# KAPITOLA 2

# Hádání slov („šibenice“)

### Jakou hru budete tvořit

Obsahem této kapitoly je vytvoření hry, ve které se hráč snaží uhodnout slovo pomocí hádání jednotlivých písmen během omezeného počtu pokusů. U nás je taková hra, klasicky hraná s papírem a tužkou, známá jako Šibenice, ovšem v textové variantě samozřejmě žádné kreslení oběšence nebude.

Kromě základních principů řešení daných problémů byste po dokončení této kapitoly měli:

- Znát a umět použít proměnné typu `bool` a `char`
- Vědět, co je pole a jak je vytvořit a také s ním umět pracovat
- Dobře ovládat práci s cyklem `for`
- Umět použít cyklus `while`
- Vědět, co je postfixový inkrement a dekrement

Před návrhem aplikace je třeba ještě zmínit, že v zájmu zachování vyšší přehlednosti a jednoduchosti budete programovat hru ve dvou fázích. V první fázi bude hráč hádat pouze jedno předem zadané slovo, aby nebylo použití vnořených cyklů příliš matoucí. Ve druhé fázi pak aplikaci upravíte – především zajistíte, že hra bude obsahovat více slov k hádání, aby bylo možné hrát ji delší dobu.

Nyní je třeba navrhnout funkční stránku první fáze hry. Stejně jako v předchozím příkladu půjde i zde o konzolovou aplikaci, jejíž specifika již znáte, a proto zde nebudou dále rozebírána.

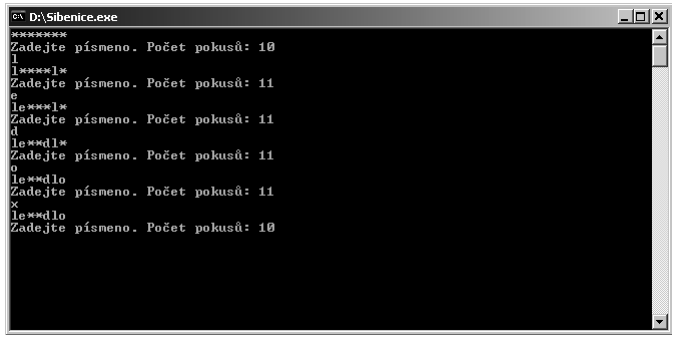
Po startu vybere program slovo (to bude zatím pevně zadané), které má hráč pomocí tipování písmen uhodnout, a zobrazí jej zamaskované – jednoduše nahradí každé písmeno v daném slově určitým znakem, například „*“. Uživatel tak bude zpočátku pouze vědět, kolik písmen hádané slovo obsahuje. A jak to v této hře bývá, hádáním správných písmen dojde k postupnému odkrytí celého slova.

Při každém tahu program zobrazí zprávu, ve které požádá uživatele, aby zadal tipované písmeno, a zároveň vypíše počet zbývajících pokusů. Když tak uživatel učiní a písmeno se ve slově skutečně vyskytuje, systém jej při dalším výpisu hádaného slova zobrazí – tedy nebude již nadále místo něj zobrazovat znak „*“. Za každé správně uhodnuté písmeno také přičte hráči další pokus.

Pokud však uživatel tipuje takové písmeno, které se ve slově nevyskytuje, je mu odečten pokus a v případě, že mu ještě nějaké pokusy zbývají, jej hra vyzve k opětovnému zadání písmene. Pokud má ale uživatel již všechny pokusy vyčerpány, dojde k vypsání zprávy o prohře a ukončení hry.

K opačné situaci, tedy výhře uživatele, dojde samozřejmě v případě, kdy jsou všechna písmena odkryta a počet zbývajících pokusů je vyšší než 0.

Když máte nyní představu o tom, jak bude hra asi fungovat, můžete rovnou založit konzolovou aplikaci pojmenovanou nejlépe „Sibenice“. I v tomto případě ještě budete veškerý kód hry psát do těla metody `Main`.




**Obrázek 2.1** Hádání ukrytého slova

# Proměnné pro počítadla a hádané slovo

Zamyslete se nyní nad návrhem hry, který je popsán výše. Jaké proměnné budete asi potřebovat? Začněte třeba tou hlavní, tedy slovem, které má hráč uhodnout. Slovo jako takové je obyčejný textový řetězec, takže příslušná proměnná musí být typu `string`. Deklarujte tedy tuto proměnnou s výstižným názvem (např. `hadaneSlovo`) a rovnou ji inicializujte na hodnotu, která tak bude představovat hádané slovo. Jak vidíte níže, v příkladu bude uživatel hádat slovo „automobil“.

```
string hadaneSlovo="automobil";
```

Jak již bylo zmíněno, uživatel bude mít na hádání písmen jen omezený počet pokusů – například 10. Z toho důvodu bude samozřejmě program muset vědět, kolik pokusů uživateli ještě zbývá. Protože jde o malé (celé) číslo, zcela postačí proměnná typu `short`, které při deklaraci přiřadíte hodnotu 10.

```
short zbyvajiciPokusy=10;
```

Podobně bude jistě vhodné vědět, kolikrát uživatel uhodl správné písmeno. To můžete využít jako jednu z možností, jak zjistit, zda již uživatel odkryl celé slovo – jednoduše by stačilo porovnat počet písmen ve slově s počtem uhodnutých. Je však třeba myslet na to, že některá písmena se mohou ve slově vyskytnout víckrát než jednou, a proto bude nutné započítat každé z nich, což uvidíte později.

Protože počet uhodnutých písmen bude velmi malé číslo, opět nám postačí proměnná typu `short`:

```
short uhodnutaPismena=0;
```

# Práce se znakem aneb typ char

Než budete moci pokračovat, je třeba si představit další užitečný typ – `char`. Proměnná tohoto typu může obsahovat jeden libovolný znak Unicode (2 bajty), tedy například číslici nebo písmeno z různých světových abeced.

