Zásobníkový rámec, funkce

ISU cv 8

http://www.fit.vutbr.cz/~isakin/isu

Funkce

Podprogram

= (často využívaná) skupina instrukcí. Volá se jako samostatná část programu

- `call src`
  - Instrukce CALL se používá pro volání podprogramu na adrese `src`
  - Na zásobník uloží návratovou adresu
  - Přejde na adresu `src` podprogramu
- `ret`
  - Přečte návratovou adresu ze zásobníku (na vrcholu zásobníku musí být uložena) a nahraje ji do registru EIP

Předávání parametrů funkcím

- **v globálních proměnných** (paměť, sekce .data)
  - globální proměnné = jakákoliv změna, kterou na nich funkce provede, se projeví globálně ⇒ musíme dávat pozor
- **v registrech** (eax, ebx, …)
  - registry ~ globální proměnné (problémy viz výše)
  - málo registrů ⇒ musím uchovat jejich obsah (bere paměť a čas procesoru)
- **na zásobníku** (= paměť, ale lokální)
  - řeší problém globálnosti = jejich změna se neprojeví – po ukončení funkce se parametry ze zásobníku „uklidí“

Ukázka

- cv8_A.asm

Úkoly

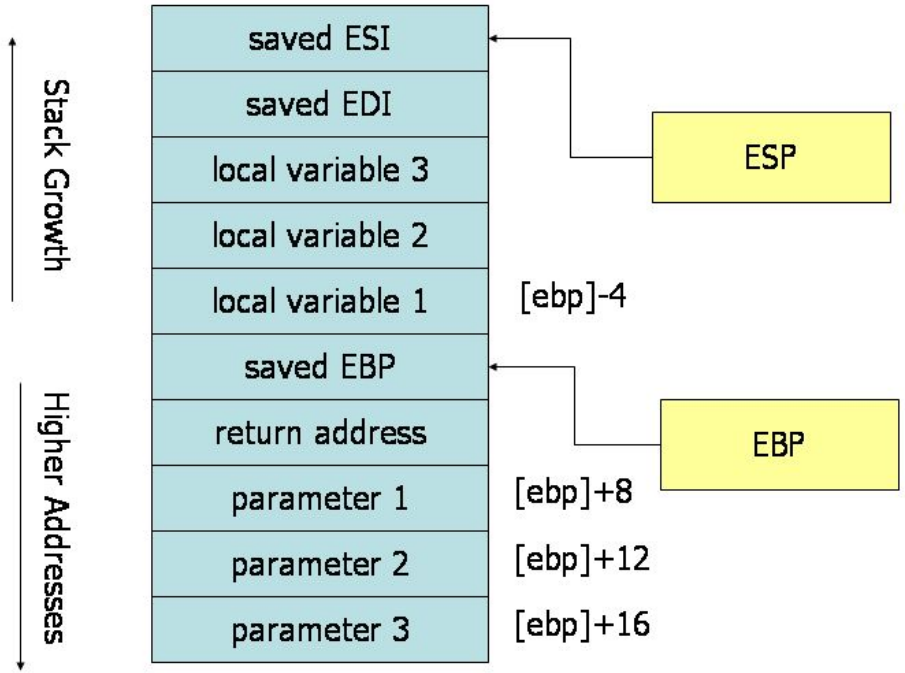
- cv8_1.asm
- cv8_2.asm

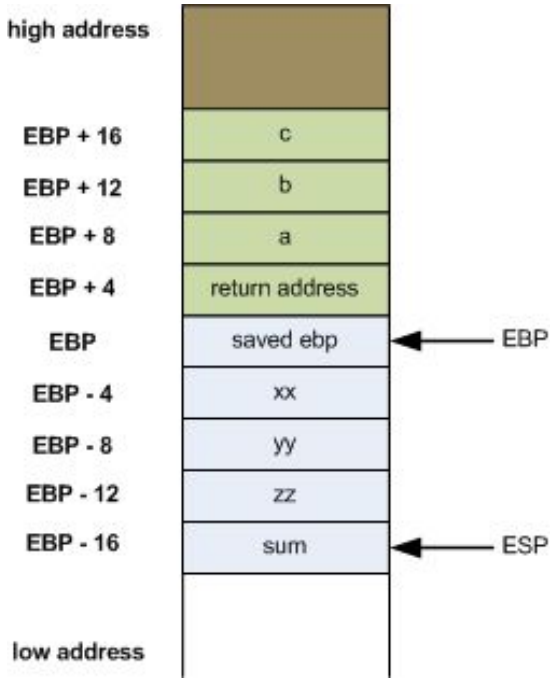
Zásobník

Zásobník

- zásobník je část paměti (když nám dojdou registry)
- se zásobníkem pracují instrukce PUSH a POP
- ze zásobníku lze číst jako z jakékoliv jiné paměti (tedy ne jen z vrcholu)
- využívaný pro zálohu registrů

Zásobník


Stack Growth
Higher Addresses
saved ESI
saved EDI
local variable 3
local variable 2
local variable 1
saved EBP
return address
parameter 1
parameter 2
parameter 3
ESP
EBP
[ebp]-4
[ebp]+8
[ebp]+12
[ebp]+16


high address
EBP + 16
EBP + 12
EBP + 8
EBP + 4
EBP
EBP - 4
EBP - 8
EBP - 12
EBP - 16
low address
c
b
a
return address
saved ebp
xx
yy
zz
sum
EBP
ESP

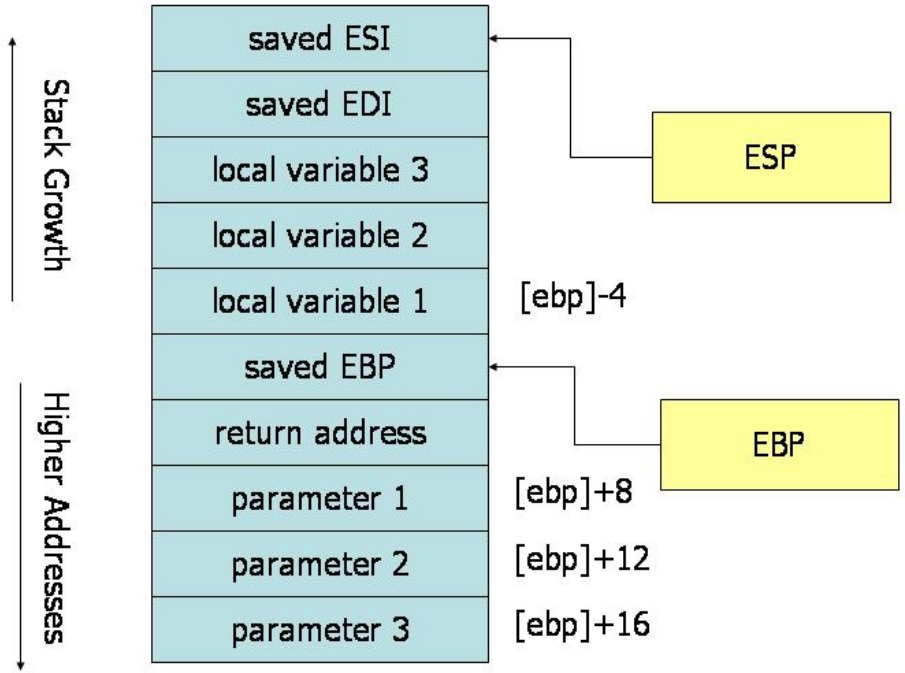
Zásobník

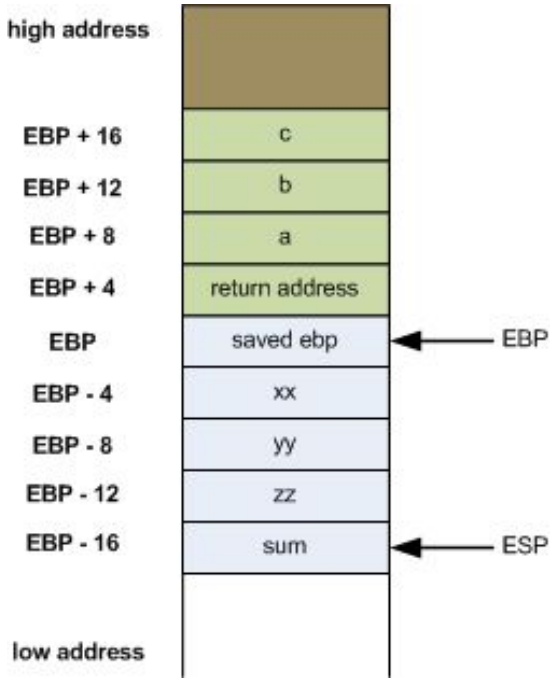
- ESP (stack pointer)
  - ukazatel na vrchol zásobníku, vždy na vrcholu, může se hýbat!
- EBP (base pointer)
  - pro “stack frame”, v rámci funkce konstantní
  - ukazuje na nějaké místo v zásobníku (např. parametry funkce)

