



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MŠMT
MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY

Základy programování a algoritmizace

Zásobník

Erik Král



2020

Informace o autorech:

Ing. et Ing. Erik Král, Ph.D.

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Nad Stráněmi 4511

760 05 Zlín

ekral@utb.cz



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání



OBSAH

OBSAH.....	3
1 ÚVOD.....	4
1.1 OPAKOVÁNÍ POJMŮ	4
1.2 ZÁSOBNÍK (STACK)	4
1.3 ALGORITMUS VÝPOČTU KOŘENŮ KVADRATICKÉ ROVNICE.....	5
1.4 POPIS PROGRAMU Z HLEDISKA ZÁSOBNÍKU	7
1.5 ZÁSOBNÍK A PARAMETRY METODY	16
SEZNAM POUŽITÉ LITERATURY	30



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání



MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



1 ÚVOD

V tomto materiálu se seznámíme s tím, jakým způsobem program alokuje automaticky paměť pro lokální proměnné na zásobníku.

1.1 Opakování pojmů

Nejdříve se zopakujme základní pojmy:

Proměnná je pojmenovaná hodnota v paměti interpretovaná podle konkrétního datového typu.

Příkaz představuje spustitelnou část programu, pokud se provede jeden příkaz, tak se vyhodnotí všechny výrazy v něm.

Příkaz se může skládat z **výrazů**, ty se vyhodnotí teprve až se provede příkaz, ve kterém jsou použité.

Například příkaz: `int x = 5;` rezervuje místo v paměti pro celé číslo, uloží tam hodnotu 5 jako množinu bitů a této hodnotě v paměti potom říkáme *x*. Typ *int* potom určuje, jaké operace (sčítání, odčítání atd.) s proměnnou *x* může program provádět.

A nakonec máme **blok příkazů**, tedy více příkazů v jednom. V bloku příkazů se jednotlivé příkazy provádějí sekvenčně shora dolů. Blok příkazů je v jazyce C#.

1.2 Zásobník (Stack)

Každý program si alokuje po spuštění řádově megabajty paměti pro lokální proměnné, návratové adresy metod a případně další data. S touto pamětí se potom pracuje jako s datovou strukturou zásobník (Stack) a také se jí tak říká.

V následujícím příkladu si ukážeme příklad na zásobník, kdy vytvoříme několik lokálních proměnných a postupně je budeme přidávat nebo odebírat ze zásobníku.





1.3 Algoritmus výpočtu kořenů kvadratické rovnice

Konkrétně si ukážeme program pro výpočet kořenů kvadratické rovnice. Program využívá typ *double*, podmíněný výraz, výpis na konzoli s využitím *string interpolation* a statickou metodu *Math.Sqrt* pro výpočet druhé odmociny. Tyto vlastnosti jazyka probereme podrobně později. Příklad by měl být ale čitelný i pro čtenáře bez znalosti těchto vlastností.

```
using System;

double a = 1.0;
double b = 5.0;
double c = 6.0;

double D = (b * b) - (4 * a * c);

if (D > 0.0)
{
    double x1 = (-b + Math.Sqrt(D)) / (2 * a);
    double x2 = (-b - Math.Sqrt(D)) / (2 * a);

    Console.WriteLine($"x1: {x1} x2: {x2}");
}

Console.WriteLine($"D: {D}");
```

Nejprve projdeme vlastní algoritmus výpočtu a poté ten samý program z hlediska alokace proměnných na zásobníku.

Kvadratická rovnice $ax^2 + bx + c = 0$ je zadána pomocí tří členů, kvadratického, lineárního a absolutního [1].