Deklarovat proměnné a přiřazovat jim hodnoty již umíte, u typu `char` je však potřeba uzavřít znakovou konstantu mezi apostrofy, jak vidíte níže:

```
//Ukázka! Tento kód není součástí hry!
char pismeno= 'A';
char cislice= '5';
```

S proměnnými tohoto typu budete v této kapitole hodně pracovat, protože hádání písmen, jejich porovnávání či nahrazování, to vše bude práce s jednotlivými znaky.

Protože uživatel bude muset zadávat hádaný znak, deklarujete pro tento účel proměnnou typu `char`. Hodnota jí bude přiřazena až později vstupem od uživatele.

```
char hadanyZnak;
```

Na tomto místě je vhodné se ještě zmínit o typu `string`, který jste poznali v předchozí kapitole. Když nyní znáte znaky (`char`), pak již můžete o typu `string` uvažovat jako o řetězci složeném z těchto znaků. Brzy uvidíte, jak toho využít.

# Práce s polem

Podle návrhu hry bude uživatel hádat jednotlivá písmena, která budou zpočátku zamaskována znakem „*“. Při správném pokusu bude uhodnuté písmeno zobrazeno na svém místě ve slově, takže hodnota příslušného znaku se změní z „*“ na dané písmeno. Abyste mohli takto pracovat s jednotlivými písmeny ve slově, budete pro každé z nich potřebovat jednu proměnnou typu `char`. To však znamená, že pro různě dlouhá slova bude program potřebovat různý počet těchto proměnných, což je problém.

Jak tedy řešit situaci, kdy potřebujete uchovávat předem neznámý počet proměnných a zároveň k nim mít snadný přístup nezávislý na pojmenování?

Odpověď vám dá datová struktura zvaná pole, která umožňuje uchovávat proměnné (nazývané prvky pole) stejného typu a přistupovat k nim pomocí číselného indexu. Pole je tedy skupina proměnných stejného typu, s níž lze pracovat jako s celkem a jejíž jednotlivé prvky jsou očíslovány (indexovány).

Velikost pole, tj. maximální počet jeho prvků, lze určit proměnnou při jeho vytváření, avšak nelze ji už později změnit.

Pole v jazyce C# začínají indexem 0, takže pokud deklarujete pole o velikosti 2, budou jednotlivé prvky přístupné pod indexy 0 a 1.

Pro lepší pochopení se podívejte na ukázkový kód níže, kde je deklarováno pole celých čísel (typ `int`) o velikosti 3. Do tohoto pole jsou následně vloženy 3 různé hodnoty typu `int`. A jak bylo již zmíněno, první prvek má vždy index 0, takže třetí prvek zde bude mít index 2.

```
//Ukázka! Tento kód není součástí hry!
int [] pole=new int [3];  //vytvoření pole o velikosti 3
pole[0]=10;   //přiřazení hodnoty 10 pod index 0
pole[1]=4;    //přiřazení hodnoty 4 pod index 1
pole[2]=5000; //přiřazení hodnoty 5000 pod index 2
```

U deklarace pole je tedy nutné nejprve uvést typ proměnných, které bude uchovávat. Následují hranaté závorky, kterými se označuje pole a název. Operátor `new` slouží k vytvoření nové instance pole, číselná hodnota v hranaté závorce pak určuje velikost pole. Místo čísla (`int`) můžete samozřejmě použít i proměnnou tohoto typu.

Přistupovat k jednotlivým prvkům pole lze snadno pomocí zápisu názvu pole následovaného indexem prvku v hranatých závorkách:

```
//Ukázka! Tento kód není součástí hry!
string [] jmena=new string [5];  //vytvoření pole o velikosti 5
jmena[4]= "Petr";   //přiřazení hodnoty "Petr" prvku s indexem 4
string A=Jmena[4]; //proměnná A bude nastavena na "Petr"
```

Nyní, když máte základní představu o tom, jak pracovat s polem, můžete se vrátit zpět ke kódu hry.

Jak už bylo řečeno, k práci s jednotlivými znaky v hádaném slově budete potřebovat stejný počet proměnných typu `char`. Toho snadno dosáhnete vytvořením pole tohoto typu, jehož velikost bude odvozena přímo z délky hádaného slova.

Proměnná typu `string` (`hadaneSlovo`) totiž disponuje řadou členů, které zpřístupníte použitím operátoru tečky (.) a mezi nimiž najdete i vlastnost `Lenght`. Ta vrací délku daného řetězce jako hodnotu typu `int`, a proto je možné její hodnotu použít při inicializaci pole:

```
char[] znakySlova=new char[hadaneSlovo.Length];
```

Pole `znakySlova` tak bude mít takovou velikost (resp. takovou délku), kolik znaků obsahuje hodnota proměnné `hadaneSlovo` v době vykonání příkazu. Připomeňte si, že této proměnné jste v úvodu přiřadili řetězec „automobil“, pole bude mít tedy 9 prvků.

# Naplnění pole pomocí cyklu for

Nyní, když jste se postarali o tvorbu pole se správnou velikostí, je třeba zajistit jeho naplnění proměnnými – znaky, kterým zpočátku nastavíte hodnotu „*“. Pole bude tedy obsahovat tolik znaků „*“, kolik má hádané slovo písmen. Tak vlastně vytvoříte (z prvků tohoto pole) zamaskovanou verzi daného slova.

Výše jste viděli, že nastavit hodnoty pro jednotlivé prvky v poli je snadné. V tomto případě byste museli napsat 9 řádků kódu, což ještě není tak špatné, ale co kdyby šlo o pole s 500 prvky nebo byste ani nevěděli, kolik prvků bude pole mít?