Zásobník - organizace v paměti

- zásobník "roste" od vyšší adresy směrem k nižší
- zásobník se vždy posouvá o násobky **čtyř** ve **32**bitovém režimu, tedy minimálně o **4** byty = **32** bitů.

Zásobník


Stack Growth
Higher Addresses
saved ESI
saved EDI
local variable 3
local variable 2
local variable 1
saved EBP
return address
parameter 1
parameter 2
parameter 3
ESP
EBP
[ebp]-4
[ebp]+8
[ebp]+12
[ebp]+16


high address
EBP + 16
EBP + 12
EBP + 8
EBP + 4
EBP
EBP - 4
EBP - 8
EBP - 12
EBP - 16
low address
c
b
a
return address
saved ebp
xx
yy
zz
sum
EBP
ESP

Zásobník - instrukce

- push op
  - uloží hodnotu na vrchol zásobníku ⇒
    - sníží hodnotu ESP o 4 byty (SUB ESP,4)
    - na adresu ESP uloží hodnotu (MOV dword [ESP],hodnota)
- pop op
  - odstraní hodnotu z vrcholu zásobníku a uloží ji do cílového operandu ⇒
    - přečte hodnotu z adresy ESP a uloží do cíle (MOV dest,[ESP])
    - zvýší hodnotu ESP o 4 byty (ADD ESP,4)

Zásobník - instrukce

- pushad
  - Uloží všechny obecné registry: EAX, ECX, EDX, EBX, ESP (před instrukcí), EBP, ESI a EDI na zásobník

```
PUSH EAX
PUSH ECX
PUSH EDX
PUSH EBX
PUSH original ESP
PUSH ESI
PUSH EDI
```

- popad - obnoví stejné registry

Ukázka

- cv8_B.asm

Úkoly

- cv8_3.asm

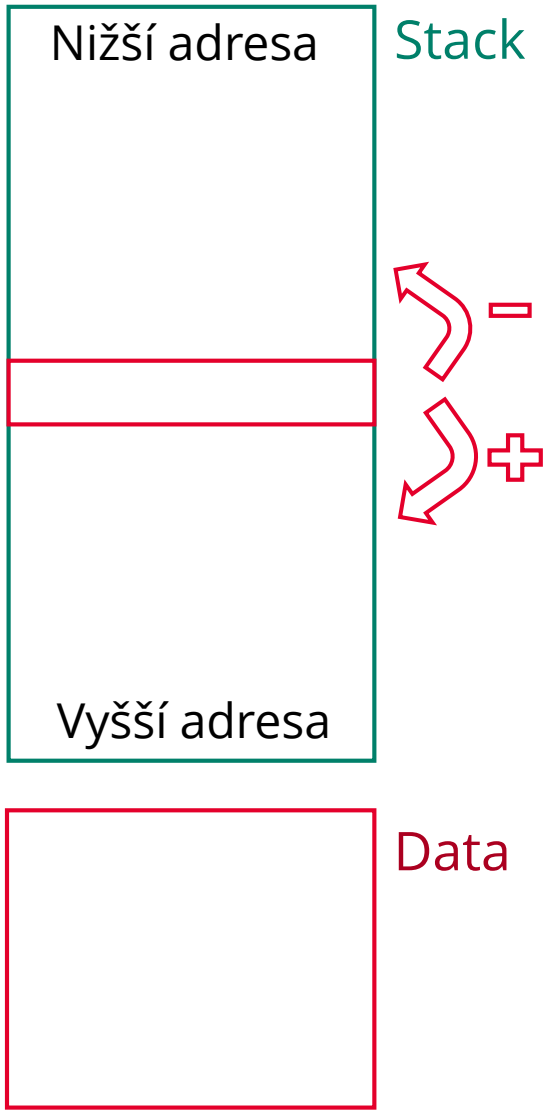
Stack frame

Stack frame (rámec zásobníku)