V programu pro ně používáme typ *double*, který představuje čísla s desetinou čárkou a znaménkem:

```
double a = 1.0;  
double b = 5.0;  
double c = 6.0;
```

Nejprve vypočítáme diskriminant pomocí následujícího aritmetického výrazu a výsledek uložíme do proměnné s názvem *D*:

```
double D = (b * b) - (4 * a * c);
```

S pomocí podmíněného příkazu *if* ověříme, zda je diskriminant *D* větší než 0 a pokud ano, tak se provede blok kódu ve složených závorkách:

```
if (D > 0.0)  
{  
  
}
```

Konkrétně spočítáme dva kořeny kvadratické rovnice *x1* a *x2*. Výraz `Math.Sqrt(D)` vrací druhou odmocninu proměnné *D*.

```
double x1 = (-b + Math.Sqrt(D)) / (2 * a);  
double x2 = (-b - Math.Sqrt(D)) / (2 * a);
```

A výsledek vypíšeme na konzoli s využitím string interpolation `$"x1: {x1} x2: {x2}"`. K místo výrazů `{x1}` a `{x2}` se vloží hodnoty proměnných *x1* a *x2*.

```
Console.WriteLine($"x1: {x1} x2: {x2}");
```



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání





Nakonec vypíšeme na konzoli hodnotu proměnné D . Tento výpis bychom mohli udělat i dříve a ne až na konci programu, ale díky výpisu na konci programu se bude lépe demonstrovat princip zásobníku a rozsah platnosti proměnné.

```
Console.WriteLine($"D: {D}");
```

1.4 Popis programu z hlediska zásobníku

Nyní si projdeme program z hlediska zásobníku. Popis zásobníku bude zjednodušený a zaměří se na základní princip zásobníku a lokálních proměnných. Připomínám, že uvedené adresy v paměti jsou ilustrační a ve skutečném programu budou jiné.


Na levé straně bude vždy uvedený kód a na pravé straně objekty v paměti RAM.

Kód	RAM
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	





Pokud definujeme například proměnnou *a*, tak se pro ni alokuje místo pro typ *double* na zásobníku, například na adrese 1000.

Kód	RAM
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	 <p>The diagram shows a memory location at address 1000 labeled 'a' containing the value 1.0. The address '1000' is written in red, and the variable 'a' is written in red above the box. The value '1.0' is written in red inside the box.</p>





Protože typ `double` zabírá v paměti 8 bajtů, tak další proměnná `b` bude do paměti uložena hned za proměnnou `a`, tedy na adresu 1008.

Kód	RAM
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	<div><div>a</div><div>1000</div><div>1.0</div></div> <div><div>b</div><div>1008</div><div>5.0</div></div>





Stejným způsobem bude na zásobníku alokováno místo pro proměnnou *c*, která bude do paměti uložena hned za proměnnou *b*, tedy na adresu 1016.

Kód	RAM						
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	<table><tr><td>a</td><td>1.0</td></tr><tr><td>b</td><td>5.0</td></tr><tr><td>c</td><td>6.0</td></tr></table> <p>1000 1008 1016</p>	a	1.0	b	5.0	c	6.0
a	1.0						
b	5.0						
c	6.0						





Další postup je stejný, pro proměnnou D opět alokujeme 8 bajtů na zásobník, tentokrát na adrese 1024.

Kód	RAM																
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	<table><tr><td>a</td><td>1.0</td></tr><tr><td>1000</td><td></td></tr><tr><td>b</td><td>5.0</td></tr><tr><td>1008</td><td></td></tr><tr><td>c</td><td>6.0</td></tr><tr><td>1016</td><td></td></tr><tr><td>D</td><td>1.0</td></tr><tr><td>1024</td><td></td></tr></table>	a	1.0	1000		b	5.0	1008		c	6.0	1016		D	1.0	1024	
a	1.0																
1000																	
b	5.0																
1008																	
c	6.0																
1016																	
D	1.0																
1024																	





Další dvě proměnné $x1$ a $x2$ budou alokovány pouze pokud bude proměnná D mít hodnotu větší než 0 .

