comece a crescer, sinalizaremos o que cada trecho de código faz:

#include <stdio.h>

int main() {

// imprime cabecalho do nosso jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

Ao compilarmos no terminal, veremos que isto não altera a execução do programa. O compilador oculta o que estiver após //. Utilizaremos esse comando para organizar e facilitar a leitura do código, quando necessário.

Nesta aula, estudamos:

* Equivalentes em Linguagem C:
  + *Enter* = \n, dentro da string
  + para adicionar comentários – visíveis somente para quem está programando – colocamos // antes do texto
* Quando há erros no código, o compilador avisa e aponta onde encontrá-los.

Já imprimimos o cabeçalho do jogo, o próximo passo será guardar um número secreto. Lembrem-se: é um jogo de adivinhação, então temos que fazer a máquina pensar em um número. A princípio, faremos com que a máquina pense em um número fixo. Adiante, programaremos para que ela execute de forma aleatória.

Suponhamos que a máquina pensou no número 42. Precisaremos guardá-lo em algum lugar, para que seja encontrado quando necessário. Para isso, nomearemos 42 como numerosecreto = 42 no Sublime.

#include <stdio.h>

int main() {

// imprime cabecalho do nosso jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

numerosecreto = 42;

}

Iniciaremos o desenvolvimento de uma variável ao nomearmos e criarmos um "espaço" para o número 42. Para programadores, as variáveis funcionam como armazenadoras de valores, um espaço de memória para guardar um número, por exemplo. Para isto, criaremos um espaço de memória nomeado como numerosecreto — poderíamos utilizar outro nome.

Porém, esse "espaço" não é uma variável ainda. Para se tornar uma, precisaremos determinar o tipo, especificando o que ela armazenará — texto, string, letra, número inteiro ou número decimal. Considerando que 42 é um número inteiro, acrescentaremos int:

int numerosecreto = 42;

Vamos declarar uma variável chamada numerosecreto de valor 42. Poderemos declarar quantas variáveis forem necessárias, inclusive em duas linhas, em um programa. Por exemplo:

int idade = 25;

int tamanhodocalcado;

tamanhodocalcado = 39;

Esse foi um exemplo para demonstrar formas de adicionarmos variáveis. Voltando ao código do jogo, no Sublime, tentaremos imprimir a variável criada por meio do printf():

#include <stdio.h>

int main() {

// imprime cabecalho do nosso jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

printf("numerosecreto é o número secreto. Não conta para ninguém!")

}

Ao rodarmos esse novo código no terminal, teremos como retorno:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

numerosecreto é o número secreto. Não conta para ninguém!Alura:adivinhacao alura$

Notamos que ele não substituiu numerosecreto pela variável. Para que isso aconteça, adicionaremos uma **máscara** por meio de %d. Dessa forma, indicaremos para printf() que neste espaço deve entrar um número inteiro, o qual será impresso na tela. Após a string, passaremos numerosecreto como segundo parâmetro, para substituir o valor.

#include <stdio.h>

int main() {

// imprime cabecalho do nosso jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

printf("%d é o número secreto. Não conta para ninguém!/n", numerosecreto)

}

Observe que os dois argumentos foram separados por vírgula ( **,**). O primeiro foi o texto, no qual inserimos a máscara %d, para ser substituída pelo que definirmos em seguida. O segundo foi o argumento para o printf (numerosecreto). No terminal, ao compilarmos o código, teremos o seguinte retorno:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

42 é o número secreto. Não conta para ninguém!

Alura:adivinhacao alura$

Agora o código foi compilado da maneira que planejamos.

A variável que substituirá a máscara %d sempre será aquela que especificarmos após a vírgula (**,**).

Em resumo, estudaremos variáveis e como:

* declará-las;
* indicar o tipo de uma variável por meio de int;
* nomeá-las;
* finalizar uma declaração de variável com ponto e vírgula (**;**);
* iniciar uma variável na mesma linha, adicionando sinal de igual (**=**) e um número inteiro (42, no exemplo);
* imprimir por meio de printf(), inserindo a máscara %d, com vírgula (**,**) e a variável que substituirá a máscara na sequência.

O printf (numerosecreto) foi utilizado somente para demonstrar como trabalharmos com variáveis. Como estamos desenvolvendo um jogo de adivinhação, excluiremos esse printf() que imprime o número secreto.

#include <stdio.h>

int main() {

// imprime cabecalho do nosso jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

}

Adiante, continuaremos o desenvolvimento do jogo.

Agora que máquina tem um número secreto, vamos programá-la para desafiar o usuário a acertar o número. O usuário irá digitar uma tentativa, e precisamos imprimir o "chute" dele na tela. Para que a máquina leia isto, precisaremos desenvolver um local para guardar a informação digitada. Anteriormente, vimos que para armazenar dados, utilizamos variáveis. Sendo assim, para guardar as tentativas do usuário, acrescentaremos ao código:

int chute;

Adicionaremos também um printf() para imprimir uma mensagem questionando qual o chute do jogador. Por fins estéticos, ao fim do texto, adicionaremos um espaço para que a resposta do usuário não fique emendada no ponto de interrogação. Lembrem-se que o jogo deve ficar atraente para o usuário.

printf("Qual é o seu chute? ");

Para que o computador leia o que é digitado, utilizaremos scanf(), função que recebe dois parâmetros, sendo o primeiro deles uma string, em que informamos o que será lido pelo programa. Nesse caso, leremos um número inteiro (1, 2, 12, 100...). Indicaremos isso da mesma forma que fizemos com o printf() quando utilizamos %d para colocar um número inteiro. O segundo parâmetro será a variável chute, que armazenará a tentativa do jogador e será inserida após uma vírgula (**,**).

scanf("%d", chute);

Um detalhe interessante: teremos que passar um "e" comercial (**&**) — conceito que estudaremos mais adiante, quando tivermos uma boa base de Linguagem C — antes da variável:

scanf("%d", &chute);

No momento, o importante é saber que a função scanf() lê o que é digitado, e a variável, que deve estar acompanhada de um "e" comercial (**&**), armazenará a informação. No exemplo do jogo de adivinhação, o programa armazenará o conteúdo digitado na variável chute. Para garantir que está tudo certo, imprimiremos a tentativa do usuário por meio de:

printf("Seu chute foi %d", chute);

Observe que neste printf() **não** colocaremos o &, que é uma função especial do scanf. Assim, o código ficará da seguinte maneira:

#include <stdio.h>

int main() {

// imprime cabecalho do nosso jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

int chute;

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

}

No terminal, limparemos a tela por meio de clear e faremos o teste do código atualizado.

Alura:adivinhacao alura$ ls

adivinhacao.c adivinhacao.out

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute?

Se respondermos a pergunta com o número 20, teremos o seguinte retorno:

Seu chute foi 20

Alura:adivinhacaoalura$

Funcionou da maneira que planejamos! Vamos revisar o código desenvolvido no Sublime:

* int main() é o ponto de partida. Todo programa começa executando o código que está dentro do método main;
* // imprime cabecalho do nosso jogo é um comentário que será visualizado somente pelo programador;
* printf() imprime na tela o que estiver escrito entre aspas, dentro do parênteses;
* \n é o código que imprime "Enter";
* ; toda linha deve apresentar ponto e vírgula no final;
* int numerosecreto = 42; é uma variável de valor 42. Importante lembrar que é um número inteiro, e por isso colocamos int na frente;
* int chute; também é uma variável, porém sem valor pois isto será inserido pelo usuário;
* scanf() é a função que lê o que é digitado, e possui dois parâmetros:
  + %d, máscara que indica o que será lido;
  + variável chute acompanhada do "e" comercial (**&**)

Lembrem-se que cada máscara deve ter um argumento, que deverá ser separado por vírgulas.

Somente para exemplificar, vamos inserir o seguinte exemplo no final do código:

printf("%d %d\n", numerosecreto, chute);

Se compilarmos esse código, o retorno será:

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

42 10

Alura:adicinhacao alura$

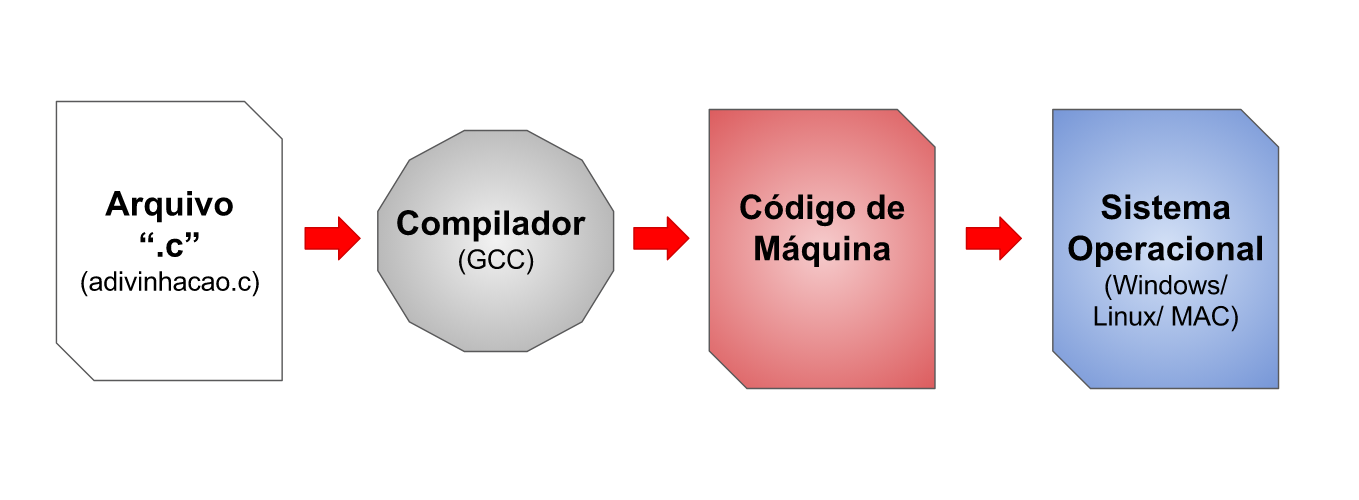
Veremos que ele imprimirá o número secreto 42, e o chute 10, seguindo-se a lógica de que o número de máscaras deve ser igual ao de argumentos. No entanto, considerando que é um jogo de adivinhação, não queremos que o número secreto seja revelado, então apagaremos do código a linha do exemplo que acabamos de conferir.

Notem que no editor de texto colocamos "Enter" (\n) para separar os códigos, deixando-os visualmente organizados. O "Enter" é ignorado pelo computador, portanto poderemos colocar a quantidade que for **necessária** para nossa organização. O objetivo é tornar seu código fácil para leitura.

Antes de rodarmos o programa no terminal, passaremos o arquivo adivinhacao.c pelo GCC, porque temos o programa no formato .c, com código em linguagem C, que estamos aprendendo a desenvolver e interpretar nesse curso. A intenção é executar o código desenvolvido no computador, que roda em um algum sistema operacional (Windows, Linux ou MAC).

Porém, o sistema operacional em si não entende linguagem C. Computadores operam por meio de linguagem de máquina, composta por números 0 e 1, que seres humanos não entendem. Portanto, para desenvolver um programa, é necessário uma linguagem intermediária (a Linguagem C, por exemplo, é um "meio termo" entre português e números 0 e 1) e um "tradutor" de vocabulário de programação para um que o sistema operacional a entenda.

Esse "tradutor" é o **compilador**, dos quais existem diversos tipos. O que utilizaremos nesse curso é o **GCC**, que lê o código em linguagem C e o transforma em código de máquina. Assim, o arquivo.c é passado para o compilador, que o transforma em código de máquina, em arquivo.out ou arquivo.exe, permitindo então que o sistema operacional leia e execute o código.



O printf() que conseguimos ler e entender é transformado em um código de máquina, e essa é a função de um compilador. Não importa a linguagem de programação que for utilizada, neste caso estamos trabalhando com C, mas há também Java, C#, Scala, entre outros, e todas elas precisam passar por um compilador para que seja produzido um código que o sistema operacional consiga entender.

Lembrem-se de que para programar em Linguagem C, será necessário compilar o arquivo salvo em formato .c , produzindo código de máquina que executável pelo sistema operacional.

Nesse início de desenvolvimento de código, abordamos muitos conteúdos, como:

* todo programa em Linguagem C tem a função main, que atua como um ponto de partida. Ao rodar o programa no terminal, ela será a primeira função executada;
* printf(), que imprime uma string na tela;
* a string pode conter caracteres especiais, como:
  + \n, equivalente a "Enter";
  + %d, máscara que permite que o local em que está seja substituído por uma variável;
* declarar variáveis — no exemplo que trabalhamos, queríamos guardar um número secreto, e para isso declaramos uma variável de tipo int, pois guardava um número inteiro;
* scanf(), função utilizada para ler o que é digitado, e que permite o uso de máscara, como %d em printf(), para guardar a variável digitada pelo usuário;
* adicionar comentários, que são bons:
  + por serem visíveis somente para o programador;
  + para explicar o código que vem em seguida.

A partir desse início de código que trabalhamos, o programa está criando forma e começando a parecer um jogo, mas temos muito o que desenvolver ainda!

O próximo passo do desenvolvimento do jogo é dizer se o número secreto que o usuário digitou está certo ou não. Se ele acertou, daremos parabéns pra ele. Se ele errou, o avisaremos. Para isso, o código poderia ser:

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

printf("Parabéns, você acertou!");

printf("Poxa vida, você errou!");

Mas se rodarmos nosso código do jeito que está, no terminal, o comportamento do programa será estranho, pois ele imprimirá **ambas** as mensagens:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ls

adivinhacao.c adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

Parabéns! Você acertou! Você errou!Alura:adivinhacao alura$

Lembrando que o GCC é o compilador, responsável por "traduzir" o código em C para o código de máquina. Voltando ao desenvolvimento do código, para obtermos um retorno correto, precisaremos definir qual mensagem imprimir dependendo da resposta. Portanto, nosso programa precisa tomar uma **decisão**, imprimindo uma mensagem **ou** outra, e existem diversas maneiras de ensinarmos o programa a decidir isto.