K provádění takových úloh, kde je nutné opakovaně vykonávat určitou činnost, lze využít cyklus `for`. Ten umožňuje ve smyčce vykonávat příkaz či blok příkazů do té doby, dokud platí zadaná podmínka, a je používán především tehdy, když je potřeba vykonat konkrétní počet opakování.

Abyste tedy každému prvku v poli nastavili hodnotu „*“, zapíšete příkaz `for` v následující podobě:

```
for (int I = 0; I < hadaneSlovo.Length; I++)
{
  znakySlova[I] = '*';
}
```

Nejprve si prohlédněte samotný zápis příkazu. V závorce za klíčovým slovem `for` najdete jako první inicializaci řídící proměnné (klasicky s názvem `I`). To se děje pouze při prvním zpracování příkazu. Následuje podmínka (`I<hadaneSlovo.Lenght`) – když je splněna, jsou příkazy v těle cyklu provedeny. Poté dojde k aktualizaci řídící proměnné (`I++`, viz níže) a cyklus pokračuje znova, ovšem již od vyhodnocení podmínky.

U příkazu `I++` je třeba se zastavit, protože jste se s ním dosud nesetkali. Jde o zápis, který je ekvivalentní výrazu `I=I+1` a nazývá se *postfixová inkrementace*. Jednoduše tak zvýší hodnou dané proměnné o 1.

Stejně tak lze zapsat postfixovou dekrementaci `I--`, což se rovná zápisu `I=I-1`. Používání těchto zápisů patří k základním návykům programátora a měli byste je používat v každém jazyce, který to umožňuje.

### Poznámka

Inkrementace i dekrementace mohou být i prefixové (++I). Rozdíl zde spočívá v prioritě provedení operace.

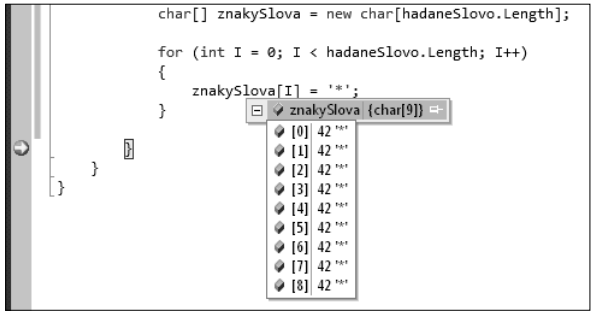
Teď se podívejte, jak cyklus funguje ve vašem případě. Nejprve je inicializována proměnná `I` s počáteční hodnotou nula. Poté je vyhodnocena podmínka – hodnota proměnné `I` musí být menší než velikost pole `hadaneSlovo` (ta je 9). Protože `I` je zatím 0, podmínka je splněna a dojde ke spuštění příkazu v těle cyklu. Poté je aktualizována proměnná `I` příkazem `I++` a smyčka pokračuje znova vyhodnocením podmínky – proměnná `I` je nyní 1, takže podmínka je opět vyhodnocena jako pravdivá, opět dojde k provedení příkazu a opět je k proměnné `I` přičtena 1. Cyklus skončí v momentě, kdy bude mít proměnná `I` hodnotu 9. Pak již nebude podmínka dále platit, cyklus bude ukončen a program bude pokračovat prvním následujícím příkazem po `for`.

### Poznámka

Především v souvislosti s cykly se často setkáte s pojmem *iterace*, který v podstatě označuje opakování příkazů – jednotlivý průchod cyklem. Dle příkladu výše lze říct, že při každé iteraci roste hodnota proměnné `I`.

Pokud se podíváte na příkaz `znakySlova[I] = '*'` v těle cyklu, vidíte, že index prvku v poli určuje hodnota proměnné `I`. Při prvním průchodu cyklem tak bude mít `I` hodnotu 0, při druhém 1, při třetím 2 atd., takže během zpracování cyklu bude postupně každému prvku v poli přiřazen znak „*“.

Pro prohlédnutí obsahu pole za běhu programu můžete použít režim ladění tak, jak jste se naučili v předchozí kapitole. Jednoduše umístěte zarážku za příkaz `for`, spusťte aplikaci, a až dojde k pozastavení programu na zadaném místě, podržte kurzor myši nad polem `znakySlova`.




**Obrázek 2.2** Zobrazení obsahu pole (proměnné znakySlova)

K lepšímu pochopení `for` cyklů můžete použít následující kód, který vypíše 50 číslic od 0 do 49. Podívejte se, že ve skutečnosti jednoduše vypisuje hodnotu proměnné `I` tak dlouho, dokud je nižší než 50. Pak již podmínka nebude platit a cyklus skončí.

```
//Ukázka! Tento kód není součástí hry!
for(int I=0;I<50;I++)
{
  Console.WriteLine(I);
}
```

# Pravda či nepravda aneb proměnná typu bool

Když si vzpomenete na vyhodnocování výrazů, vybaví se vám pravděpodobně tvrzení, že výraz může být vyhodnocen jako pravdivý či nikoli. Nyní je čas trochu toto tvrzení upřesnit.

Víte, že existují různé typy reprezentované hodnotou, kterou mohou obsahovat – tak třeba typ `int` pro celá čísla či `char` pro znaky. A protože pravda či nepravda jsou logické hodnoty, je pro ně určen datový typ boolean – `bool`.