- Oblast mezi EBP a ESP
- Vymezení prostoru pro funkci
- Vytváří se při volání podprogramu, kdy jsou **parametry předány přes zásobník**.
- Jednotlivé prvky zásobníku jsou vyhrazeny pro:
  - Parametry podprogramu
  - Lokální proměnné podprogramu
  - Návratová adresa z volajícího podprogramu (uložený EIP)
- Obsahuje minimálně návratovou adresu.

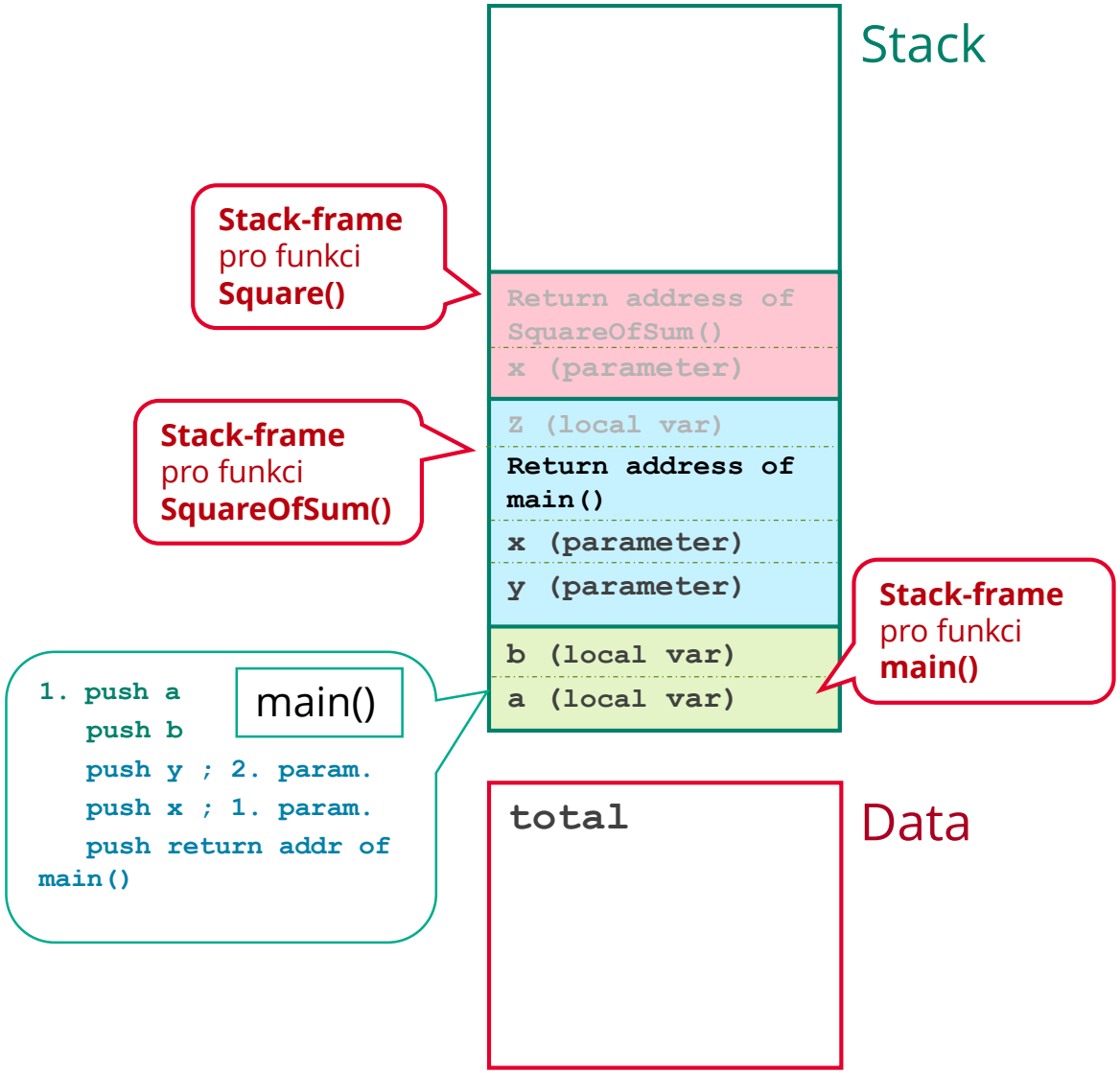
Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```