Em nosso jogo, a decisão é fácil: se o chute dado pelo usuário for igual ao número secreto, avisaremos que ele acertou; caso contrário, que ele errou. Por exemplo, se pensarmos no código da mesma forma que pensamos para nos comunicar em português, desenvolveríamos um pseudo-código, como:

SE O CHUTE FOR IGUAL AO NUMERO SECRETO

printf("Parabéns! Você acertou!");

CASO CONTRARIO

printf("Parabéns! Você errou!");

Em código, é bem parecido, mas usaremos palavras em inglês. Vamos começar com o "se", que é uma **condição** e em inglês é if. Em linguagem C, o nosso comando que está em português como "SE O CHUTE FOR IGUAL AO NUMERO SECRETO" ficaria da seguinte forma:

if(chute == numerosecreto) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

Em que:

* if() equivale à condição "se". Entre os parênteses, colocamos a condição que ele deve avaliar;
* chute e numerosecreto são as variáveis com que estamos trabalhando;
* o duplo sinal de igual (**==**) tem a função de comparar, para verificar se os valores são os mesmos. Ele se difere do sinal de igual simples (**=**), utilizado para atribuir valor para variáveis;
* {}, como no main, abre-se uma chave pra indicar que ali há um novo bloco de código, que deverá ser executado caso aquela condição seja verdadeira.

O "caso contrário", em C, será substituído por else:

else {

printf("Parabéns! Você errou!\n");

printf("Mas não desanime, tente de novo!\n");

}

Rodaremos o código atualizado no terminal. A primeira tentativa será com um número errado (10):

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ls

adivinhacao.c adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

Você errou!

Mas não desanime, tente de novo!

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Identifica-se que a resposta não está correta e imprime-se a mensagem para caso de resposta errada, como gostaríamos. Se tentarmos novamente, com o número correto (42), teremos o seguinte retorno:

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Alura:adivinhacao alura$

O programa ficou inteligente! Ensinamos a máquina a pensar e a tomar decisões pelo if, com base nas condições que definimos. Existem diversas formas de passar condições, e uma delas é por meio de **==**, que compara valores e funciona perfeitamente para o jogo de adivinhação que estamos desenvolvendo.

Entretanto, o usuário precisa de mais dicas, e precisaremos dizer se o chute dele foi maior ou menor que o número secreto. Para isso, adicionaremos um código em else, pois é o trecho que será executado, caso o chute seja diferente do número secreto. Queremos que mensagens sejam exibidas caso o chute seja maior ou menor, e isso nos remete à condicionalidade. Sendo assim, acrescentaremos if em else. Poderemos, também, acrescentar um if dentro de outro. Desta vez a condição será um pouco diferente, pois olharemos se o chute é maior (**>**) ou menor (**<**) que o número secreto. Veja o código:

else {

if(chute > numerosecreto) {

printf("Seu chute foi maior que o número secreto\n");

}

if(chute < numerosecreto) {

printf("Seu chute foi menor que o número secreto\n");

}

Notem que:

* quando adicionamos if(), abrimos outro bloco de código por meio de chaves **{}**, sempre indentando com "Tab" para manter o código organizado e fácil para leitura;
* em cada if() foi acrescentado printf() para imprimir uma mensagem, caso o chute do usuário seja maior ou menor;
* apagamos os printf() que estavam em else para o caso de erro — com as novas condições, o jogador receberá dicas que o levem ao acerto, no lugar das mensagens que apenas comunicavam que o chute estava errado;
* para inserir if(), não é obrigatório ter else, como no if() em caso de chute menor que o número secreto;

Rodaremos o código atualizado no terminal, e a primeira tentativa será com um número menor (10):

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Alura:adivinhacao alura$

Funcionou como planejamos; ao chutarmos um número menor, receberemos como retorno uma mensagem nos indicando o erro. O próximo teste será com um número maior (70):

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 70

Seu chute foi 70

Seu chute foi maior que o número secreto

Alura:adivinhacao alura$

Também deu certo! Fomos informados de que o chute foi maior que o número secreto. Vamos testar o número secreto correto para garantir que, se acertarmos, o programa continua funcionando:

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Alura:adivinhacao alura$

É importante testar o código e garantir que ele está funcionando da maneira esperada, como fizemos, por meio de três caminhos diferentes:

* acerto do número;
* erro com número menor;
* erro com número maior.

Em resumo, vimos que:

* para inserir if(), não é obrigatório ter else;
* if(), assim como else, pode ser inserido dentro de outro;
* sinal de maior (**>**), utilizado em if(), compara se um valor é maior que outro, assim como o sinal de menor (**<**) compara se um valor é menor que outro.

Vamos mudar um pouco o código para percebermos que é possível desenvolvê-lo de diversas formas. Desta forma, desenvolveremos esse "instinto" de programador, de visualizar um código e perceber as diferentes formas de escrevê-lo. Por exemplo, a condição

if(chute == numerosecreto)

Pode ser inserida em uma variável:

int acertou = (chute == numerosecreto);

Notem que isolamos (chute == numerosecreto) entre parênteses para fazermos a expressão de igualdade e compararmos um valor com outro. Quando for necessário utilizar a condicional if(chute == numerosecreto), poderemos substituí-la pela variável acertou:

int acertou = (chute == numerosecreto);

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

A leitura do código ficou mais simples: é mais fácil lermos if(acertou), que em português seria "se acertou", do que if(chute == numerosecreto). Criar uma variável que explica a condicional facilita a legibilidade e entendimento do código. É possível entendermos o if() sem saber ou pensar a regra ou condição presente na variável acertou.

O **nome** é importante, pois é explicativo, portanto preocupem-se com a nomeação de arquivos e variáveis. De volta ao código, notem que a igualdade **==** se encontra em um inteiro. Isso quer dizer que quando estabelecemos condições com o if() — igualdade, maior ou menor — teremos como resolução 0 caso a expressão avaliada seja falsa, e 1, caso seja verdadeira.

A expressão (chute == numerosecreto) retornará um número, que guardaremos em inteiro. Acrescentaremos um printf() no editor de texto para visualizarmos o que acontece:

int acertou = (chute == numerosecreto);

printf("Acertou: %d\n", acertou)

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

Compilaremos e rodaremos o código no terminal, primeiro errando o chute:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

Acertou: 0

Seu chute foi menor que o número secreto

Alura:adivinhacao alura$

O Acertou: 0 indica que erramos o chute, indicando que o Acertou é falso. Se tentarmos com o número secreto correto, teremos 1 no retorno:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 42

Seu chute foi 42

Acertou: 1

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Alura:adivinhacao alura$

A comparação devolve um inteiro (0 ou 1), e o if() verifica se o que está na variável acertou resulta em 1 ou 0 para então executar ou não o que se encontra no bloco. Em outras linguagens, dizemos que isso é um *boolean*, que vem da álgebra de Boole e recebe apenas *true* ou *false*. É a álgebra que trabalha com verdadeiro ou falso.

Em linguagem C, falso será 0 e verdadeiro será 1. Apagaremos do código o printf("Acertou: %d\n", acertou), utilizado-o somente para demonstrar o retorno no terminal. Colocaremos a condição de maior dentro de uma variável em else, também:

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

Ao compilarmos e rodarmos, teremos no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 72

Seu chute foi 72

Seu chute foi maior que o número secreto

Alura:adivinhacao alura$

Continua funcionando! Anteriormente, vimos que para inserirmos if(), não é obrigatório ter else. No Sublime, vamos fazer um teste e trocar if(chute < numerosecreto) por else:

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

} else {

printf("Seu chute foi menor que o número secreto\n");

}

É mais fácil entendermos um else do que dois if separados. Se olharmos, escrevemos que se o chute for maior — if(maior), ele imprimirá "Seu chute foi maior que o número secreto" e else imprimirá "Seu chute foi menor que o número secreto". Vejam que, nesse caso, else **terá** que ser menor. Se os números fossem iguais, cairíamos em:

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

Se cairmos na situação abaixo, quer dizer que os números são diferentes. Sendo assim, será maior — if(maior) — **ou** menor — else {}.

else {

printf("Seu chute foi menor que o número secreto\n");

}

Vimos que o if basicamente analisa se a condição é:

* 0 = falsa, e não executa if() ou
* 1 = verdadeira, e executa o corpo do if().

A expressão que colocamos em if() pode ser colocada em uma variável do tipo inteiro, que será sempre avaliada em 0 e 1 — e por isso guardaremos em int. Deem bons nomes para variáveis se colocarem expressões dentro delas, pois fará a diferença na legibilidade do código.

Vocês devem ter percebido que declaramos variáveis em vários pontos diferentes do código: temos chute e numerosecreto, por exemplo, declaradas logo abaixo da função main, bem como a variável maior, dentro de else. É possível declararmos variáveis em qualquer lugar do programa, mas será que conseguimos acessá-las a qualquer momento?

Poderemos utilizar acertou ou chute em else, considerando que elas estão no topo do código, sendo que else é delimitado por um novo bloco de chaves? Vamos fazer o teste com acertou:

int acertou = chute == numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

printf("Acertou: %d\n", acertou);

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior do que o número secreto!\n");

} else {

printf("Seu chute foi menor do que o número secreto!\n");

}

}

Ao compilarmos e rodarmos o programa no terminal, teremos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

Acertou: 0

Seu chute foi menor que o número secreto

Alura:adivinhacao alura$

Sim, funciona! A variável acertou foi impressa corretamente. Mas será que conseguimos imprimir a variável maior antes de sua declaração no código?

print("%d", maior);

int acertou = chute == numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior do que o número secreto!\n");

} else {

printf("Seu chute foi menor do que o número secreto!\n");

}

}

Ao compilarmos, teremos um erro no terminal:

adivinhacao.c:18:15: error: use of undeclared indentifier 'maior'

printf("%d", maior);

^

1 error generated.

Alura:adivinhacao alura$

É indicado o uso de uma variável não declarada, no caso, maior. Acontecerá o mesmo se tentarmos utilizá-la após o fechamento das chaves do espaço em que foi declarada:

adivinhacao.c:34:15: error: use of undeclared indentifier 'maior'

printf("%d", maior);

^

1 error generated.

Alura:adivinhacao alura$

Faz sentido, considerando que maior só é declarada no primeiro else, então, tentar utilizá-la antes de declará-la ou após o fechamento do bloco em que foi declarada não surtirá efeito. Sempre que declararmos uma variável, ela fica disponível no **escopo** em que está, e nos escopos do bloco em que foi declarada.

"Escopo" é o nome dado aos trechos do código, situados entre as chaves (**{}**), onde determinada variável é válida após ser declarada. Declaramos a variável chute dentro do escopo da função main, portanto ela é válida dentro de toda a função e também nos escopos declarados dentro dela, como nos if() e else que temos no programa.

No entanto, a variável maior é visível somente dentro do escopo de else (delimitado pelas chaves que abrimos ao escrevê-lo), onde foi declarada, e nos escopos internos. Significa que se tentarmos usar essa variável em um escopo além de else, não conseguiremos. E se acrescentarmos printf("%dºn", maior); no escopo de else, obteremos:

int acertou = chute == numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior do que o número secreto!\n");

print("%d\n", maior);

} else {

printf("Seu chute foi menor do que o número secreto!\n");

}

}

E no terminal, ao "chutarmos" um número maior, reparem que maior apareceu:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 70

Seu chute foi 70

Seu chute foi maior que o número secreto

1

Alura:adivinhacao alura$

Notem que as variáveis só existem dentro do escopo em que foram declaradas.

No exemplo acima, o 1 é um indicador de que a condição é verdadeira.

Lembrem-se de declarar suas variáveis no local correto. Na prática, elas sempre serão declaradas acima da linha em que serão utilizadas.

Para simplificar, entenderemos as chaves (**{}**) como referência do escopo. A variável chute, por exemplo, foi declarada entre as chaves da função main(), portanto pode ser utilizada neste espaço, em qualquer linha abaixo da que foi declarada.

Do mesmo modo, a variável maior foi declarada dentro do escopo de else, e por isto existirá no espaço delimitado por chaves. A ideia é sempre **declararmos a variável em um escopo antes de utilizá-la**.

Até agora, o programa nos pergunta qual é o chute do jogador e, após receber uma resposta, diz se está correta ou errada, encerrando jogo. O próximo passo será possibilitar mais tentativas ao usuário após errar um chute. O programa deve continuar perguntando, até um número limite de tentativas. Neste momento, estabeleceremos um limite de 3 chances para o usuário acertar o número. Para isso, a ideia é repetir 3 vezes o código responsável pelo chute:

// 1a tentativa

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

if (acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

} else {

printf("Seu chute foi menor que o número secreto\n");

}

}

// 2a tentativa

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

if (acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

} else {

printf("Seu chute foi menor que o número secreto\n");

}

}

// 3a tentativa

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

if (acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

} else {

printf("Seu chute foi menor que o número secreto\n");

}

}

Teríamos que acertar alguns detalhes, mas essa ideia funciona. Porém, o código ficou imenso e repleto de repetições. Se tivermos que alterar o texto de alguma das mensagens, teremos que fazer a alteração nos três locais em que a mensagem aparece no programa.

Repetição de código é impraticável, devemos evitá-la ao máximo.

No exemplo acima, repetimos apenas três vezes, mas imaginem se quiséssemos determinar como limite 10 chances - qualquer alteração no código será muito trabalhosa. Poderemos dizer ao nosso programa para repetir trechos de código de forma automática, adicionando-se **loops**.