Ten může nabývat hodnot pouze `true` (pravda) či `false` (nepravda). Podívejte se, jak lze pracovat s proměnnou tohoto typu:

```
//Ukázka! Tento kód není součástí hry!
bool odpoved=false; //původně je proměnné přiřazena hodnota false
odpoved=true;       //zde přiřadíte hodnotu true
```

Je třeba si uvědomit, že výsledkem vyhodnocení logického výrazu je vždy jedna z těchto hodnot, kterou můžete snadno zjistit. V příkladu níže vidíte, že proměnné `vysledek` bude přiřazena hodnota podle výsledku vyhodnocení výrazu. Pokud bude číslo 10 větší než náhodně vygenerované číslo v proměnné `cislo`, výraz bude pravdivý. Proměnná `vysledek` tak bude mít hodnotu `true`.

Příkaz `if`, jak víte, rozhoduje na základě vyhodnocení logického výrazu, jinými slovy očekává hodnotu buď `true`, anebo `false` – tedy typ `bool`. Proměnnou tohoto typu tak samozřejmě můžete použít jako podmínku:

```
//Ukázka! Tento kód není součástí hry!
Random nahodnaCisla=new Random();
int cislo=NahodnaCisla.Next(1,20);
bool vysledek = 10 > cislo;
if (vysledek)
{
  ...
}
```

Zpět ale ke kódu vaší hry. V proměnné typu `bool` s názvem `hraProbiha`, kterou deklarujete za tělem `for` cyklu, budete udržovat informaci o tom, zda hra ještě probíhá (slovo tedy zatím není celé odkryté), či nikoli. To bude později důležité pro určení chování programu.

Po startu aplikace bude hra samozřejmě probíhat, proto tuto proměnnou inicializujte na hodnotu `true`:

```
bool hraProbiha = true;
```

Dosavadní kód hry v metodě `Main` by měl vypadat takto:

```
string hadaneSlovo="automobil";
short zbyvajiciPokusy=10;
short uhodnutaPismena=0;
char hadanyZnak;
char[] znakySlova=new char[hadaneSlovo.Length];
for (int I = 0; I < hadaneSlovo.Length; I++)
{
  znakySlova[I] = '*';
}
bool hraProbiha = true;
```

# Smyčka hry pomocí cyklu while

Po úvodním načtení proměnných se program musí dostat do stavu, ve kterém bude uživatel moci hádat a zadávat písmena tak, aby odkryl zamaskované slovo. Po každém takovém pokusu systém znovu toto slovo vypíše, dosadí případně uhodnutá písmena a opět nechá hráče hádat. Program by se tedy v této části měl pohybovat ve smyčce do té doby, než hráč vyčerpá pokusy či celé slovo odkryje.

K opakovanému vykonávání příkazů již umíte použít cyklus `for`, ale nyní se naučíte i cyklus `while`, který je pro tento účel vhodnější.

Jak vidíte níže, jeho zápis je jednodušší než v případě `for` cyklu – spuštění příkazů v těle cyklu je závislé pouze na pravdivosti zadaného výrazu. To znamená, že příkazy budou vykonávány tak dlouho, dokud bude výraz vyhodnocován jako `true`.

Protože v těle tohoto cyklu bude obsažen kód hry určený k opakování (smyčka popsaná výše), bude tato smyčka probíhat tak dlouho, dokud bude mít proměnná `hraProbiha` hodnotu `true`:

```
while (hraProbiha)
{
}
```

V momentě, kdy bude mít při vyhodnocování podmínky proměnná `hraProbiha` hodnotu `false` (to zajistíte později v případě, kdy již uživatel nemá žádné další pokusy nebo odkryl celé slovo), smyčka skončí a program bude pokračovat prvním následujícím příkazem.

Když však program nyní spustíte, bude smyčka probíhat do nekonečna, protože hodnota proměnné `hraProbiha` bude stále `true`.

**Poznámka**

V určitých aplikacích může být potřeba vytvořit nekonečnou smyčku, k čemuž se běžně používá jednoduchý zápis while (true).

# Vypsání maskovaného slova a zprávy pro uživatele

Jak bylo již nastíněno, v samotné hře se opakuje sekvence těchto kroků: zobrazení maskovaného slova, vypsání zprávy s počtem pokusů a zadání písmena uživatelem.

Začněte prostým zobrazením maskovaného slova, tedy vlastně vypsáním obsahu pole `znakySlova`, které zatím obsahuje jen samé znaky „*“.

Vypisovat jednotlivé prvky pole vlastním způsobem by sice bylo možné, ale rychlejší způsob je využít toho, že metoda `Console.WriteLine` dokáže vypsat prvky pole znaků jako řetězec. To znamená, že projde polem prvek po prvku a každý znak vypíše.

Zápis je snadný, do závorek za volání metody jednoduše uvedete název pole, jehož prvky chcete vypsat – tedy `znakySlova`. Nezapomeňte také, že píšete kód uvnitř `while` cyklu.

```
Console.WriteLine(znakySlova);
```

Poté už stačí jen vypsat zprávu pro uživatele s počtem zbývajících pokusů. Nejde o nic jiného než o zadání textového řetězce a zobrazení požadované proměnné:

```
Console.WriteLine("Zadejte písmeno. Počet pokusů:
  "+zbyvajiciPokusy);
```

# Zadání písmene uživatelem

Po zobrazení potřebných informací je nyní řada na uživateli. Ten má tedy zadat jeden znak (ideálně samozřejmě písmeno), o kterém si myslí, že se ve slově může vyskytovat.

Přestože existuje lepší možnost, prozatím získání vstupu od uživatele provedete opět pomocí již známé metody `Console.ReadLine`. Budete však muset vyřešit jeden problém: tato metoda totiž umožňuje uživateli zadat textový řetězec, ovšem vy potřebujete pouze jeden znak.