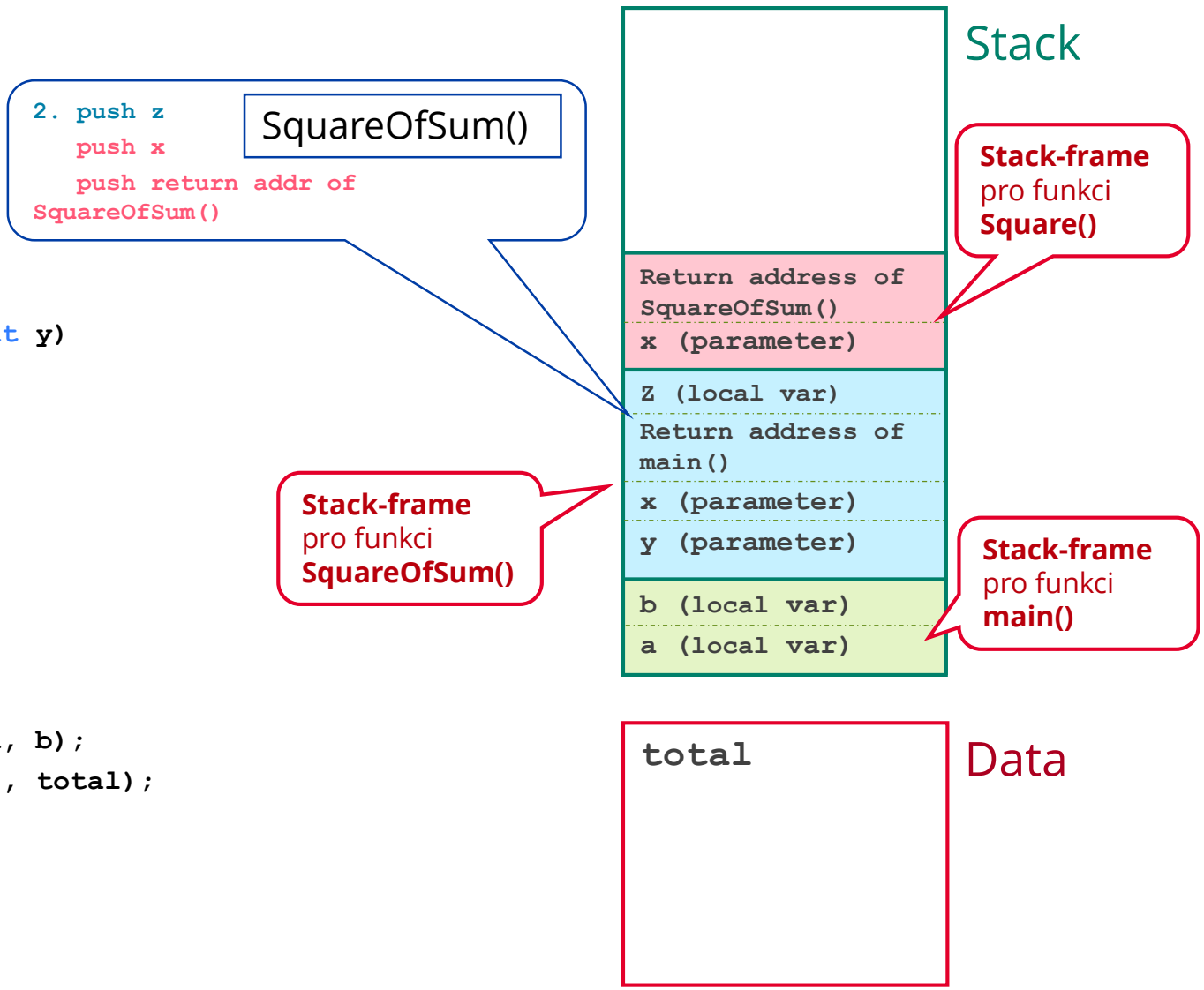
Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```