A primeira maneira de se escrever um loop é utilizando o for(), que usaremos por sabermos que queremos repetir o código exatamente 3 vezes. Ensinaremos o programa a comparar algo ao número limite de vezes, e para isso declararemos a variável i, e estabeleceremos que ela é igual a 1 — i = 1. O código terá que ser repetido sob uma condição: enquanto i for menor ou igual a 3 — i <= 3. E o número de tentativas crescerá de um em um, i++:

for(int i = 1; i <= 3; i++) {

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

if (acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

}

else {

int maior = chute > numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

} else {

printf("Seu chute foi menor que o número secreto\n");

}

}

}

Vamos entender o passo a passo do que acontece com este trecho de código:

for(int i = 1; i <= 3; i++)

* o for() é um loop, composto por três partes;
* declaramos a variável i, que começa valendo 1 — i = 1;
* estabelecemos que o código deve ser repetido enquanto i for menor que 3: i <= 3
  + passamos a condição que diz para o programa perguntar o chute do usuário em até 3 vezes. A partir do 4º chute, o programa deve parar de perguntar;
* informamos de quanto em quanto i cresce, com i++;
* as **três partes** do for() estão separadas por ponto e vírgula (**;**)
  + a primeira é a variável que será usada de **contador**
  + a segunda é a **condição** de parada do loop
  + a terceira é o **incremento**
* acrescentamos chaves (**{}**) após o for(), indicando um novo bloco de código, e indentamos o que ficou dentro desse bloco.

Tentaremos rodar o programa no terminal, e teremos como resultado:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Alura:adivinhacao alura$

Notem que após o terceiro chute errado, o programa parou de perguntar "Qual é o seu chute?". Poderemos aproveitar a variável i para exibirmos o número da tentativa. Assim, o jogador saberá quantos chutes ainda possui. Para isso, usaremos a variável i no printf:

for(int i = 1; i <= 3; i++) {

printf("Tentativa %d de 3\n", i);

printf("Qual é o seu chute? ");

// ...

}

Ao executarmos o programa no terminal, o resultado será o seguinte:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2 de 3

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 3 de 3

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Alura:adivinhacao alura$

Dessa forma, ensinamos o computador a repetir um código de acordo com o número de vezes que estabelecemos. Faremos o "Teste de Mesa", que é uma simulação do algoritmo:

* O for() executa o primeiro bloco, de inicialização, e declara a variável i, atribuindo o valor 1 a ela.
* Em seguida, ele valida a condição que está no bloco 2. Como 1 é menor-ou-igual a 3, a comparação retorna verdadeira.
* Como a condição é verdadeira, ele executa o bloco de código.
* Em seguida, ele executa o terceiro bloco do for(), que é o incremento. Ali, fazemos i++, e a variável passa a valer 2.
* Novamente, ele valida a condição. Como 2 é menor-ou-igual a 3, a comparação retorna verdadeira, executando-se o bloco de código.
* Novamente, ele executa o incremento, e a variável i passa a valer 3.
* Ele valida a condição; como 3 é menor-ou-igual a 3, a comparação retorna verdadeira.
* Sendo assim, o bloco de código é executado, e o incremento também.
* A variável i passa a valer 4, e a condição é validada.
* Como 4 **não** é menor-ou-igual a 3, a comparação retorna falsa, e o loop acaba.

Acrescentaremos um printf(), que informará o fim do jogo após a terceira tentativa:

printf("Fim de jogo!\n");

O programa continua, e a mensagem "Fim de jogo" é impressa:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2 de 3

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 3 de 3

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Fim de jogo!

Alura:adivinhacao alura$

Temos um problema de lógica no jogo! Mesmo se acertarmos o número, ele continuará nos pedindo mais números pra chutar, o que não deveria ocorrer:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2 de 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Tentativa 3 de 3

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Fim de jogo!

Alura:adivinhacao alura$

O jogo deve terminar assim que o usuário acertar o chute, portanto é preciso melhorarmos o loop, o qual deverá ser repetido 3 vezes **ou** até o jogador acertar o número secreto. Para isso, ao fim do if(acertou), acrescentaremos um comando para parar, a palavra break, instrução "irmã" do for().

for(int i = 1; i <= 3; i++) {

printf("Tentativa %d de 3\n", i);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

// PARAR DE EXECUTAR FOR

break;

}

else {

int maior = chute> numerosecreto;

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

} else {

printf("Seu chute foi menor que o número secreto\n");

}

}

}

Ela simplesmente para a execução do loop, passando para a próxima linha, logo após o fim do laço. É necessário quebrarmos o loop no momento em que o jogador acerta o número - ou seja, é no if() que se verifica isso. Vamos fazer o teste no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 2 de 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Alura:adivinhacao alura$

Notem que break quebrou o loop, indo direto para o fim do jogo, sem pedir a terceira tentativa. Testaremos se ele continua parando caso todas as tentativas estejam erradas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2 de 3

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 3 de 3

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Fim de jogo!

Alura:adivinhacao alura$

O break será executado somente se o jogador acertar o número secreto, pois está no escopo do if(acertou). Vimos que break pode ser utilizado para interromper o loop de duas maneiras:

* quando a condição i <= 3 é falsa;
* quando, no meio da execução do loop, é necessário pará-lo por motivo de lógica.

Definimos que o jogador terá 3 chances de adivinhar o número secreto. Se quisermos aumentar esta quantidade para 5, teríamos que fazer a alteração no laço for() e na printf(), que imprime o número de tentativas:

for(int i = 1; i <= 5; i++) {

printf("Tentativa %d de 5\n", i);

//...

}

Repare que se mudarmos em um local, teremos que procurar e mudar em todos os outros em que o número aparece, e isso exige trabalho. O programa ainda está pequeno, mas no futuro, caso se trabalhe com códigos maiores, será muito difícil ficar procurando onde aplicar as alterações manualmente. Por este motivo, procuramos definir os valores uma única vez, então poderemos guardar o valor em um lugar e reutilizá-lo. Faremos isso de forma elegante, como já vimos, adicionando uma nova variável, logo abaixo do cabeçalho do jogo:

int numerodetentativas = 3;

Após declararmos a variável, substituiremos os locais do código que estão com o número por numerodetentativas:

#include <stdio.h>

int main() {

// imprime cabecalho do jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

int numerodetentativas = 3;

int chute;

for(int i = 1; i <= numerodetentativas; i++) {

printf("Tentativa %d de %d\n", i, numerodetentativas);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

//...

}

}

Ao rodarmos o código no terminal, ele continuará funcionando com 3 tentativas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 3

Qual é o seu chute? 3

Seu chute foi 3

Seu chute foi menor que o número secreto

Tentativa 2 de 3

Qual é o seu chute? 3

Seu chute foi 3

Seu chute foi menor que o número secreto

Tentativa 3 de 3

Qual é o seu chute? 3

Seu chute foi 3

Seu chute foi menor que o número secreto

Fim de jogo!

Alura:adivinhacao alura$

Se trocarmos o número 3 por 5 na variável numerodetentativas, e rodarmos no terminal, teremos 5 tentativas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 2 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 3 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 4 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 5 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Fim de jogo!

Alura:adivinhacao alura$

Funcionou, sem termos que alterar diversos pontos do código. Porém, no caso do 5, que será um **valor fixo**, evitaremos o uso de variáveis. O ideal seria colocá-lo em uma **constante**, fixa, cuja declaração se difere das variáveis. Fora do método main(), utilizaremos a diretiva #define, pois **tudo que começa com cerquilha (#) é definido como diretiva, na linguagem C**. Chamaremos a constante de NUMERO\_DE\_TENTATIVAS, para a qual atribuiremos o valor 5:

#define NUMERO\_DE\_TENTATIVAS 5

O nome em letras maiúsculas não é obrigatório, mas é uma convenção. Cada linguagem tem a sua. No caso do C, sugere-se que:

* o nome das **variáveis** e das **funções** sejam em **letras minúsculas**;
* o nome das **constantes** sejam em letras **maiúsculas** e com espaçamento entre palavras marcado por ***underscore*** (**\_**).

A padronização dos nomes facilita a leitura para programadores que trabalham com esta linguagem e precisam entender códigos desenvolvidos por outros. Lembrem-se de alterar as variáveis numerodetentativas pelas constantes NUMERO\_DE\_TENTATIVAS no código:

#include <stdio.h>

#define NUMERO\_DE\_TENTATIVAS 5

int main() {

// imprime cabecalho do jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

int chute;

for(int i = 1; i <= NUMERO\_DE\_TENTATIVAS; i++) {

printf("Tentativa %d de %d\n", i, NUMERO\_DE\_TENTATIVAS);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

int acertou = (chute == numerosecreto);

//...

}

}

Definimos uma constante de valor fixo, que não muda. Ao compilarmos, confirmaremos que as 5 tentativas continuam:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 2 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 3 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 4 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 5 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Fim de jogo!

Alura:adivinhacao alura$

Assim, vimos que define são diretivas que nos ajudam a definir **constantes**, que nunca mudam. São ideais para cenários como o que encontramos no jogo de adivinhação, onde precisamos definir um número fixo de tentativas. Elas facilitaram a leitura do código, pela substituição dos números 3 e 5, que antes precisávamos entender o que significavam.

Enquanto desenvolvemos o código, é fácil entender a que eles se referem, mas imagine se parássemos de trabalhar nesse jogo por um tempo e tentarmos lembrar depois a que se referem. Com a constante, fica claro que os números são referente ao número de tentativas.

Lembram que discutimos que podemos extrair variáveis, como fizemos com int acertou, que facilitou a leitura e compreensão do código? O mesmo ocorre com **Números Mágicos**, os quais devem ser evitados ao máximo; eles possuem um significado importante e estão "jogados" pelo código.

É o caso do 3 e do 5. Se você mostrar esse código para outra pessoa, ela precisará ler o código inteiro para entender o significado deles. Ao passo que, se trocarmos os números pela palavra NUMERO\_DE\_TENTATIVAS, tudo fica mais claro. Portanto, coloque-os em constantes ou variáveis, analisando qual será melhor para cada caso, mas **evite utilizar Números Mágicos**, pois eles dificultam a manutenção do código.

Estudamos a utilização de if(), instrução que nos ajuda a mudar o fluxo da execução do programa, e do loop, que repete um trecho de código por determinado número de vezes. Quanto ao if(), temos duas maneiras de escrevê-los:

* **sem** else, o que faz com que a condição seja avaliada e:
  + se verdadeira, o bloco de código em seu escopo é executado;
  + se falsa, o bloco de código não é executado e nada acontece.
* **com** else, quando o programa avalia a condição de if() e:
  + se verdadeira, executa o código correspondente;
  + se falsa, executa o código de else.

Há uma terceira maneira: após o if() passar por uma condição e, se ela for falsa, há outra condição a ser testada, e caso seja falsa também, haverá uma terceira condição. Caso esta seja falsa, um caso à parte será executado. Em português, isto parece complicado, mas em C ficará mais simples. O que queremos fazer é ter várias condições seguidas e que cada uma seja testada.

Quando uma delas for verdadeira, o programa executará o bloco de código correspondente, e as outras serão ignoradas. Como a linguagem de código é sucinta, esse raciocínio será claro. Então, vamos aplicar algumas alterações:

* adicionaremos novas variáveis, maior e menor — reparem que não utilizaremos parênteses nelas, pois eles servem somente para facilitar a visualização do que será executado;
* apagaremos o else e int maior que estavam logo abaixo de break;
* trocaremos o else, abaixo de if(maior), por if(menor);
* arrumaremos os espaçamentos.

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

int menor = chute < numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

break;

}

if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

if(menor) {

printf("Seu chute foi menor que o número secreto\n");

}

}

//...

O código ficou com três if() em sequência. Vamos avaliar os três casos:

Em if(acertou), se o usuário acertou o chute (chute == numerosecreto), a condição é verdadeira. Então, o bloco de código correspondente será executado e o laço será quebrado em break.

Se if(acertou) for falso, o programa analisará a segunda condição, if(maior). Se ela for verdadeira, o bloco de código de seu escopo será executado e a mensagem de printf(), referente a ela, será impressa na tela.

O programa continuará rodando, e se if(maior) for verdadeira, o if(menor), obrigatoriamente será falso, já que não é possível ter um chute maior e menor ao mesmo tempo. Mesmo que em milésimos de segundos, gasta-se um tempo computacional desnecessário no processo de análise de condições, quando aquela que for verdadeira já foi executada.

Reescreveremos if de forma que as condições sejam avaliadas na ordem de sequência e, se uma delas for verdadeira, a próxima não será executada. Para isso, misturaremos if com else:

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

break;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

else if(menor) {

printf("Seu chute foi menor que o número secreto\n");

}

}

//...

Dessa forma, o if(acertou) será testado. Se verdadeiro, os outros não serão testados — haverá uma quebra do loop por meio do break. Se for falso, a próxima condição else if(maior) será testada, executará o bloco de código, e não testará else if(menor). O programa não pensará se a próxima condição é verdadeira (1) ou falsa (0).

Ou seja, o programa testará a sequência de if(). Sendo um deles for verdadeiro, o bloco de código será executado, e os próximos else if serão desconsiderados. Isso é *if-else-if*, vamos juntando um if no outro. Ainda poderemos trocar else if por else, simplesmente:

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

break;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

else {

printf("Seu chute foi menor que o número secreto\n");

}

}

//...

Se if(acertou) não for verdadeiro, else if(maior) será testada. Se for falsa também, não haverá outra possibilidade. Se o valor não é igual ou maior, por lógica, será menor. Portanto, nesse caso, o else funciona melhor do que else if(menor). Vimos uma terceira forma de utilizar if(), que será muito útil.

Sendo assim, as três maneiras de utilizar o if() são:

1. if() sozinho — executará o bloco de código somente se a condição for verdadeira;
2. else if() — executará o primeiro bloco de código se a condição for verdadeira, ou o segundo bloco, se a primeira for falsa;
3. if-elseif-else — procura a condição verdadeira para executar, ignorando as demais ou, se todas são falsas, executa o último bloco de código else.

Vamos rodar o programa para ver se continua funcionando?