Nejprve založíte proměnnou typu `string` s názvem `zadanyText`, které přiřadíte hodnotu vrácenou metodou `Console.ReadLine`. Zatím to znamená, že tato proměnná bude obsahovat jakkoli dlouhý text, který uživatel zadá.

```
string zadanyText = Console.ReadLine();
```

Jak získat z daného textového řetězce pouze jeden (první) znak? Vzpomeňte si na poznámku, že o typu `string` lze uvažovat jako o řetězci znaků typu `char`. Ve skutečnosti je navíc dokonce možné přistupovat ke znakům v typu `string`, jako by to byly prvky pole – vzhledem k názvu proměnné tedy

můžete přistupovat k hodnotě prvního znaku zápisem `zadanyText[0]`, kde index v hranaté závorce udává pozici znaku v řetězci.

Když nyní víte, jak první znak získat, je třeba jeho hodnotu někam uložit, abyste s ní mohli dále pracovat. Získaná hodnota bude typu `char` a právě k tomuto účelu jste již v úvodu deklarovali proměnnou `hadanyZnak`. Nyní je tedy na čase přiřadit jí hodnotu:

```
hadanyZnak=zadanyText[0];
```

Proměnná `hadanyZnak` tak bude obsahovat hodnotu prvního znaku z textu zadaného uživatelem, což můžete opět snadno ověřit v režimu ladění. Je zde ovšem jedno riziko, které bude třeba ošetřit.

Ať uživatel zadá textový řetězec s délkou jednoho či více znaků, program vždy správně použije pouze první z nich. Problém ale nastane v případě, kdy uživatel nezadá žádný vstup a pouze stiskne Enter. Pokud to zkusíte, dojde v programu k chybě (výjimce).

Abyste totiž získali první znak ze zadaného vstupu, používáte zápis `zadanyText[0]`. Když však uživatel žádný text nezadá, proměnná `zadanyText` nebude obsahovat žádné znaky, a pokud o ní uvažujete jako o poli znaků, tak toto pole bude mít tedy nulovou velikost. Když se tak pokusíte získat hodnotu jeho prvního prvku, dojde k chybě – zadaný index je vyšší než velikost pole.

K ošetření tohoto problému postačí, když získání prvního znaku provedete jen tehdy, když zadaný text obsahuje více než 0 znaků. Jinak řečeno, pokud je délka vstupu 0, uživatel jej bude muset zopakovat. Upravte tedy kód takto:

```
string zadanyText = Console.ReadLine();
if (zadanyText.Length == 0)
{
  continue;
}
hadanyZnak = zadanyText[0];
```

Jestliže je tedy délka vstupu v proměnné `zadanyText` rovna 0, je spuštěn příkaz `continue`. Ten způsobí, že zbývající příkazy v cyklu budou přeskočeny a cyklus pokračuje dalším opakováním.

Zadáním písmene využil uživatel jeden pokus, musíte tedy snížit hodnotu proměnné `zbyvajiciPokusy` o 1:

```
zbyvajiciPokusy--;
```

Tělo příkazu `while` by mělo tedy vypadat takto:

```
while (hraProbiha)
{
  Console.WriteLine(znakySlova);
  Console.WriteLine("Zadejte písmeno. Počet pokusů:"
    + zbyvajiciPokusy);
  string zadanyText = Console.ReadLine();
  if (zadanyText.Length == 0)
  {
    continue;
  }
  hadanyZnak = zadanyText[0];
  zbyvajiciPokusy--;
}
```

# Porovnávání písmen a odkrývání slova

Nyní se pustíte do asi nejzajímavější části celého kódu. Zde budete zjišťovat, zda se hádané písmeno v původním slově vyskytuje, a pokud ano, nahradíte jím znak „*“ na příslušné pozici v poli `znakySlova`. Připomeňte si, že obsah tohoto pole představuje vlastně zamaskované slovo a zatím obsahuje jen tolik znaků „*“, kolik je v daném slově písmen.

Princip není složitý – program vždy zkontroluje každý dosud zamaskovaný znak, zda na jeho pozici není v hádaném slově stejné písmeno, jaké zadal uživatel.

Abyste mohli porovnávat jednotlivá písmena, bude nutné zpracovávat obsah pole `znakySlova` prvek po prvku. K tomu využijete `for` cyklus, jenž bude probíhat tolikrát, kolik má dané pole prvků (neboli slovo písmen). To zjistíte stejně jako u typu `string` pomocí vlastnosti `Lenght`. Tento kód je samozřejmě součástí herní smyčky, budete jej tedy zapisovat do těla příkazu `while`.

```
for (int I = 0; I < znakySlova.Length; I++)
{
}
```

V těle cyklu budete nejprve pro každý prvek pole `znakySlova` zjišťovat, zda je jeho hodnota rovna „*“, a jde tedy o dosud neodhalené písmeno.

K tomu použijete příkaz `if`. Aby měl cyklus vůbec smysl a skutečně procházel polem prvek po prvku, musíte jako index opět použít řídicí proměnnou `I`, protože při každém průchodu cyklem je její hodnota navýšena o 1:

```
if(znakySlova[I]== '*')
{
}
```

Jestliže je podmínka platná, písmeno na pozici `I` je zatím neuhodnuté. V takovém případě musí program zjistit, zda se uživatelem zadané písmeno nachází na této pozici i v původním slově.

Podívejte se na zamýšlený proces blíže. Představte si, že cyklus `for` probíhá poprvé (`I=0`), a podmínka tak kontroluje hodnotu prvku pole s indexem 0. Jestliže bude splněna, není první písmeno zatím odkryté. V takovém případě musí program zjistit, zda je uživatel právě neuhodl, a to bude obsahem těla příkazu `if`.