Kód	RAM																								
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	<table><tr><td>a</td><td>1.0</td></tr><tr><td>1000</td><td></td></tr><tr><td>b</td><td>5.0</td></tr><tr><td>1008</td><td></td></tr><tr><td>c</td><td>6.0</td></tr><tr><td>1016</td><td></td></tr><tr><td>D</td><td>1.0</td></tr><tr><td>1024</td><td></td></tr><tr><td>x1</td><td>-2.0</td></tr><tr><td>1032</td><td></td></tr><tr><td>x2</td><td>-3.0</td></tr><tr><td>1040</td><td></td></tr></table>	a	1.0	1000		b	5.0	1008		c	6.0	1016		D	1.0	1024		x1	-2.0	1032		x2	-3.0	1040	
a	1.0																								
1000																									
b	5.0																								
1008																									
c	6.0																								
1016																									
D	1.0																								
1024																									
x1	-2.0																								
1032																									
x2	-3.0																								
1040																									





Až skončí blok, ve kterém byly definovány proměnné *x1* a *x2*, tak budou tyto proměnné uvolněné z paměti zásobníku.

Kód	RAM
<pre>using System;</pre>	
<pre>double a = 1.0;</pre>	a 1000 1.0
<pre>double b = 5.0;</pre>	b 1008 5.0
<pre>double c = 6.0;</pre>	c 1016 6.0
<pre>double D = (b * b) - (4 * a * c);</pre>	D 1024 1.0
<pre>if (D > 0.0)</pre>	
<pre>{</pre>	
<pre> double x1 = (-b + Math.Sqrt(D)) / (2 * a);</pre>	x1 1032 -2.0
<pre> double x2 = (-b - Math.Sqrt(D)) / (2 * a);</pre>	x2 1040 -3.0
<pre> Console.WriteLine(\$"x1: {x1} x2: {x2}");</pre>	
<pre> } ←</pre>	
<pre>Console.WriteLine(\$"D: {D}");</pre>	






V zásobníku tedy zůstanou jen čtyři proměnné a poslední příkaz vypíše hodnotu proměnné *D*.

Kód	RAM								
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre>	<table><tr><td>a</td><td>1.0</td></tr><tr><td>b</td><td>5.0</td></tr><tr><td>c</td><td>6.0</td></tr><tr><td>D</td><td>1.0</td></tr></table>	a	1.0	b	5.0	c	6.0	D	1.0
a	1.0								
b	5.0								
c	6.0								
D	1.0								





Paměť pro proměnné a , b , c a d bude uvolněna na konci programu.

Kód	RAM
<pre>using System; double a = 1.0; double b = 5.0; double c = 6.0; double D = (b * b) - (4 * a * c); if (D > 0.0) { double x1 = (-b + Math.Sqrt(D)) / (2 * a); double x2 = (-b - Math.Sqrt(D)) / (2 * a); Console.WriteLine(\$"x1: {x1} x2: {x2}"); } Console.WriteLine(\$"D: {D}");</pre> 	





1.5 Zásobník a parametry metody

Na zásobník se také mohou ukládat parametry metod a návratové adresy metod, tedy adresa, od které má pokračovat provádění programu po ukončení metody. Problematika volání metod je obecně složitější a závisí na Calling Conventions (konvencích volání) a dalších optimalizacích [3]. V tomto materiálu budeme popisovat pouze **zjednodušené principy** pro pochopení základů.

Pokud bychom volali metodu rekurzivně, tedy metoda by neustále volala sama sebe, tak by časem došlo k přetečení zásobníku (stack overflow).

Následující příklad demonstruje, jakým způsobem se na zásobník ukládají parametry metody. Konkrétně budeme mít statickou metodu *Soucet*, která má dva parametry a vrací součet těchto parametrů.

Členské metody a statické metody probereme později, zatím nám budou složit jen pro demonstraci principu fungování zásobníku.