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 2 de 5

Qual é o seu chute? 70

Seu chute foi 70

Seu chute foi maior que o número secreto

Tentativa 3 de 5

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Alura:adivinhacao alura$

O if-elseif-elseif funcionou! O benefício de utilizarmos uma condição a menos não faz diferença no tempo total de execução. O processador executa o código de forma extremamente rápida e imperceptível. É desnecessário testarmos uma segunda condição sabendo que a primeira é verdadeira. Deste modo, estamos programando e utilizando lógica de programação de maneira adequada.

O usuário pode digitar o número que quiser, portanto nada o impede de digitar um número negativo. Em todo sistema de software, é comum usuários digitarem valores inválidos, e você, programador, deve impedir que ele insira dados inválidos na aplicação, como palavras em um campo destinado a números, por exemplo.

O jogo que estamos desenvolvendo aceita somente números positivos. Assim, devemos programar uma mensagem de erro para o usuário, caso ele digite números negativos, para avisá-lo de que não perderá uma rodada se fizer isso. Isto é possível de ser feito com o que já estudamos, acrescentando-se:

* if(), verificando se o chute é menor que 0, logo após capturarmos o número do usuário por meio de scanf(), e
* printf() para imprimir uma mensagem, caso o chute seja um número negativo.

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

}

//...

No terminal, se chutarmos um número negativo (-2), teremos o seguinte retorno:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? -2

Seu chute foi -2

Você não pode chutar números negativos!

Seu chute foi menor que o número secreto

Tentativa 2 de 5

Qual é o seu chute?

Notem que uma tentativa foi descartada pelo chute com o número negativo, mas isto não é o que queremos. Lembram que o laço de repetição executa i++? De volta ao código, acrescentaremos i--, oposto do i++, para subtrair uma unidade da variável, já que essa rodada não vale.

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

i--;

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

}

//...

Testaremos o programa novamente no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? -2

Seu chute foi -2

Você não pode chutar números negativos!

Seu chute foi menor que o número secreto

Tentativa 1 de 5

Qual é o seu chute?

Funcionou! Depois do chute -2, continuamos na primeira tentativa das cinco possíveis. Só tem um problema: se o jogador chutar um número negativo, o programa avisará que isso é inválido e que o chute é menor do que o número secreto. Não queremos isso, e sim parar a execução do loop caso o número seja inválido. Podemos fazer isso com if-else:

if(chute < 0) {

printf("Você não pode chutar números negativos\n");

i--;

}

else if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bomjogador!\n");

break;

}

else if(maior) {

printf("Seu chute foi maior do que o número secreto!\n");

}

else {

printf("Seu chute foi menor do que o número secreto!\n");

}

}

Como o colocamos em primeiro caso, ele será o primeiro a ser executado. Ao compilarmos e rodarmos o programa, funcionará como gostaríamos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? -2

Seu chute foi -2

Você não pode chutar números negativos!

Tentativa 1 de 5

Qual é o seu chute?

Nenhuma tentativa foi descartada, e fomos avisados somente sobre o uso de números negativos. Mas podemos fazer isso de forma diferente: no código, por meio do atalho "Ctrl + Z", apagaremos a declaração das variáveis acertou, maior e menor, pois elas não serão necessárias se o número for negativo.

Para isso, utilizaremos continue, "irmão" de break. Percebam que essa é uma situação diferente da que usamos o break, que parava o loop. Com continue, queremos continuar a execução de for, partindo para a próxima iteração do loop.

if(chute < 0) {

printf("Você não pode chutar números negativos\n");

i--;

continue;

}

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

int menor = chute < numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

break;

}

//...

Vamos testar novamente no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1 de 5

Qual é o seu chute? -2

Seu chute foi -2

Você não pode chutar números negativos!

Tentativa 1 de 5

Qual é o seu chute? 2

Seu chute foi 2

Seu chute foi menor que o número secreto

Tentativa 2 de 5

Qual é o seu chute? 70

Seu chute foi 70

Seu chute foi maior que o número secreto

Tentativa 3 de 5

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo! Você é um bom jogador!

Fim de jogo!

Alura:adivinhacao alura$

Isto é, utilizaremos continue quando quisermos continuar a execução do loop sem quebrá-lo, como break.

Vamos mudar um pouco a regra do nosso jogo! No momento, o jogador tem até cinco tentativas, que definimos por meio da constante NUMERO\_DE\_TENTATIVAS. Se ele não acertar até a quinta chance, perde o jogo.

Estabeleceremos uma nova regra na qual as tentativas serão ilimitadas e contadas. No final do jogo, após o usuário acertar o número, uma mensagem será impressa, informando em qual tentativa o chute está correto. Temos que pensar em outro tipo de loop, porque for é um loop contado, que sabe exatamente quantas vezes será rodado.

Sua utilização é ideal quando sabemos o número de vezes que queremos repetir o código, por exemplo, quando queríamos que o número de tentativas fosse 5. Agora, não sabemos quantas tentativas serão necessárias para que o usuário acerte. Precisaremos trocar for por um loop que execute o bloco de código, **enquanto** o jogador não ganha.

Para isso:

* deletaremos for;
* declararemos a variável ganhou, atribuindo a ela o valor 0, pensando na álgebra Booleana, na qual 0 é falso e 1 é verdadeiro;
* acrescentaremos o loop while(), que em inglês, significa "enquanto": while(ganhou == 0).

Dessa forma, informaremos o programa de que enquanto o jogador **não ganhar** — isto é, ganhou for falso — deve se repetir o loop. Com a inserção de while(), teremos que pensar no funcionamento do código e aplicar algumas alterações:

* deletaremos:
  + %d e NUMERO\_DE\_TENTATIVAS do printf() que imprime o número de tentativas;
  + a constante #define NUMERO\_DE\_TENTATIVAS 5;
  + i--;
  + a variável menor, já que ela não é mais utilizada — descartem a declaração de variáveis que não serão utilizadas;
* trocaremos break por ganhou = 1, que indicará se ganhou é verdadeiro de acordo com a álgebra Booleana. Assim, quando o jogador acertar o número, o loop será interrompido, pois determinamos que o código será repetido **enquanto** ganhou for igual a 0.
  + notem que continue funciona em while() também, portanto permanece.

Acostumem-se a esse processo: programar é escrever e apagar código, de acordo com as alterações necessárias. Dito isto, o código ficará assim:

#include <stdio.h>

int main () {

//imprime cabecalho do jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int numerosecreto = 42;

int chute;

int ganhou = 0;

while(ganhou == 0) {

printf("Tentativa %d\n", i);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

continue;

}

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

ganhou = 1;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

}

printf("Fim de jogo!\n");

Ao testarmos o programa no terminal, teremos o seguinte erro:

adivinhacao.c:17:28: error: use of undeclared identifier 'i'

printf("Tentativa %d\n", i);

^

1 error generated.

Nesse caso, o erro se dá pelo uso da variável i, não declarada. Isso aconteceu porque quando exibíamos o número das tentativas, tínhamos a variável i, que agora não existe mais. Para corrigir isso, declararemos uma nova variável tentativas:

int numerosecreto = 42;

int chute;

int ganhou = 0;

int tentativas = 1;

Atribuiremos a ela o valor inicial 1, considerando que a primeira tentativa será contada como 1. Tentativa 0 faz sentido para quem programa, mas para o usuário é mais fácil iniciar a contagem de tentativas a partir de 1. Somaremos 1 à variável tentativas, no final do código:

while(1) {

printf("Tentativa %d\n", tentativas+1);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

continue;

}

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

ganhou = 1;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

tentativas = tentativas + 1;

}

printf("Fim de jogo!\n");

Testaremos o programa mais uma vez no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Tentativa 4

Qual é o seu chute? 40

Seu chute foi 40

Seu chute foi menor que o número secreto

Tentativa 5

Qual é o seu chute? 50

Seu chute foi 50

Seu chute foi maior que o número secreto

Tentativa 6

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 7

Qual é o seu chute? 43

Seu chute foi 43

Seu chute foi maior que o número secreto

Tentativa 8

Qual é o seu chute? 41

Seu chute foi 41

Seu chute foi menor que o número secreto

Tentativa 9

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Alura:adivinhacao alura$

Notem que da Tentativa 1 até Tentativa 4, os chutes foram menores que o número secreto e o programa nos avisa corretamente. Na Tentativa 5, quando chutamos um número maior, a mensagem impressa muda, avisando que o chute foi maior. As dicas nos deram um parâmetro que facilita o acerto do número secreto, então, o jogo funciona. Acrescentaremos um printf() que imprima a quantidade de tentativas até o acerto do número:

printf("Fim de jogo!\n");

printf("Você acertou em %d tentativas!\n", tentativas);

No terminal, ao compilarmos e rodarmos o programa, teremos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 4 tentativas!

Alura:adivinhacao alura$

Opa! O programa imprimiu que acertamos em 4 tentativas, mas na contagem foram 3. Isso acontece porque somamos 1 à variável tentativas, o que poderemos resolver de duas maneiras:

**1.** Subtraindo (-1) no printf():

printf("Você acertou em %d tentativas!\n", tentativas-1);

**2.** Alterando o valor 1 para 0 na declaração da variável, não subtraindo (-1) de tentativas no printf() final e somando (+1) no printf() da contagem de tentativas:

int tentativas = 0;

while(ganhou == 0) {

printf("Tentativa %d\n", tentativas+1);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

continue;

}

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

ganhou = 1;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

}

printf("Fim de jogo!\n");

printf("Você acertou em %d tentativas!", tentativas);

Das duas formas, o programa funcionará e obteremos um retorno correto:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 3 tentativas!

Alura:adivinhacao alura$

Vamos revisar alguns pontos do código:

* elaboramos um loop genérico, while(), que não sabemos quando irá parar, diferente de for, no qual definimos este exato momento;
* a condição para o loop continuar a executar o código é a variável ganhou ser igual a 0;
* criamos a variável tentativa que armazenará a quantidade de tentativas. Atribuímos a ela o valor 0 na declaração, mas a imprimimos somando um (+1) para ficar coerente para o usuário;
* no escopo de if(acertou), substituímos break por ganhou = 1;
* quando ganhou for igual a 1, será diferente de 0, e o loop será interrompido, encerrando-se o jogo com o printf() final;
* no código atual, com while(), utilizamos tentativas = tentativas + 1. Para for, utilizamos tentativas++. Ambas funcionam somando-se 1 à variável. ++ é um resumo de tentativas + 1.

Se quisermos um **loop infinito**, substituiremos ganhou = 1 por break, e não precisaremos mais da variável ganhou, e sim de um loop que rode para sempre, até que break o quebre. Para um loop rodar para frente, a condição deve ser sempre verdadeira. Faremos isso atribuindo o valor 1 para while:

int chute;

int tentativas = 0;

while(1) {

printf("Tentativa %d\n", tentativas+1);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!ºn");

continue;

}

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

break;

}

Assim, quando while() avaliar a expressão (1), o resultado será sempre verdadeiro, mantendo-o em um loop infinito. Lembrem-se que acrescentamos um break no escopo de if(acertou). Vamos testar o programa no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 2

Seu chute foi 2

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Alura:adivinhacao alura$

Reparem que acertamos na Tentativa 3, mas o programa imprime que acertamos em 2 tentativas - isso aconteceu porque quebramos o loop com break, mas a variável tentativas começa em 0:

int tentativas = 0

Se inciarmos tentativas em 1 e deletarmos (+1) do printf() da contagem de tentativas, o programa será executado da forma que planejamos:

int tentativas = 1

while(ganhou == 0) {

printf("Tentativa %d\n", tentativas);

//...

}

Ao executarmos o código no terminal, teremos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 2

Seu chute foi 2

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 3 tentativas!

Alura:adivinhacao alura$

Observem que:

* iniciamos tentativas em 1, com int tentativas = 1;
* se acertarmos, quebraremos o loop e imprimiremos 1 — verdadeiro — no final;
* se errarmos, o break não será executado e o jogo seguirá para tentativas++ em else, imprimindo as próximas tentativas até que o usuário acerte o número.

Sendo assim, estudamos:

* while(), um outro tipo de loop, que não sabemos quando irá parar;
* loops infinitos, utilizados em while(1), lembrando que é recomendado sempre tomar cuidado, encontrando-se um momento apropriado para quebrá-lo, se não o programa nunca será interrompido. Geralmente, o programa chega a um fim;
* a execução de uma mesma função por meio de códigos diferentes.

Legal! Nosso jogo está ficando cada vez melhor. Vimos na prática o que é programar, pois estudamos como fazer condicionais, adicionando if(), a qual guia os rumos do programa, e escrevemos loops para repetir trechos do programa quantas vezes forem necessárias.

Programar é isso: escrever um programa de computador e ensiná-lo a tomar decisões.

Mas ainda temos muito a aprender!

Vamos adicionar uma função no jogo: pontos. Ao final da partida, diremos ao jogador quantos pontos ele fez, para compararmos entre jogadores e sabermos quem obteve a maior pontuação. A regra será: toda partida começa com 1000 pontos e, a cada rodada, pontos serão subtraídos, até que o jogador acerte o número secreto.

A subtração funcionará da seguinte forma: suponhamos que o número secreto seja 50 e o chute do usuário seja 70. Calcularemos a diferença entre esses dois números (70 - 50), que é 20, e a dividiremos por 2, obtendo-se o valor 10, o qual equivale à pontuação a ser subtraída dos 1000 pontos iniciais. Sendo assim, nesse exemplo, após a primeira tentativa, o jogador terá 990 (1000 - 10) pontos.