Uvědomte si, že první prvek pole představuje první písmeno. Program tak musí zjistit, jaké je tedy první písmeno v hádaném slově (proměnná `hadaneSlovo`). Pokud by šlo o druhý prvek (`I=1`), program by musel zjistit druhé písmeno z hádaného slova. Jinými slovy, pozice prvku v poli představuje pozici daného písmene ve slově. Je tedy zřejmé, že pokud použijete `I` pro určení pozice prvku v poli, stejně tak můžete získat odpovídající písmeno ve slově – víte totiž, že k jednotlivým znakům v textovém řetězci lze přistupovat stejně jako k prvkům pole. Získané písmeno (resp. znak) porovnáte se znakem zadaným uživatelem, který jste uložili do proměnné `hadanyZnak`:

```
if (hadaneSlovo[I] == hadanyZnak)
{
}
```

Pokud je podmínka splněna, a tedy písmeno na dané pozici ve slově je stejné jako to, které zadal uživatel, jde o úspěšný pokus. Program tak bude muset uložit hodnotu uhodnutého písmene do prvku pole `znakySlova` pod příslušným indexem.

Nezapomeňte, že celou dobu víte, jaký index má daný prvek – určili jste jej přece podmínkou, že dosud obsahuje hodnotu „*“! Proto je zřejmé, že právě na tuto pozici v poli uložíte hodnotu uhodnutého znaku:

```
znakySlova[I]=hadanyZnak;
```

Podle původního návrhu dostane uživatel za správně tipované písmeno jeden pokus zpět. Pokud to neuděláte, ubude pokus i za uhodnuté písmeno. Pozornější čtenáři si možná všimnou, že pokud se bude ve slově vyskytovat uhodnuté písmeno více než jednou, dostane uživatel zpátky jeden pokus za každý výskyt.

Stejně tak je třeba aktualizovat proměnnou, která udržuje počet dosud uhodnutých písmen.

```
uhodnutaPismena++;
zbyvajiciPokusy++;
```

Protože tato část byla možná o něco obtížnější, zkontrolujte si nyní obsah `for` cyklu:

```
for (int I = 0; I < znakySlova.Length; I++)
{
  if (znakySlova[I] == '*')
  {
    if (hadaneSlovo[I] == hadanyZnak)
    {
      znakySlova[I] = hadanyZnak;
      uhodnutaPismena++;
      zbyvajiciPokusy++;
    }
  }
}
```

Po spuštění nyní můžete zadávat správná písmena a sledovat, jak se postupně doplňují do slova. Přestože pokusy ubývají, zatím nemáte nijak ošetřený konec hry, což bude účel další části.

# Přerušení herní smyčky aneb konec hry

Víte, že posloupnost herních kroků probíhá v cyklu `while` tak dlouho, dokud má proměnná `hraProbiha` hodnotu `true`. Když tedy uživateli nezbývají žádné pokusy nebo je celé slovo uhodnuté, změníte hodnotu této proměnné na `false`.

Nejprve ošetříte situaci, kdy uživatel vyčerpal všechny pokusy. Nemělo by vám to již dělat žádný problém, je to velmi prosté: je-li hodnota proměnné `zbyvajiciPokusy` rovna 0, nastavíte proměnné `hraProbiha` hodnotu `false`. Příkaz zapište až za tělo `for` cyklu (ale samozřejmě v těle cyklu `while`):

```
if(zbyvajiciPokusy==0)
{
  hraProbiha=false;
}
```

A nyní tedy ukončení v případě, kdy je slovo celé odhalené. To poznáte snadno, protože jste zajistili, že při každém uhodnutém písmeně se zvyšuje hodnota proměnné `uhodnutaPismena`. Když bude proto hodnota této proměnné rovna počtu písmen ve slově, znamená to, že jsou všechna písmena odkryta:

```
if(uhodnutaPismena==hadaneSlovo.Length)
hraProbiha=false;
```

Podívejte se nyní na obě podmínky. Ať je splněna první **nebo** druhá z nich, v obou případech se hodnota proměnné `hraProbiha` nastaví na `false`. Proč je tedy nesloučit do jedné pomocí operátoru `||`(nebo)?

```
if(zbyvajiciPokusy==0 || uhodnutaPismena==hadaneSlovo.Length)
hraProbiha=false;
```

Podmínka v příkazu `while` tak již nebude nadále platná, a k dalšímu opakování příkazů v těle cyklu tak nedojde.

**Poznámka**

Podmínka je vyhodnocována vždy až poté, co jsou všechny příkazy v těle cyklu vykonány. Pokud je třeba přerušit cyklus okamžitě, je nutné použít příkaz `break`.

# Určení výsledku hry

Po skončení herní smyčky dojde k vyhodnocení hry, kde teprve odlišíte, zda uživatel vyhrál, či nikoli. Jednoduše opět pomocí podmínky v příkazu `if` ověříte počet zbývajících pokusů – pokud je roven 0, uživatel prohrál. Pokud podmínka nebude platit, provede se příkaz za klauzulí `else`. Protože vyhodnocení hry následuje až po ukončení herní smyčky, zapisujete jej samozřejmě až za tělo cyklu `while`:

```
if(zbyvajiciPokusy==0)
{
  Console.WriteLine("Prohrál jsi!");
}
else
{
  Console.WriteLine("Uhodl jsi!");
}
```

Aby aplikace neskončila a uživatel měl možnost přečíst si výsledek, použijte opět metodu `Console.ReadKey`:

```
Console.ReadKey();
```

Tím jste dokončili první fázi hry, a pokud ji nyní spustíte, měla by fungovat přesně tak, jak bylo navrženo v úvodu.

# První vylepšení – hádání více slov

Jako první vylepšení vyřešíte největší nedostatek dosavadní hry – pracuje pouze s jedním slovem, po jehož uhodnutí hra skončí. Bylo by tedy dobré zařídit, aby hra obsahovala více slov a hráč mohl po uhodnutí jednoho pokračovat dál.