V příkladu tedy máme následující kód, který obsahuje metody *Soucet* a *Main*. Tentokrát nepoužijeme Top Level Statements [2], ale již zmíněnou metodu *Main*.

```
using System;

namespace CviceniZap
{
    class Program
    {
        static int Soucet(int a, int b)
        {
            return a + b;
        }

        static void Main(string[] args)
        {
            int a = 2;
            int b = 3;

            int s = Soucet(a, b);

            Console.WriteLine($"soucet: {s}");
        }
    }
}
```





Metoda *Soucet* má dva parametry *a*, *b* typu celého čísla, tedy *int*. Tato metoda vrací součet těchto parametrů.

```
static int Soucet(int a, int b)
{
    return a + b;
}
```

Metoda *Main* definuje dvě lokální proměnné *a*, *b* typu *int*. Je důležité si uvědomit, že jde o jiné proměnné než parametry metody. Protože rozsah platnosti lokálních proměnných i parametrů metod je jen uvnitř bloků metod ohraničených složenými závorkami, mohou se tyto proměnné jmenovat stejně a přitom jde o jiné proměnné.

```
int a = 2;
int b = 3;
```

Nakonec v programu voláme metodu *Soucet* a předáváme jí hodnoty proměnných *a*, *b* jako argumenty. Návrátová hodnota metody *Soucet* se potom přiřadí jako hodnota lokální proměnné *s*.

```
int s = Soucet(a, b);
```

Poslední příkaz `Console.WriteLine($"soucet: {s}");` vypíše hodnotu proměnné *s* na konzoli.

Na dalších stránkách projdeme postupně jednotlivé příkazy z hlediska paměti a zásobníku. Z předchozího textu už by mělo být zřejmé jak se na zásobníku rezervuje automaticky paměť pro lokální proměnné. Z příkladu bychom mohli vynechat alokaci prvních lokálních proměnných, ale pro případ, kdyby chtěl někdo studovat tento příklad samostatně bude uvedena veškerá alokace proměnných postupně.





Zobrazení zásobníku je opět schématické a adresy proměnných v příkladu jsou vymyšlené. Skutečné adresy a alokace na zásobníku v reálném programu se může lišit.

Na levé straně bude opět vždy uvedený kód a na pravé straně objekty v paměti RAM.

Kód	RAM
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	





Nejprve definujeme paměť pro proměnnou *a* v kontextu metody *Main*, to znamená že se alokuje místo pro typ *int*, tedy celé číslo na zásobníku, například na adrese 1000.

Kód	RAM
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<div>Main: a 1000</div> <div>2</div>





Protože typ *int* zabírá v paměti 4 bajty, tak další proměnná *b* bude do paměti uložena hned za proměnnou *a*, tedy na adresu 1004.

Kód	RAM
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<div> <div> Main: a 1000 </div> <div>2</div> </div> <div> <div> Main: b 1004 </div> <div>3</div> </div>





Při volání metody *Soucet* provede program skok na adresu kde jsou instrukce této metody. Aby potom věděl na jakou adresu skočit po ukončení metody, může si program uložit návratovou adresu na zásobník. Adresa zabere u 64 bitového programu 64 bitů, tedy 8 bajtů. Konkrétní implementace se ale může lišit dle platformy, překladače a souvisejících nastavení.

Kód	RAM						
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; Adresa 3000 → int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>3</td></tr><tr><td>Návratová adresa 1008</td><td>3000</td></tr></table>	Main: a 1000	2	Main: b 1004	3	Návratová adresa 1008	3000
Main: a 1000	2						
Main: b 1004	3						
Návratová adresa 1008	3000						





Nyní program alokuje paměť pro parametr *a* v kontextu metody *Main*, to znamená že se alokuje místo pro typ *int* a přiřadí mu hodnotu argumentu *a*. V reálném programu může být postup jiný, program může přímo využít registry, nebo další optimalizace a vše je závislé na platformě, překladači a dalších souvisejících nastaveních. Z hlediska principu z pohledu programovacího jazyka je pro nás důležité, že získáváme **kopii hodnoty**.

Kód	RAM								
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>3</td></tr><tr><td>Návratová adresa 1008</td><td>3000</td></tr><tr><td>Soucet: a 1016</td><td>2</td></tr></table>	Main: a 1000	2	Main: b 1004	3	Návratová adresa 1008	3000	Soucet: a 1016	2
Main: a 1000	2								
Main: b 1004	3								
Návratová adresa 1008	3000								
Soucet: a 1016	2								





Nyní program alokuje stejným způsobem paměť pro parametr *b* v kontextu metody *Main* a přiřadí mu hodnotu argumentu *b*, tedy hodnotu 3.