Em resumo, a fórmula da regra será: 1000 - (chute - numerosecreto) / 2. No final do jogo, quando o usuário acertar o número, a quantidade de pontos será impressa. Anteriormente, estudamos como declarar variáveis, inclusive do tipo inteiro. Criamos algumas delas, como numerosecreto, chute e tentativas, e declararemos mais uma (pontos), para armazenar a pontuação inicial:

int pontos = 1000;

Veremos que é possível executar operações matemáticas com as variáveis, se quisermos:

* multiplicar com asterisco (**\***) — pontos \* 2;
* dividir com barra (**/**) — pontos / 2;
* somar com o sinal de soma (**+**) — pontos + 2;
* subtrair com o sinal de subtração (-) — pontos - 2;
* criar expressões, como na escola, acrescentando-se parênteses — (pontos - 2) \* 7 / 3.

Poderemos guardar os resultados das operações em uma na variável pontos ou criar novospontos:

int pontos = pontos \* 2;

Também é possível adicionar variáveis nas operações:

pontos = (pontos - 2) \* tentativas / 3;

Voltando ao código, no final do loop acrescentaremos:

* a contagem de pontos, por meio da variável pontosperdidos;
* printf() para imprimir o total de pontos.

int chute;

int tentativas = 1;

int pontos = 1000;

while(1) {

printf("Tentativa %d\n", tentativas);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

continue;

}

int acertou = (chute == numerosecreto);

int maior = chute > numerosecreto;

if(acertou) {

printf("Parabéns! Você acertou!\n");

printf("Jogue de novo, você é um bom jogador!\n");

break;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

else {

printf("Seu chute foi menor que o número secreto\n");

}

tentativas++;

int pontosperdidos = (chute - numerosecreto) / 2;

pontos = pontos - pontosperdidos;

}

printf("Fim de jogo!\n");

printf("Você acertou em %d tentativas!\n", tentativas);

printf("Total de pontos: %d\n", pontos);

}

Notem a importância de parênteses na operação da variável pontosperdidos, sem os quais o resultado da operação seria outro. numerosecreto seria primeiro dividido por dois, e depois subtraído de chute. Com os parênteses, especificaremos que a diferença entre chute e numerosecreto deve ser calculada antes da divisão por dois.

Recapitulando o que adicionamos no código:

* declaramos a variável pontos, com valor inicial 1000;
* ao final de cada rodada, a diferença entre chute e numerosecreto é calculada e dividida por 2 e, em seguida, o resultado da operação é subtraído da pontuação inicial;
* no fim do jogo, o total de pontos é impresso por meio de printf(), inserido fora do loop.

Vamos testar no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 60

Seu chute foi 60

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 50

Seu chute foi 50

Seu chute foi maior que o número secreto

Tentativa 3

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 3 tentativas!

Total de pontos: 987

Alura:adivinhacao alura$

O número secreto é 42. Se aplicarmos a operação na primeira tentativa, teremos: (60 - 42) / 2 = 9. Em seguida, chutaremos 50, e com a operação, teremos: (50 - 42) / 2 = 4. Ao somarmos esse resultado ao anterior, 9, obteremos 13, que subtraído da pontuação inicial 1000, nos trará o mesmo número do total de pontos: 987 - a regra funciona e calcula o total de pontos da maneira que planejamos.

Sempre que utilizamos uma variável, ela já foi inicializada. Atribuímos valores a elas antes de executá-las em uma operação ou adicioná-las em um printf(), seja na mesma linha, como em numerosecreto ou captando-as dos usuários por meio de scanf(), e em ambos os casos elas estavam inicializadas.

Vamos fazer um teste para descobrir o que há em uma variável antes de inicializá-la, acrescentando printf() na variável chute, declarada no início do código, e que será inicializada em scanf().

int chute;

printf("chute %d\n", chute)

Ao compilarmos e rodarmos o código no terminal, teremos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

chute 32767

Tentativa 1

Qual é o seu chute? ^Z

[5]+ Stopped ./adivinhacao.out

Alura:adivinhacao alura$

Aparece o número 32767, que não sabemos a que se refere. É difícil saber o que uma variável armazena quando não está inicializada - mas isso depende do compilador. Em alguns deles, quando não atribuímos valor à variável, será armazenado o que estiver na região de memória do programa anterior.

Ou seja, ao declarar uma variável, o computador cria um espaço de memória que, provavelmente, foi utilizado para guardar algum dado em outro programa anteriormente. É comum desconhecermos valores de variáveis, que podem ser dados da memória de um programa anterior.

O GCC do Mac, por exemplo, guarda valores padrões. No teste que fizemos, o inteiro sempre inicializará como 32767. Em algumas linguagens, como Java, o código não seria compilado, e o programador seria avisado de que a variável não foi inicializada. Sem saber o que a variável armazena, não saberemos como o programa irá se comportar. Portanto, é recomendado utilizar somente variáveis já inicializadas, seja em linha, em operação ou em scanf().

**Atenção**: em C, é preciso prestarmos atenção, pois o compilador não avisa, permitindo o uso de variável não inicializada. Portanto, sempre atribua o valor que deseja à variável antes de utilizá-la.

A ideia da pontuação é ótima, pois permite que o jogador saiba como foi a sua performance durante o jogo, se os chutes foram bons ou não — quanto mais pontos, melhores foram os chutes, e há possibilidade de competição entre jogadores. No entanto temos um problema, um bug, que veremos executando o jogo no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 999

Alura:adivinhacao alura$

Notem que o total de pontos está errado. Se calcularmos os pontos perdidos na primeira tentativa, teremos: 1000 - (45 - 42) / 2 = 998,5, enquanto o total de pontos deveria ser 998,5 e não 999. O programa fez a divisão com números inteiros, desconsiderando casas decimais e ignorando a perda de 0,5 pontos.

Temos que corrigir essa falha, para que a casa decimal seja considerada na divisão e o programa apresente o total de pontos de forma precisa quando a diferença entre chute e número secreto for ímpar. Analisaremos o código para entender o que aconteceu:

int numerosecreto = 42;

int chute;

int tentativas = 1;

int pontos = 1000;

while(1) {

printf("Tenativa %d\n", tentativas);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("seu chute foi %d\n", chute);

if(chue < 0) {

printf("Você não pode chutar números negativos!\n");

continue;

}

Notem que declaramos variáveis do tipo inteiro (int) que, como o nome já diz, guardam números inteiros, sem casas decimais. Sendo assim, a conta está incorreta no programa porque especificamos que ela deve ser feita com números inteiros. Quando surgir uma conta com casas decimais, a vírgula será ignorada, como no teste que fizemos e a divisão de 3 por 2 resultou em 1, não em 1,5.

Utilizamos int para declarar variáveis de números inteiros, mas precisaremos explorar outros tipos, como double, que utilizaremos para guardar operações com números decimais. Sendo assim, trocaremos int da declaração da variável pontos por double:

double pontos = 1000

Testaremos a aplicação no terminal, e o retorno será:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

adivinhacao.c:55:34: warning: format specifies type 'int' but the argument has a type 'double'

[-Wformat]

printf("Total de pontos: %d\n", pontos);

~~ ^~~~~~

%f

1 warning generated.

Alura:adivinhacao alura$

Recebemos um aviso de que o formato especifica inteiro (int), mas o parâmetro da variável é double, tornando %d inválido. Estávamos imprimindo pontos com a máscara %d, que é de números inteiros. A máscara de double é %f. Temos que alterar a máscara em printf() do total de pontos:

printf("Total de pontos: %f\n", pontos)

Notem que o aviso sugere essa troca. Testaremos no terminal novamente:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 999.000000

Alura:adivinhacao alura$

Melhorou um pouco! O total de pontos está armazenado em um tipo de variável que suporta números decimais, mas a conta continua errada. De volta ao código, notem que na declaração de pontosperdidos, em que a operação é feita, estamos guardando o resultado como número inteiro (int). Devemos trocá-lo por double:

double pontosperdidos = (chute - numerosecreto) / 2;

pontos = pontos - pontosperdidos;

Verificaremos no terminal se funciona:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 999.000000

Alura:adivinhacao alura$

Não mudou; o total de pontos continua sendo 999.000000. Verificaremos o código mais uma vez, pensando como o compilador. Está correto declararmos a variável pontosperdidos como double, afinal queremos que o resultado seja exibido como número decimal. No entanto, o compilador lê o código da direita para a esquerda, resolvendo primeiramente (chute - numerosecreto) / 2 e depois declarando a variável, armazenando seu resultado.

Ou seja, quando a operação é calculada, ainda não se sabe onde o resultado será armazenado e, feito isto, constata-se que chute e numerosecreto são variáveis do tipo inteiro, assim como o divisor 2. Logo, (chute - numerosecreto) / 2 será feito como se fosse int, e guarda um resultado inteiro em double, perdendo a precisão da casa decimal.

Uma forma de obtermos o resultado que desejamos é inserindo uma vírgula (**,**) no divisor: 2.0. Observem que no código devemos utilizar ponto (**.**) no lugar de vírgula, em números decimais:

double pontosperdidos = (chute - numerosecreto) / 2.0;

pontos = pontos - pontosperdidos;

Assim, a operação acontece com um ponto flutuante, (chute - numerosecreto) / 2.0, e o resultado será armazenado em uma variável que o suporta: double pontosperdidos. Mais uma vez, compilaremos e rodaremos no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 998.500000

Alura:adivinhacao alura$

Funcionou! Estudamos outro tipo de variável, o double, que é utilizado para operações com casas decimais.

Antes de continuarmos, faremos um ajuste de interface. Reparem que no total de pontos, 998.500000, há muitos zeros (0) desnecessários. Se pensarmos bem, os resultados nunca terão mais de uma casa decimal, pois os números são inteiros e divididos por 2. Assim, a variação será sempre entre .0 e .5. Uma casa decimal será suficiente para o jogo.

Para corrigir o excesso de números na máscara %f, acrescentaremos .1 entre o sinal de porcentagem (%) e f:

printf("Total de pontos: %.1f\n", pontos)

.1 especifica o número de casas decimais que desejamos exibir. Vamos testar no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 998.5

Alura:adivinhacao alura$

Vejam que funcionou, e o número de casas decimais exibidas correspondem ao número especificado na máscara, entre % e f. Se trocarmos .1 por .2, o que ocorrerá é o seguinte:

printf("Total de pontos: %.1f\n", pontos)

Desse modo, duas casas decimais serão impressas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 998.50

Alura:adivinhacao alura$

Lembrando que esse raciocínio funciona com variáveis do tipo double, e não com as do tipo int, que não possuem casas decimais.

Trabalhamos com dois tipos de variáveis: int, para guardar números inteiros e double, para os números decimais, e ambas possuem tamanho definido. Por exemplo, int utiliza 4 bytes ou 32 bits, considerando que 1 byte equivale a 8 bits, para armazenar um número.

Não precisaremos nos aprofundar em aritmética binária, mas precisaremos entender que int armazena aproximadamente 2³² números diferentes. É muito, porém pode ser insuficiente, dependendo do tipo de aplicação.

A capacidade de armazenamento de double é de 8 bytes. No entanto, não precisaremos utilizá-la se quisermos guardar um número inteiro que seja maior que 2³² em int. Existem outras formas de declararmos variáveis no intervalo entre estas. Nesse caso, poderemos utilizar long, "irmão" de int, por exemplo.

Trata-se de um inteiro longo, que utiliza 8 bytes (64 bits) para armazenar números. Ou seja, cerca de 2⁶⁴ números distintos. Pode parecer que nunca será necessário armazenar-se um número maior que 2³², mas contarei uma história para vocês: em 2014, o clipe da música *Gangnam Style*, do coreano *Psy*, usava inteiros para guardar o número de vezes que um vídeo foi visto no Youtube. Porém, o clipe viralizou, sendo o primeiro a ultrapassar o limite dos inteiros em visualizações, e a alteração para long foi necessária. Existem outros casos famosos como esse.

No Twitter, tweets — frases pequenas — são postados e enumerados a cada publicação. Em um dado momento, o número de tweets ultrapassou 2³², e a alteração de int para long também se fez necessária. Um inteiro, representado por int, possui 4 bytes, enquanto long suporta inteiros de 8 bytes.

Há também o "irmão" menor de int, o short, que utiliza 2 bytes. Dependendo da plataforma utilizada, a escolha do tipo de variável é importante. Variáveis econômicas, em termos de espaço, podem fazer a diferença no desenvolvimento de dispositivos pequenos e com memória limitada. Analisem os tipos de variáveis para escolherem a mais adequada.

double também possui uma opção para armazenamento de números decimais menores: float, que utiliza 4 bytes. Em resumo, sem entrar no mérito da aritmética binária, vimos novos tipos de variáveis relacionados às que já estudamos:

**Inteiros:**

* short = 2 bytes
* int= 4 bytes = 32 bits ≅ 2³²
* long = 8 bytes = 64 bits ≅2⁶⁴

**Ponto Flutuante:**

* float = 4 bytes
* double = 8 bytes = 64 bits

Assim, temos variáveis com diferentes capacidades de armazenamento, que podem ser escolhidas de acordo com a memória disponível.

Trabalharemos um detalhe do 2.0, que acrescentamos anteriormente na declaração de pontosperdidos para que o código funcionasse da maneira planejada.

double pontosperdidos = (chute - numerosecreto) / 2.0;

Antes de adicionarmos .0 a 2, a operação de double desconsiderava casas decimais, fornecendo um resultado inteiro, diferente do que precisávamos. Discutiremos outra maneira de transformar um número inteiro em double, criando um arquivo em C à parte, no Sublime, para não misturar no código do jogo:

#include <stdio.h>

int main() {

double pontos = 3 / 2;

printf("%f\n", pontos);

}

Salvaremos como teste.c na pasta adivinhacao, que criamos para o jogo. Se testarmos no terminal, teremos:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

1.000000

Alura:adivinhacao alura$

Teremos o mesmo problema, isto é, o resultado da divisão de 3 / 2, que é 1.5, não é exibido com a casa decimal (0.5) e o resultado impresso é 1.000000. Sabemos que a leitura do código é feita da direita para a esquerda, então a operação 3 / 2 é calculada antes de se verificar o tipo da variável. Como a conta é composta por dois números inteiros (3 e 2), a divisão é considerada inteira e a casa decimal é "trancada", independente de a variável estar declarada como double.