Víte, že hádané slovo obsahuje proměnná s názvem `hadaneSlovo`. Stačilo by tedy, aby po každém uhodnutí slova došlo k tomu, že se celý kód hry provede znova jen s jinou hodnotou této proměnné. Jinými slovy, když uhodne uživatel první slovo, program změní hodnotu proměnné `hadaneSlovo` a spustí celý kód od začátku. Pokud vás napadá, že jde vlastně o jakousi smyčku, tak máte pravdu.

Nejprve bude nutné zařídit, aby existovala zásoba slov určených k hádání. Jde tedy o vytvoření určité skupiny proměnných, se kterými budete postupně pracovat, a to je opět příležitost pro použití pole. Vytvořte tedy pole proměnných typu `string` o velikosti 6 (bude tak moci obsahovat 6 slov) s názvem `slova`, a protože tento kód bude ovlivňovat celou doposud napsanou hru, zapište jej na úplný začátek metody `Main`:

```
string [] slova=new string[6];
```

Nyní je nutné naplnit pole proměnnými, resp. slovy, která bude možné ve hře hádat. Každému prvku v poli tak přiřadíte jednu hodnotu typu `string`:

```
slova[0] = "letadlo";
slova[1] = "automobil";
slova[2] = "fotografie";
slova[3] = "lokomotiva";
slova[4] = "televize";
slova[5] = "velryba";
```

Teď už zbývá jen zařídit, aby hra po skončení a vyhodnocení výsledku neskončila, ale pokračovala znova s jiným (dalším) slovem. Celá hra se tak bude opakovat tolikrát, kolik je v poli slov.

K opakování kódu hry použijete opět příkaz `for`. Počet opakování bude určen velikostí pole `slova`. Kód samotné hry tak uzavřete do těla `for` cyklu, takže tělo metody `Main` nyní vypadá takto:

```
static void Main(string[] args)
{
  string[] slova = new string[5];
  slova[0] = "letadlo";
  slova[1] = "automobil";
  slova[2] = "fotografie";
  slova[3] = "lokomotiva";
  slova[4] = "televize";
  for (int X = 0; X < slova.Length; X++)
  {
    ...zde máte dosavadní kód hry
  }
}
```

Určitě jste si všimli, že ve `for` cyklu je nyní použita řídící proměnná s názvem `X`. Proč není opět použit klasický název `I`, je prosté. Víte, že v těle cyklu můžete pracovat s jeho řídící proměnnou jako by to byla běžně deklarovaná proměnná, ale mimo tělo cyklu tato proměnná neexistuje. V tomto cyklu je ale vnořen celý kód hry, ve kterém jste již proměnné s názvem `I` použili. Došlo by tedy k tomu, že na některých místech by existovaly dvě proměnné se stejným názvem, což je samozřejmě nepřípustné.

Kdybyste teď hru spustili, sice by se šestkrát opakovala, ale pokaždé se stále stejným hádaným slovem. Nemáte zatím totiž zajištěno, aby se hodnota proměnné `hadaneSlovo` měnila s každým opakováním cyklu.

Najděte tedy řádek obsahující `string hadaneSlovo="automobil"` a upravte jej do následující podoby:

```
//Úprava existujícího kódu
string hadaneSlovo=slova[X];
```

Hodnota proměnné `hadaneSlovo` tak bude při každém opakování hry nastavena na hodnotu prvku pole `slova` podle indexu `X`. Připomeňte si, že `X` je řídící proměnná `for` cyklu, takže při každém opakování bude její hodnota vyšší, a bude proto použit prvek s vyšším indexem.

Při startu hry proto bude použito první slovo z pole, resp. prvek s indexem 0 (`X=0`). Po uhodnutí slova (a zatím i po vyčerpání pokusů) již nebudou v těle cyklu žádné další příkazy k provedení a cyklus tak začne znova. Při druhém opakování (`X=1`) tak bude použit druhý prvek pole a tak dále.

Zbývá ještě upravit jednu maličkost. Když totiž uživatel vyčerpá pokusy, sice je vypsána zpráva o prohře, ale protože je i tento kód nyní umístěn ve `for` cyklu, hra následně pokračuje normálně dalším slovem. Je třeba jej tedy upravit tak, aby v případě, že hráč již nemá žádné pokusy, došlo k přerušení cyklu `for`. K tomu slouží příkaz `break`.

Upravte tedy kód do této podoby:

```
//Úprava existujícího kódu
if(zbyvajiciPokusy==0)
{
  Console.WriteLine("Prohrál jsi!");
  Console.ReadKey();
  break;
}
else
{
  Console.WriteLine("Uhodl jsi!");
}
```

Po splnění podmínky bude zobrazena zpráva a po stisku klávesy (zajišťuje `Console.ReadKey`) dojde k ukončení celého `for` cyklu. V aplikaci tak již nebudou žádné další příkazy ke spuštění, a dojde tak k ukončení.

# Druhé vylepšení – skóre za celou hru

Prozatím se uživateli počítají pouze zbývající pokusy, přičemž u každého nového slova jich má opět 10. Hru by však určitě velmi zpestřilo, kdyby existovalo celkové skóre za celou dobu hraní. Stačilo by pouze za každé správně uhodnuté písmeno přičíst bod a naopak za špatné jeden bod odečíst.

Na této úpravě je zajímavé to, že všechny proměnné (počítadla pokusů atd.) jsou při každém opakování hry nastaveny znovu, protože jejich deklarace i přiřazení hodnot jsou v těle `for` cyklu. Abyste tedy mohli pracovat s proměnnou, jejíž hodnota zůstává, je třeba deklarovat ji ještě před `for` cyklem. V místě deklarace pole `slova` nyní deklarujte proměnnou `celkoveSkore` a přiřaďte ji hodnotu 0:

```
int celkoveSkore=0;
```

Přičíst bod za uhodnuté písmeno bude snadné, ale jak poznat, kdy se uživatelem zadané písmeno ve slově nevyskytlo, a on tedy neuhodl?