Kód	RAM										
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>3</td></tr><tr><td>Návratová adresa 1008</td><td>3000</td></tr><tr><td>Soucet: a 1016</td><td>2</td></tr><tr><td>Soucet: b 1020</td><td>3</td></tr></table>	Main: a 1000	2	Main: b 1004	3	Návratová adresa 1008	3000	Soucet: a 1016	2	Soucet: b 1020	3
Main: a 1000	2										
Main: b 1004	3										
Návratová adresa 1008	3000										
Soucet: a 1016	2										
Soucet: b 1020	3										





Výsledek součtu proměnných si program pro použití příkazu *return* uloží do registru procesoru a vrátí se na návratovou adresu aby pokračoval v metodě *Main*.

Kód	RAM										
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>5.0</td></tr><tr><td>Návratová adresa 1008</td><td>3000</td></tr><tr><td>Soucet: a 1016</td><td>2</td></tr><tr><td>Soucet: b 1020</td><td>3</td></tr></table>	Main: a 1000	2	Main: b 1004	5.0	Návratová adresa 1008	3000	Soucet: a 1016	2	Soucet: b 1020	3
Main: a 1000	2										
Main: b 1004	5.0										
Návratová adresa 1008	3000										
Soucet: a 1016	2										
Soucet: b 1020	3										





Parametry a , b a návratová adresa se po skončení metody *Soucet* uvolní z paměti.

Kód	RAM										
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>5.0</td></tr><tr><td>Návratová adresa 1008</td><td>3000</td></tr><tr><td>Soucet: a 1016</td><td>2</td></tr><tr><td>Soucet: b 1020</td><td>3</td></tr></table>	Main: a 1000	2	Main: b 1004	5.0	Návratová adresa 1008	3000	Soucet: a 1016	2	Soucet: b 1020	3
Main: a 1000	2										
Main: b 1004	5.0										
Návratová adresa 1008	3000										
Soucet: a 1016	2										
Soucet: b 1020	3										





Na zásobníku se potom rezervuje místo pro proměnnou *s* do které se uloží návratová hodnota vrácená z programu *Soucet* příkazem *return* (dočasně uložená v registru procesoru).

Kód	RAM						
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>5.0</td></tr><tr><td>Suma: s 1016</td><td>5</td></tr></table>	Main: a 1000	2	Main: b 1004	5.0	Suma: s 1016	5
Main: a 1000	2						
Main: b 1004	5.0						
Suma: s 1016	5						





V závěru potom program vypíše hodnotu proměnné s na konzoli.

Kód	RAM						
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	<table><tr><td>Main: a 1000</td><td>2</td></tr><tr><td>Main: b 1004</td><td>5.0</td></tr><tr><td>Soucet: s 1000</td><td>5</td></tr></table>	Main: a 1000	2	Main: b 1004	5.0	Soucet: s 1000	5
Main: a 1000	2						
Main: b 1004	5.0						
Soucet: s 1000	5						





Po ukončení metody *Main* dojde k uvolnění všech lokálních proměnných, které byly v metodě *Main* alokovány.

Kód	RAM
<pre>using System; namespace CviceniZap { class Program { static int Soucet(int a, int b) { return a + b; } static void Main(string[] args) { int a = 2; int b = 3; int s = Soucet(a, b); Console.WriteLine(\$"soucet: {s}"); } } }</pre>	





SEZNAM POUŽITÉ LITERATURY

- [1] Kvadratické rovnice – Matematika.cz. *Matematika pro střední a základní školy* – Matematika.cz [online]. Copyright © 2006 [cit. 10.12.2020]. Dostupné z: <https://matematika.cz/kvadraticke-rovnice>
- [2] What's new in C# 9.0 - C# Guide | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 22.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9#top-level-statements>
- [3] Calling Conventions | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 01.01.2021]. Dostupné z: <https://docs.microsoft.com/en-us/cpp/cpp/calling-conventions?view=msvc-160>



EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