Para corrigir isto, anteriormente vimos que basta converter **um** dos números para double, acrescentando-se .0. Se o aplicarmos no divisor da operação, por exemplo:

#include <stdio.h>

int main() {

double pontos = 3 / 2.0;

printf("%f\n", pontos);

}

Ao compilarmos e rodarmos no terminal, veremos que o código funciona:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

1.500000

Alura:adivinhacao alura$

Criaremos a variável b para confirmar o que acabamos de testar:

#include <stdio.h>

int main() {

double b = 2.0;

double pontos = 3 / b;

printf("%f\n", pontos);

}

A variável b possui valor 2.0, e será divisora de 3, número inteiro na operação contida em pontos. No terminal, então, teremos:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

1.500000

Alura:adivinhacao alura$

O resultado permanece correto, e faremos outro teste adicionando uma variável do tipo inteiro para o dividendo 3 e transformando double, de b, em int:

#include <stdio.h>

int main() {

int a = 3

int b = 2;

double pontos = a / b;

printf("%f\n", pontos);

}

Notem que colocar .0 após a ou b não teria efeito, pois inserimos os valores em variáveis. Agora, para convertê-las em double sem alterarmos a declaração int delas, precisaremos aplicar a conversão casting, com o tipo especificado entre parênteses, antes da variável. Observem b na declaração de pontos a seguir:

#include <stdio.h>

int main() {

int a = 3

int b = 2;

double pontos = 3 / (double)b;

printf("%f\n", pontos);

}

Assim, b será interpretado como double na linha em que adicionamos o parênteses com a especificação. No terminal, ao executarmos, veremos que funciona como esperado:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

1.500000

Alura:adivinhacao alura$

Especificar o tipo antes de utilizar a variável é o casting, útil em casos como o que trabalhamos agora. Poderíamos adicionar antes de a também, e o resultado permaneceria double. Se dividirmos int por double, o resultado será double. Portanto, se dividirmos double por double, o resultado também será double. O uso de casting é vantajoso por ser simples e fazer conversões de todos os tipos para qualquer outro. Porém, temos que tomar cuidado com as conversões, como no caso a seguir:

double pi = 3.1415;

int piconvertido = (int)pi;

É possível armazenar números decimais em pi, declaradas como double. No entanto, se adicionarmos um inteiro piconvertido e fizermos um casting de pi — que é double — e utilizarmos as máscaras % f e %d para imprimir as variáveis declaradas, ao compilarmos e rodarmos o programa no terminal, teremos:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

1.500000

3.141500 3

Alura:adivinhacao alura$

Percebam que pi é uma variável double, com maior capacidade para guardar informações que int, que é pequeno e suporta somente numerais inteiros. É como se tivéssemos uma caixa grande armazenando double inserindo-a em outra menor (int). Ou seja, quebraríamos a caixa pequena e perderíamos o conteúdo da grande.

É o que acontece no exemplo acima, pois quando aplicamos casting, de double, para int, perdemos informação. Por isso, no terminal, 3.1415 é exibido como 3. Pontanto, **cuidado ao utilizarem casting**, porque se aplicarem conversões de um tipo maior para um com menor suporte de informação, como no caso de double para int ou long para int, dados serão perdidos.

casting é vantajoso em alguns casos, mas exige atenção. Alguns compiladores, inclusive os de linguagem C, não avisam quando perdemos dados em função de uma conversão mal aplicada. Adicionaremos casting no código do jogo e, de volta ao arquivo adivinhacao.c, substituiremos 2.0 por (double)2:

double pontosperdidos = (chute - numerosecreto) / (double)2;

pontos = pontos - pontosperdidos;

E, ao testarmos no terminal, o retorno continuará correto:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 45

Seu chute foi 45

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 998.5

Alura:adivinhacao alura$

O código também funcionará corretamente no terminal, ao aplicarmos casting antes de (chute - numerosecreto). Assim, vimos que casting é a conversão de tipos de variáveis por meio da especificação, entre parênteses, antes de utilizarmos uma variável ou um número.

Ainda temos um pequeno problema de pontuação para resolver... Programar exige análise de casos particulares, como o que veremos a seguir. Vamos compilar e rodar o programa:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 1006.0

Alura:adivinhacao alura$

Reparem no total de pontos: 1006.0. Estamos "enganando" o jogo, pois o máximo de pontos é 1000, dos quais seria feita uma subtração a cada tentativa, certo? O total de pontos ficou acima de 1000 porque, antes, os chutes eram **maiores** — 45, 50, 60 — que o número secreto (42), o que tornava os resultados da conta (chute - numerosecreto) sempre positivos.

Agora que chutamos um número menor (30), a diferença entre chute e numerosecreto fica negativa (-12). O resultado divido por 2 é igual a -6, e na linha abaixo da declaração de pontosperdidos, estabelecemos a conta:

pontos = pontos - pontos perdidos;

Ao subtrairmos um número negativo, esta operação passa a ser soma, isto é, foram somados 6 pontos à pontuação inicial. Precisaremos ajustar o código para que quando os chutes forem menores que o número secreto, pontos sejam subtraídos, e não somados. Teremos que olhar a diferença de um número para outro, desconsiderando se ela é positiva ou negativa. Pode-se resolver essa questão adicionando if() abaixo da declaração de pontosperdidos:

double pontosperdidos = (double) (chute - numerosecreto) / (double)2;

if(pontosperdidos < 0) {

pontosperdidos = pontosperdidos \* -1;

}

pontos = pontos - pontosperdidos;

Assim, se o resultado for negativo, ao ser multiplicado por -1, ele se tornará positivo. Como estabelecemos a condição em if(), a multiplicação só será aplicada se o resultado da subtração for negativo (< 0). Vamos testar no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 994.0

Alura:adivinhacao alura$

A diferença 6, que antes era somada à pontuação inicial, agora foi subtraída, fornecendo o resultado correto: 994. É possível aplicarmos essa correção de outra maneira; estudamos funções como printf() para imprimirmos na tela, e scanf() para captar o que o usuário digita no teclado.

Existem outras funções, e a que veremos agora converte números negativos em positivos, transformando-os em absolutos, ou seja, independentes de sinal. Para isso, utilizaremos a função abs(), que testaremos no arquivo teste.c, criada anteriormente para visualizar o funcionamento de uma função, sem prejudicar o código do jogo. Declararemos as variáveis dos tipos inteiros a e b, e vamos imprimi-las:

#include <stdio.h>

int main() {

int a = 3;

int b = -3;

printf("%d %d\n", a, b);

}

No terminal, teremos:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

3 -3

Alura:adivinhacao alura$

Agora, aplicaremos a função abs(), antes dos valores das variáveis:

#include <stdio.h>

#include <stdlib.h>

int main() {

int a = abs(3);

int b = abs(-3);

printf("%d %d\n", a, b);

}

Notem que para adicionarmos abs() tivemos que acrescentar #include <stdlib.h>, referente à standard library, local em que a função está definida. Testaremos no terminal para ver como ficou, e esperamos que a permaneça sendo 3 e que b seja convertido de -3 para 3:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

3 3

Alura:adivinhacao alura$

Funcionou! abs pode ser utilizado em qualquer lugar, por exemplo, poderíamos retirar os que adicionamos em a e b, e declarar uma nova variável c:

#include <stdio.h>

#include <stdlib.h>

int main() {

int a = 3;

int b = -3;

int c = abs(a \* b);

printf("%d\n", c);

}

Multiplicando a por b, o resultado esperado é 9. Vamos verificar no terminal:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

9

Alura:adivinhacao alura$

Está correto, e se tirarmos abs() de c, o resultado fica negativo:

Alura:adivinhacao alura$ gcc teste.c -o teste.out

Alura:adivinhacao alura$ ./teste.out

-9

Alura:adivinhacao alura$

Quer dizer que por meio de abs() convertemos números negativos em positivos. A vantagem de utilizá-la é a simplicidade na hora de alterarmos o código. Reparem a seguir que, com abs(), fizemos menos alterações do que com o if() inserido anteriormente:

double pontosperdidos = abs(chute - numerosecreto) / (double)2;

pontos = pontos - pontosperdidos;

Assim, após a subtração, o número se transforma em absoluto e a divisão por 2 acontece naturalmente. A leitura do código ficou mais fácil, certo? Lembrem-se que ao inserirmos abs(), adicionamos #include <stdlib.h> no início do código, para que o compilador encontre a função. Vamos voltar ao terminal para testar:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 42

Seu chute foi 42

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 2 tentativas!

Total de pontos: 994.0

Alura:adivinhacao alura$

Tudo está funcionando da maneira que planejamos. Estudamos a função abs(), que fica em em outro Header File (<stdlib.h>) e converte números negativos para positivos. Vimos também que a conversão poderia ser feita por if(), mas abs() é mais prático. Agora, a parte de pontuação do jogo ficou perfeita!

Notem que o jogo fica chato a partir da segunda partida, porque o número secreto é sempre o mesmo (42). Deixaremos ele mais legal fazendo com que o número secreto seja **randômico**, isto é, diferente a cada partida. Precisaremos pedir para o computador pensar em um número e devolvê-lo de forma aleatória, e para isso basta aplicarmos a função rand(), de *random*, ou "randômico" em inglês.

Essa função nos devolve um inteiro, que podemos guardar em uma variável para utilizar onde for necessário. Antes de aplicar no código do jogo, testaremos no arquivo teste.c:

#include <stdio.h>

#include <stdlib.h>

int main() {

int n1 = rand();

int n2 = rand();

printf("%d %d\n", n1, n2);

}

Guardaremos a função rand() nas variáveis n1 e n2 para garantir que teremos números distintos. Por meio de printf(), imprimiremos o que acontece com o acréscimo de rand(). Dessa forma, chamaremos o método rand() duas vezes e pediremos a impressão na tela. Ao executarmos no terminal, teremos:

Alura:adivinhacao alura$ gcc teste.c -o test.out

Alura:adivinhacao alura$ ./teste.out

16807 282475249

Alura:adivinhacao alura$

Notem que com rand(), obtivemos números aleatórios e grandes — 16807 e 282475249 —, que poderiam ser pequenos. Não sabemos quais serão, porque são randômicos. De volta ao código do jogo, renomearemos a variável que armazena rand(), trocando n1 por numerogrande.

Definiremos um limite, estabelecendo que seja um número de 0 a 99, pois adivinhar um com dois dígitos é mais fácil do que um com nove. Para tal, utilizaremos uma operação matemática, calculando o resto da divisão inteira. Se pegarmos um número qualquer para dividi-lo por 5, teremos 5 opções de resto: 0, 1, 2, 3 e 4. Considerando-se que estamos fazendo uma operação com números inteiros, sem lidar com casas decimais.

Seguindo esse raciocínio, se queremos um número de 0 a 99, basta dividirmos numerogrande por 100. Ao fazermos isto, saberemos que o resto da divisão será algum número deste intervalo. A barra (**/**) é utilizada para divisão, e para o cálculo do **resto da divisão**, em adivinhacao.c, utilizaremos o sinal de porcentagem (**%**).

int numerogrande = rand();

int numerosecreto = numerogrande % 100;

int chute;

int tentativas = 1;

double pontos = 1000;

Assim, calcularemos o resto da divisão por 100, de forma que o retorno será sempre um número entre 0 e 99. Antes de continuarmos, analisaremos a execução de teste.c no terminal:

Alura:adivinhacao alura$ ./teste.out

16807 282475249

Alura:adivinhacao alura$

Alura:adicvinhacao alura$ ./teste.out

16807 282475249

Alura:adivinhacao alura$

Alura:adicvinhacao alura$ ./teste.out

16807 282475249

Alura:adivinhacao alura$

Tivemos os mesmos números três vezes seguidas, o que indica que rand() não está nos devolvendo números randômicos. Não alteramos o que queríamos acrescentando a função. Calculando o resto da divisão desses números por 100, os números permanecerão os mesmos, e o jogo permanecerá previsível.

A rand() não está devolvendo números diferentes porque computadores não são capazes de gerar números 100% randômicos. Dizemos que eles são **pseudo-randômicos**. Normalmente, as implementações de funções que devolvem números aleatórios utilizam alguma operação matemática complexa para gerar um número imprevisível.

Para garantirmos então que essa operação matemática usada por rand() seja diferente a cada execução, precisaremos definir uma "semente" diferente a cada execução do nosso programa. Porém, desta forma entramos em um paradoxo, caso precisemos fazer com que a "semente" seja aleatória ao passo que não conseguimos gerar números randômicos.

Para resolver isto, utilizaremos como semente uma função da linguagem C, que devolve o número de segundos passados desde o dia 01 de janeiro de 1970. Em computação, essa data é chamada de **EPOCH** e é importante por ser utilizada em muitas linguagens de programação para fazer contas, pois mesmo com um número extenso de segundos, basta fazer uma conta baseada nela para descobrir em qual data está.

Em C, a função time(0) devolve exatamente o número de segundos passados desde 01 de janeiro de 1970 até a data atual. Essa contagem de segundos será interessante para o jogo, porque sempre muda. Se rodarmos o jogo agora, teremos um número, e se rodarmos um tempo depois, teremos outro. Utilizaremos ele como semente para a função matemática de rand(), por meio de srand():

#include <stdio.h>

#include <stdlib.h>

int main() {

int segundos = time(0);

srand(segundos);

int n1 = rand();

int n2 = rand();

printf("%d %d\n", n1, n2);

}

Assim, sempre que rodarmos o programa, teremos o número de segundos do momento atual, que será sempre diferente, e passaremos como semente para rand(), que sempre utilizará uma função matemática diferente, produzindo resultados diferentes também. Ao rodarmos teste.c no terminal, teremos um aviso de retorno:

teste.c:6:17: warning: implicit declaration of function 'time' is invalid in c99

[-Wimplicit-function-declaration]

int segundos = time(0);

^

1 warning generated.