Jedno z možných řešení je například zjistit, zda se po zadání písmene a následné kontrole slova změnil počet uhodnutých písmen (tzn. pokud uživatel neuhodl, nezmění se hodnota proměnné `uhodnutaPismena`), což si právě sami vyzkoušíte.

Nejprve najděte část kódu, ve které pomocí `for` cyklu kontrolujete znaky slova a přičítáte úspěšné pokusy za uhodnutá písmena. Abyste mohli zjistit, zda se hodnota proměnné `uhodnutaPismena` změnila, bude třeba uložit její původní hodnotu pro pozdější porovnání. Před příkazem `for` proto deklarujte proměnnou typu `int` s názvem například `puvodniPocet` a přiřaďte jí takovou hodnotu, jakou má proměnná `uhodnutaPismena`.

Zařiďte také, že při uhodnutí písmene přičtete bod k celkovému skóre, pro které jste si již založili proměnnou s názvem `celkoveSkore`.

Doplněný kód bude tedy vypadat takto:

```
//Úprava existujícího kódu
int puvodniPocet=uhodnutaPismena;
for (int I = 0; I < znakySlova.Length; I++)
{
  if (znakySlova[I] == '*')
  {
    if (hadaneSlovo[I] == hadanyZnak)
    {
      znakySlova[I] = hadanyZnak;
      uhodnutaPismena++;
      zbyvajiciPokusy++;
      celkoveSkore++; //přidaný příkaz
    }
  }
}
```

Proměnná `puvodniPocet` je tedy inicializována na stejnou hodnotu jako proměnná `uhodnutaPismena`. Pokud však uživatel písmeno uhodl, během cyklu se hodnota proměnné `uhodnutaPismena` zvýší.

Jestliže bude hodnota obou uvedených proměnných stejná i poté, co proběhla kontrola slova (nezměnil se tak počet uhodnutých písmen), uživatel písmeno neuhodl, a je proto potřeba odečíst mu bod.

Za `for` cyklus tak umístěte podmínku k ověření hodnot obou proměnných a příkaz, který v případě splnění podmínky sníží hodnotu celkového skóre:

```
if(puvodniPocet==uhodnutaPismena)
{
  celkoveSkore--;
}
```

A teď už stačí jen zobrazit skóre v textu, který vypisujete v případě výhry. Upravte tedy kód za klauzulí `else` v příkazu `if (zbyvajiciPokusy == 0)`.

```
//Úprava existujícího kódu
if (zbyvajiciPokusy == 0)
{
  Console.WriteLine("Prohrál jsi!");
  break;
}
else
{
  Console.WriteLine("Uhodl jsi! Celkové skóre je:
    "+celkoveSkore);
}
```

Touto poslední úpravou jste úspěšně dokončili druhou kapitolu.

# Shrnutí

| Probraná látka | Příklad |
|---|---|
| Proměnná typu `char` může obsahovat jeden libovolný znak Unicode. Při zadávání znakové konstanty je třeba použít apostrofy. | `char Znak='A';`<br>`char B;`<br>`B='5';` |
| Proměnná typu `bool` může nabývat hodnoty pravda (`true`) nebo nepravda (`false`). Tento typ je výsledkem vyhodnocení každého logického výrazu, a proto jej můžete používat stejným způsobem. | `bool Vysledek=true;`<br>`if (Vysledek)`<br>`{`<br>`  bool PravdaNeboNe=1<2;`<br>`}` |
| Pole je datová struktura, která umožňuje uchovávat skupinu proměnných stejného datového typu a pracovat s ní jako s celkem. Prvky pole jsou interně očíslovány, takže k jednotlivým proměnným lze přistupovat pomocí číselného indexu zapsaného v hranatých závorkách za názvem pole. Při vytváření pole je možné jeho velikost určit dynamicky, ovšem později už ji není možné změnit.<br>Index prvního prvku pole je vždy 0. | `//vytvoření pole`<br>`int[] Cisla=new int[2];`<br>`//přiřazení hodnoty prvkům`<br>`Cisla[0]=4;`<br>`Cisla[1]=99;`<br>`//proměnné X přiřazujete`<br>`//hodnotu prvku s indexem 1`<br>`int X=Cisla[1]; //X bude 99` |
| Příkaz `for` slouží k opakování příkazů v cyklu po dobu platnosti zadané podmínky. Jeho součástí je řídící proměnná, jejíž hodnota se mění s každým opakováním cyklu. Používá se především tehdy, když je nutný určitý konkrétní počet opakování. | `//výpis čísel 0-99`<br>`for(int I=0;I<100;I++)`<br>`{`<br>`  Console.WriteLine(I);`<br>`}` |
| Příkaz `while` umožňuje opakovaně vykonávat příkazy v cyklu po dobu platnosti zadané podmínky. Oproti `for` cyklu se používá tehdy, když není nutné pracovat s pořadím jednotlivých opakování, jejichž počet není předem známý. | `//nekonečná smyčka`<br>`while(true)`<br>`{`<br>`  Console.WriteLine("Ahoj!")`<br>`}` |

| Probraná látka | Příklad |
|---|---|
| Zápis A++ znamená totéž jako A=A+1 a jde o postfixovou inkrementaci.<br><br>Zápis A-- znamená totéž jako A=A-1 a jde o postfixovou dekrementaci. | `int Cislo=0;`<br>`Cislo++; //hodnota je nyní 1`<br>`Cislo--;//hodnota je opět 0` |