Alura:adivinhacao alura$

Ele indica que precisaremos incluir a função time(0). Faremos isso por meio da inclusão de #include:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

Faz sentido uma função que lida com data e hora estar em um arquivo que se chama <time.h>. Compilaremos novamente no terminal, para testarmos se funciona corretamente:

Alura:adivinhacao alura$ gcc teste.c -o test.out

Alura:adivinhacao alura$ ./teste.out

2058474850 825250780

Alura:adicvinhacao alura$ ./teste.out

2058525271 1672676527

Alura:adicvinhacao alura$ ./teste.out

2058575692 372618627

Alura:adivinhacao alura$

Funcionou! Obtivemos números diferentes cada vez que rodamos o código, porque agora estamos passando sementes (srand()) diferentes o tempo inteiro para a função rand(). Para gerarmos números randômicos de verdade, pegaremos os segundos atuais, um número que nunca se repete, considerando que sempre serão maiores do que foram no momento anterior, sempre diferentes. Acrescentaremos srand() ao código do jogo:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

//imprime cabecalho do jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int segundos = time(0);

srand(segundos);

int numerogrande = rand();

int numerosecreto = numerogrande % 100;

int chute;

int tentativas = 1;

double pontos = 1000;

Não podemos nos esquecer de importar <time.h>, necessário para que time(0) funcione. O código está pronto; notem que geramos uma semente diferente (srand()) para o mecanismo de números randômicos com base nos segundos atuais (segundos), e utilizamos rand() para gerar um número aleatório. Para evitar que o número seja extenso, calcularemos o resto (**%**) da divisão inteira por 100 (numerosecreto), que fará com que o retorno seja um número entre 0 e 99.

Compilaremos e rodaremos o jogo no terminal novamente para testá-lo:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 50

Seu chute foi 50

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 25

Seu chute foi 25

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 35

Seu chute foi 35

Seu chute foi menor que o número secreto

Tentativa 4

Qual é o seu chute? 40

Seu chute foi 40

Seu chute foi maior que o número secreto

Tentativa 5

Qual é o seu chute? 37

Seu chute foi 37

Seu chute foi menor que o número secreto

Tentativa 6

Qual é o seu chute? 38

Seu chute foi 38

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 6 tentativas!

Total de pontos: 984.5

Alura:adivinhacao alura$

Funcionou, mas rodaremos novamente para testar se teremos um número diferente do anterior 38:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 50

Seu chute foi 50

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi maior que o número secreto

Tentativa 3

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi maior que o número secreto

Tentativa 4

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi maior que o número secreto

Tentativa 5

Qual é o seu chute? 5

Seu chute foi 5

Seu chute foi menor que o número secreto

Tentativa 6

Qual é o seu chute? 6

Seu chute foi 6

Parabéns! Você acertou!

Jogue de novo, você é um bom jogador!

Fim de jogo!

Você acertou em 6 tentativas!

Total de pontos: 956.5

Alura:adivinhacao alura$

O número secreto mudou, e se rodarmos de novo, será gerado outro, já que utilizamos a função rand() que nos fornece números randômicos.

Resumindo, vimos:

* como gerar números randômicos;
* a função rand() que devolve inteiros extensos;
* o resto da divisão inteira, que utilizamos como truque matemático para que seja entre 0 e 99;
* que a função rand() é pseudo-randômica e necessita de uma operação matemática para ser executada da maneira que queremos;
* que é necessário mudar a semente - o "s" em srand() vem de *seed*, "semente" em inglês;
* que números diferentes em srand() fazem com que a função matemática sempre devolva números distintos;
* a utilização de segundos do momento atual para a função, pois considerando que o número de segundos é sempre diferente, os números calculados com base neles também serão.

Agora o jogo realmente é de adivinhação, porque a máquina pensa em números. Estamos quase finalizando o jogo!

Antes de terminarmos o jogo, faremos uma alteração na regra do número de tentativas. No início do desenvolvimento do jogo, tínhamos um número limite de tentativas, que depois foi trocado por um loop infinito, permitindo que o jogador chutasse até acertar o número. Vamos voltar a limitar o número de tentativas, e para isso

* declararemos uma variável numerodetentativas, iniciando com o número 5;
* trocaremos while(1) por for;
* quebraremos o loop;
* passaremos a declaração de acertou para fora do loop;
* transferiremos a mensagem de parabéns em if(acertou) para depois do loop:
* printf() de fim de jogo será movido para cima de if(acertou).

O código ficará assim:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

//imprime cabecalho do jogo

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\* Bem-vindo ao nosso jogo de adivinhação \*");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int segundos = time(0);

srand(segundos);

int numerogrande = rand();

int numerosecreto = numerogrande % 100;

int chute;

int tentativas = 1;

double pontos = 1000;

int acertou = 0;

int numerodetentativas = 5;

for(int i = 1; i <= numerodetentativas; i++) {

printf("Tentativa %d\n", tentativas);

printf("Qual é o seu chute? ");

scanf("%d", &chute);

printf("Seu chute foi %d\n", chute);

if(chute < 0) {

printf("Você não pode chutar números negativos!\n");

continue;

}

else if(maior) {

printf("Seu chute foi maior que o número secreto\n");

}

else {

printf("Seu chute foi menor que o número secreto\n");

}

tentativas++;

double pontosperdidos = abs(chute - numerosecreto) / (double)2;

pontos = pontos - pontosperdidos;

}

printf("Fim de jogo!\n");

if(acertou) {

printf("Você ganhou!\n");

printf("Você acertou em %d tentativas!\n", tentativas);

printf("Total de pontos: %.1f\n", pontos);

} else {

printf("Você perdeu! Tente de novo!\n");

}

}

Em resumo:

* declaramos a variável numerodetentativas;
* aplicamos um for convencional, em que i varia de 1 até menor ou igual (**<=**) número de tentativas;
* declaramos a variável acertou em um escopo maior, pois a utilizaremos no final do código, depois do loop;
* o loop roda, comparando os chutes para que em caso de acerto se quebre o loop, indo direto ao fim do jogo, e imprimindo a mensagem de acerto.

Vamos compilar e rodar o novo código no terminal:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 40

Seu chute foi 40

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

Tentativa 4

Qual é o seu chute? 20

Seu chute foi 20

Seu chute foi menor que o número secreto

Tentativa 5

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

Perdemos na quinta tentativa. Vamos tentar ganhar agora:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tentativa 1

Qual é o seu chute? 50

Seu chute foi 50

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 75

Seu chute foi 75

Seu chute foi maior que o número secreto

Tentativa 3

Qual é o seu chute? 65

Seu chute foi 65

Seu chute foi maior que o número secreto

Tentativa 4

Qual é o seu chute? 60

Seu chute foi 60

Seu chute foi maior que o número secreto

Tentativa 5

Qual é o seu chute? 56

Seu chute foi 56

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

Não conseguimos ganhar, mesmo seguindo as dicas. O jogo está difícil, mas está funcionando. Podemos alterar o código do jogo para que o usuário tenha a opção de escolher em que nível quer jogar: fácil, médio ou difícil. Para isso, antes do loop:

* declararemos a variável nível;
* acrescentaremos printf() que:
  + questiona qual nível de dificuldade o usuário deseja;
  + exibe os níveis possíveis;
  + mostra a escolha do jogador;
* scanf() que capta a escolha do jogador.
* deletaremos 5 da variável numerodetentativas;
* acrescentaremos if() e else para condicionar os níveis de dificuldade.

No Sublime, esse trecho do código ficará assim:

int acertou = 0;

int nivel;

printf("Qual o nível de dificuldade?\n");

printf("(1) Fácil (2) Médio (3) Difícil\n\n");

printf("Escolha: ");

scanf("%d", &nivel);

int numerodetentativas;

if(nivel == 1) {

numerodetentativas = 20;

}

else if (nivel == 2) {

numerodetentativas = 15;

}

else {

numerodetentativas = 6;

}

No terminal, testaremos compilando e rodando, e obteremos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 1

Tentativa 1

Qual é o seu chute? 10

Seu chute foi 10

Seu chute foi menor que o número secreto

//...

Tentativa 20

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

Funcionou; escolhemos o nível (1) Fácil e tivemos 20 tentativas. Se optarmos pelo nível (2) Médio, teremos 15 tentativas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 2

Tentativa 1

Qual é o seu chute? 2

Seu chute foi 2

Seu chute foi menor que o número secreto

//...

Tentativa 15

Qual é o seu chute? 2

Seu chute foi 2

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

Também funcionou! Vamos testar o nível (3) Difícil, que deve ter 6 tentativas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 3

Tentativa 1

Qual é o seu chute? 6

Seu chute foi 6

Seu chute foi menor que o número secreto

//...

Tentativa 6

Qual é o seu chute? 6

Seu chute foi 6

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

Funcionou perfeitamente. Adicionamos mais uma funcionalidade ao jogo: perguntar o nível de dificuldade. É possível escrevermos esse mesmo código de outra forma, utilizando switch(), um "irmão" de if:

int acertou = 0;

int nivel;

printf("Qual o nível de dificuldade?\n");

printf("(1) Fácil (2) Médio (3) Difícil\n\n");

printf("Escolha: ");

scanf("%d", &nivel);

int numerodetentativas;

switch(nivel) {

case 1:

numerodetentativas = 20;

case 2:

numerodetentativas = 15;

default:

numerodetentativas = 6;

Funcionará como se estivéssemos usando um if() em (nível), e passaremos a condição na linha abaixo de switch(), com case. Assim,

* se case é de 1, o número de tentativas será 20;
* se case é de 2, o número de tentativas será 15;
* caso contrário, ou else, será default e o número de tentativas será igual a 6.

Notem que isso equivale ao if-elseif-else que estudamos anteriormente, e que switch() é semelhante a if(). Se tivermos uma sequência de if(), poderemos substitui-los por switch(), seguido:

* pela variável que será verificada entre parênteses;
* abertura de chaves (**{}**);
* estabelecimento de quantos case forem necessários;
* substituição de else por default.

Um detalhe ao qual temos que ficar atentos é que ao entrarmos no case 1, a execução continua. Por exemplo, se o o usuário escolher o nível (1) Fácil, na sequência serão executados os níveis (2) Médio e (3) Difícil. Da mesma forma que se o nível escolhido for (2) Médio, (3) Difícil será executado depois, mesmo sem ser selecionado. Essa é a forma como switch() funciona. Para adequá-la ao jogo, acrescentaremos break ao fim de cada case:

int acertou = 0;

int nivel;

printf("Qual o nível de dificuldade?\n");

printf("(1) Fácil (2) Médio (3) Difícil\n\n");

printf("Escolha: ");

scanf("%d", &nivel);

int numerodetentativas;

switch(nivel) {

case 1:

numerodetentativas = 20;

break;

case 2:

numerodetentativas = 15;

break;

default:

numerodetentativas = 6;

break;

Portanto, ao utilizarmos *switch/case*, lembrem-se de inserir break para a "quebra", impedindo-se que as execuções de case e default que aparecem posteriormente. Vamos fazer o teste no terminal, inicialmente selecionando o nível (3) Difícil, que deve ter 6 tentativas:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 3

Tentativa 1

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

//...

Tentativa 6

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

Funcionou! Desta vez, testaremos o nível (2) Médio , com 15 chutes:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Bem-vindo ao nosso jogo de adivinhação \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 2

Tentativa 1

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

//...

Tentativa 15

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

Alura:adivinhacao alura$

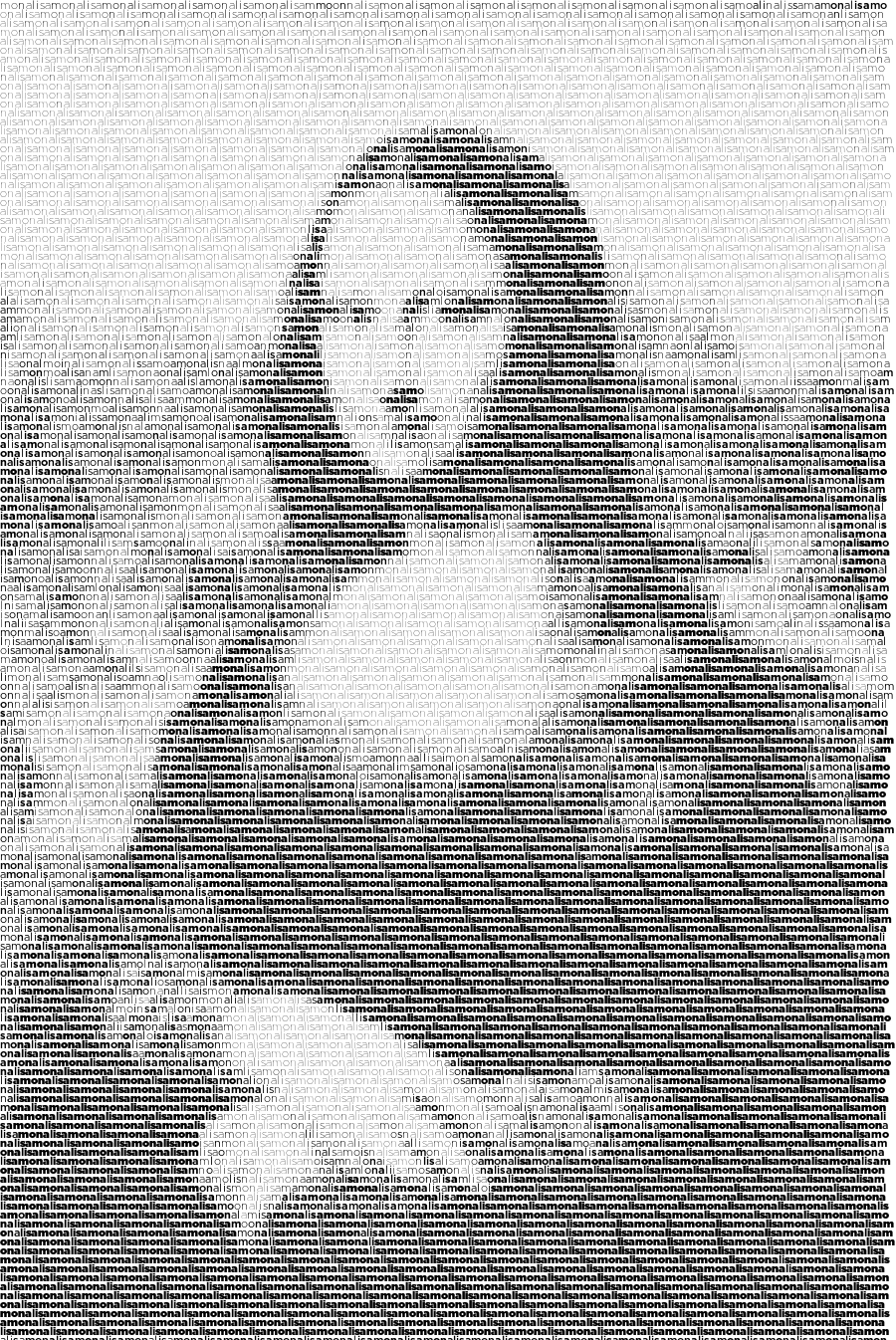
Também funciona. Assim, vimos switch() como alternativa para if(), que funciona com:

* a avaliação da variável especificada entre parênteses seguida pela análise de cada case;
* utilização de default, que não é obrigatório e pode ser usado no lugar de else.

E não se esqueçam de acrescentar break, após o final de cada case.

Precisaremos deixar a aplicação um pouco mais apresentável, nos preocupando com a interface, fundamental para os usuários. As mensagens apresentadas ao ganharmos e perdermos o jogo e o cabeçalho podem ser aprimorados, por exemplo. Porém, neste curso não estudaremos como fazer softwares com janelas, como em jogos do Windows, Linux ou Mac. Aprenderemos a trabalhar somente com o console.

Se analisarmos o código, ele está organizado com a inserção de \n, o que proporciona uma aparência agradável ao jogo. No entanto, vamos tentar acrescentar imagens para melhorá-la ainda mais. Para isso, darei uma dica: é possível encontrar desenhos no Google buscando por "ascii art", desenhos super criativos feitos com letras do teclado, como esta Mona Lisa:



Para o jogo que estamos desenvolvendo, acrescentaremos:

* um castelo ao cabeçalho;

P /\_\ P

/\_\\_|\_|\_/\_\

n\_n | ||. .|| | n\_n Bem vindo ao

|\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!

|" " | |\_| |" " |

|\_\_\_\_\_| ' \_ ' |\_\_\_\_\_|

\\_\_|\_|\_\_/

* um *smile* à mensagem de parabéns;

OOOOOOOOOOO

OOOOOOOOOOOOOOOOOOO

OOOOOO OOOOOOOOO OOOOOO

OOOOOO OOOOO OOOOOO

OOOOOOOO # OOOOO # OOOOOOOO

OOOOOOOOOO OOOOOOO OOOOOOOOOO

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

OOOO OOOOOOOOOOOOOOOOOOOOOOOOO OOOO

OOOO OOOOOOOOOOOOOOOOOOOOOOO OOOO

OOOO OOOOOOOOOOOOOOOOOOOO OOOO

OOOOO OOOOOOOOOOOOOOO OOOO

OOOOOO OOOOOOOOO OOOOOO

OOOOOO OOOOOO

OOOOOOOOOOOO

* um boneco fazendo careta à mensagem caso o jogador perca.

\|/ \_\_\_\_ \|/

@~/ ,. \~@

/\_( \\_\_/ )\_\

\\_\_U\_/

Todos estes desenhos foram encontrados via Google, e a ideia é copiar e colá-los, mas se quiserem, podem se divertir e desenvolver desenhos próprios. Adicionaremos os desenhos ao código e, para imprimi-los, precisaremos de printf() à frente de cada linha dos desenhos. Vamos inserir o castelo primeiro:

int main() {

printf(" P /\_\ P ");

printf(" /\_\\_|\_|\_/\_\ ");

printf(" n\_n | ||. .|| | n\_n Bem vindo ao ");

printf(" |\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!");

printf(" |" " | |\_| |" " | ");

printf(" |\_\_\_\_\_| ' \_ ' |\_\_\_\_\_| ");

printf(" \\_\_|\_|\_\_/ ");

/...

}

Sempre lembrando da indentação para manter a organização do código. Testaremos no terminal para ver como ficou:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

adivinhacao.c:7:25: warning: unknown scape sequence '\ '

printf(" P /\_\ P ");

^~

adivinhacao.c:8:21: warning: unknown scape sequence '\\_'

printf(" /\_\\_|\_|\_/\_\ ");

^~

adivinhacao.c:8:29: warning: unknown scape sequence '\'

printf(" /\_\\_|\_|\_/\_\ ");

^~

adivinhacao.c:13:20: warning: unknown scape sequence '\\_'

printf(" \\_\_|\_|\_\_/ ");

^~

4 warnings generated.

Alura:adivinhacao alura$

Obtivemos quatro avisos de retorno, relacionados a *unknown escape sequence*. Se analisarmos, veremos que indicam o uso da barra (**\**). No código, ela possui função, anunciando que em seguida entrará um caractere especial. Lembrem-se que após "n" (**\n**), ela equivale a "Enter".

Para imprimir somente a barra (**\**), acrescentaremos mais uma (**\\**). No editor de texto, a cor das barras até muda. No código, o acréscimo da barra movimenta os caracteres à direita, mas ao executarmos o programa, eles permanecerão em suas posições.

int main() {

printf(" P /\_\\ P ");

printf(" /\_\\\_|\_|\_/\_\\ ");

printf(" n\_n | ||. .|| | n\_n Bem vindo ao ");

printf(" |\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!");

printf(" |" " | |\_| |" " | ");

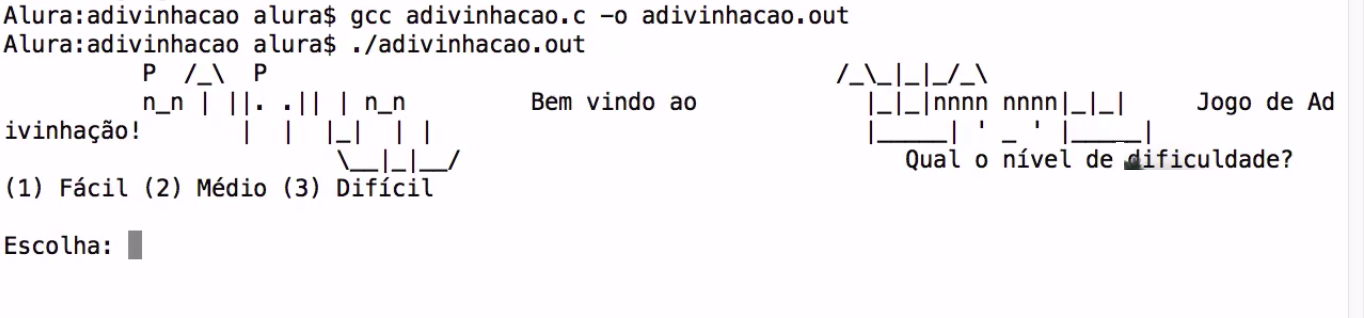
printf(" |\_\_\_\_\_| ' \_ ' |\_\_\_\_\_| ");

printf(" \\\_\_|\_|\_\_/ ");

/...

}

Ao testarmos no terminal, teremos como retorno o desenho desconfigurado, com os espaçamentos fora de lugar:



Isso aconteceu porque não quebrarmos as linhas com \n. Acrescentaremos dois antes e dois no final do desenho, para que o castelo fique isolado.

int main() {

printf("\n\n");

printf(" P /\_\\ P \n");

printf(" /\_\\\_|\_|\_/\_\\ \n");

printf(" n\_n | ||. .|| | n\_n Bem vindo ao \n");

printf(" |\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!\n");

printf(" |" " | |\_| |" " | \n");

printf(" |\_\_\_\_\_| ' \_ ' |\_\_\_\_\_| \n");

printf(" \\\_\_|\_|\_\_/ \n");

printf("\n\n");

//...

}

Se compilarmos e rodarmos novamente no terminal, teremos o castelo na forma que planejamos:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

P /\_\ P

/\_\\_|\_|\_/\_\

n\_n | ||. .|| | n\_n Bem vindo ao

|\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!

|" " | |\_| |" " |

|\_\_\_\_\_| ' \_ ' |\_\_\_\_\_|

\\_\_|\_|\_\_/

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha:

Ficou muito melhor! Notem que os caracteres permaneceram em seus devidos lugares e que onde haviam duas barras (**\\**) no código, no terminal foram impressas somente uma (**\**). Vamos acrescentar as outras figuras ao código: o *smile* na mensagem de parabéns em if(acertou) caso o jogador vença, e a careta em else, caso o jogador perca:

if(acertou) {

printf(" OOOOOOOOOOO \n");

printf(" OOOOOOOOOOOOOOOOOOO \n");

printf(" OOOOOO OOOOOOOOO OOOOOO \n");

printf(" OOOOOO OOOOO OOOOOO \n");

printf(" OOOOOOOO # OOOOO # OOOOOOOO \n");

printf(" OOOOOOOOOO OOOOOOO OOOOOOOOOO \n");

printf("OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO \n");

printf("OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO \n");

printf("OOOO OOOOOOOOOOOOOOOOOOOOOOOOO OOOO \n");

printf(" OOOO OOOOOOOOOOOOOOOOOOOOOOO OOOO \n");

printf(" OOOO OOOOOOOOOOOOOOOOOOOO OOOO \n");

printf(" OOOOO OOOOOOOOOOOOOOO OOOO \n");

printf(" OOOOOO OOOOOOOOO OOOOOO \n");

printf(" OOOOOO OOOOOO \n");

printf(" OOOOOOOOOOOO \n");

printf("\n\n");

printf("Parabéns! Você ganhou!\n");

printf("Você acertou em %d tentativas!\n", tentativas);

printf("Total de pontos: %.1f\n", pontos);

} else {

printf("Você perdeu! Tente de novo!\n");

printf(" \\|/ \_\_\_\_ \\|/ \n");

printf(" @~/ ,. \\~@ \n");

printf(" /\_( \\\_\_/ )\_\\ \n");

printf(" \\\_\_U\_/ \n");

//...

}

Reparem que acrescentamos barras (**\**) na careta. Cuidado para não se esquecerem de nenhuma, para evitar os avisos que recebemos na hora de executar o desenho do castelo. Vamos fazer o teste no terminal, primeiro perdendo:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

P /\_\ P

/\_\\_|\_|\_/\_\

n\_n | ||. .|| | n\_n Bem vindo ao

|\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!

|" " | |\_| |" " |

|\_\_\_\_\_| ' \_ ' |\_\_\_\_\_|

\\_\_|\_|\_\_/

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 3

Tentativa 1

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 2

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 4

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 5

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Tentativa 6

Qual é o seu chute? 1

Seu chute foi 1

Seu chute foi menor que o número secreto

Fim de jogo!

Você perdeu! Tente de novo!

\|/ \_\_\_\_ \|/

@~/ ,. \\~@

/\_( \\_\_/ )\_\

\\_\_U\_/

Alura:adivinhacao alura$

Funcionou, e ficou legal! Vamos tentar ganhar agora, selecionando um nível mais fácil:

Alura:adivinhacao alura$ gcc adivinhacao.c -o adivinhacao.out

Alura:adivinhacao alura$ ./adivinhacao.out

P /\_\ P

/\_\\_|\_|\_/\_\

n\_n | ||. .|| | n\_n Bem vindo ao

|\_|\_|nnnn nnnn|\_|\_| Jogo de Adivinhação!

|" " | |\_| |" " |

|\_\_\_\_\_| ' \_ ' |\_\_\_\_\_|

\\_\_|\_|\_\_/

Qual o nível de dificuldade?

(1) Fácil (2) Médio (3) Difícil

Escolha: 1

Tentativa 1

Qual é o seu chute? 50

Seu chute foi 50

Seu chute foi maior que o número secreto

Tentativa 2

Qual é o seu chute? 25

Seu chute foi 25

Seu chute foi menor que o número secreto

Tentativa 3

Qual é o seu chute? 35

Seu chute foi 35

Seu chute foi maior que o número secreto

Tentativa 4

Qual é o seu chute? 30

Seu chute foi 30

Seu chute foi menor que o número secreto

Tentativa 5

Qual é o seu chute? 32

Seu chute foi 32

Seu chute foi maior que o número secreto

Tentativa 6

Qual é o seu chute? 31

Seu chute foi 31

Seu chute foi menor que o número secreto

Fim de jogo!

OOOOOOOOOOO

OOOOOOOOOOOOOOOOOOO

OOOOOO OOOOOOOOO OOOOOO

OOOOOO OOOOO OOOOOO

OOOOOOOO # OOOOO # OOOOOOOO

OOOOOOOOOO OOOOOOO OOOOOOOOOO

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

OOOO OOOOOOOOOOOOOOOOOOOOOOOOO OOOO

OOOO OOOOOOOOOOOOOOOOOOOOOOO OOOO

OOOO OOOOOOOOOOOOOOOOOOOO OOOO

OOOOO OOOOOOOOOOOOOOO OOOO

OOOOOO OOOOOOOOO OOOOOO

OOOOOO OOOOOO

OOOOOOOOOOOO

Parabéns! Você ganhou!

Você acertou em 6 tentativas!

Total de pontos: 984.5

Alura:adivinhacao alura$

Ganhamos e o *smile* foi impresso, como esperado. Assim, vimos que para imprimir barra (**\**) com printf() é necessário duplicá-la (**\\**) para indicar que só queremos imprimi-la. Utilizamos somente funcionalidades que já havíamos estudado, a diferença é que acrescentamos ascii art para deixar o jogo mais bonito.