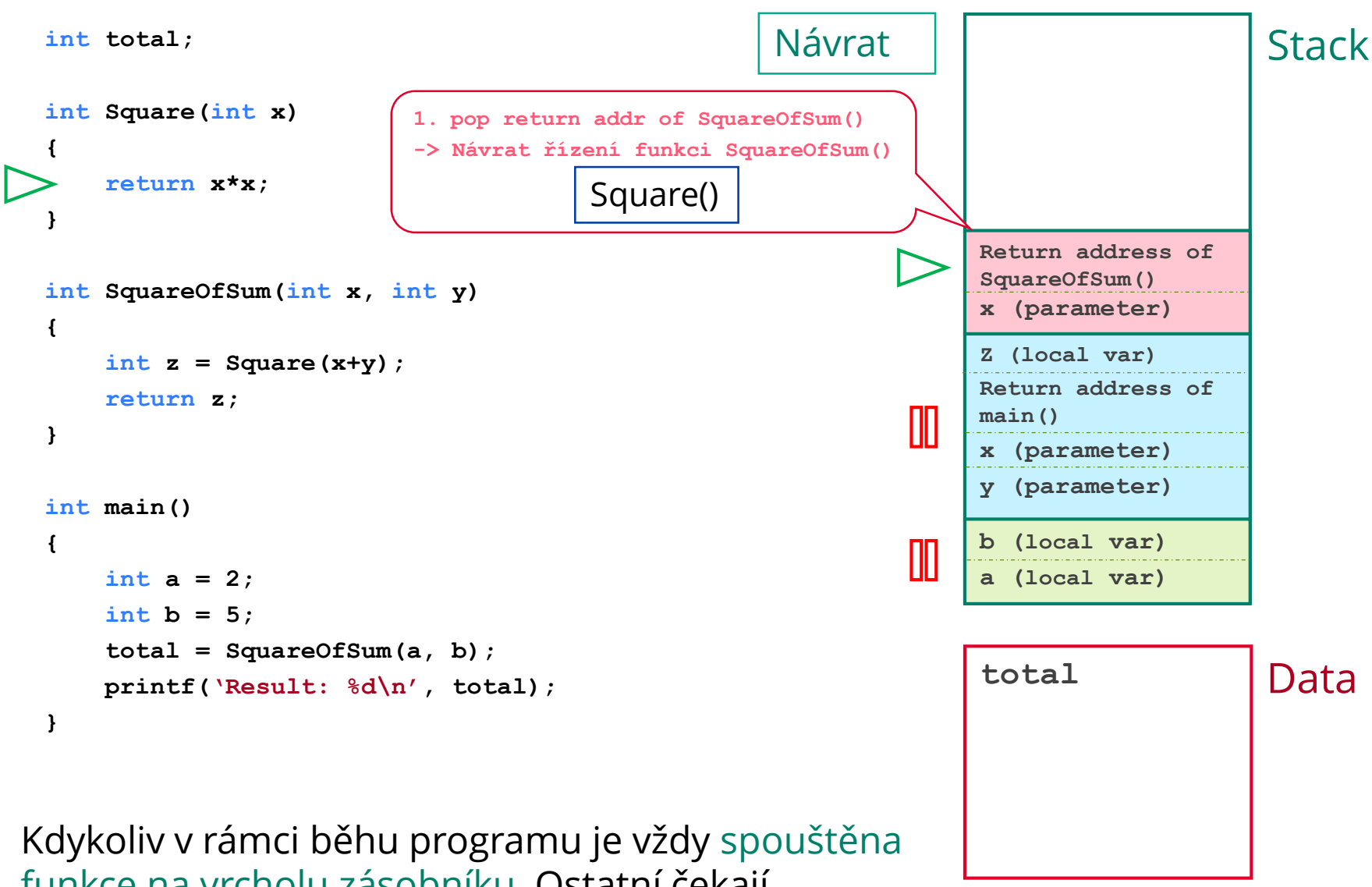
Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```



Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```



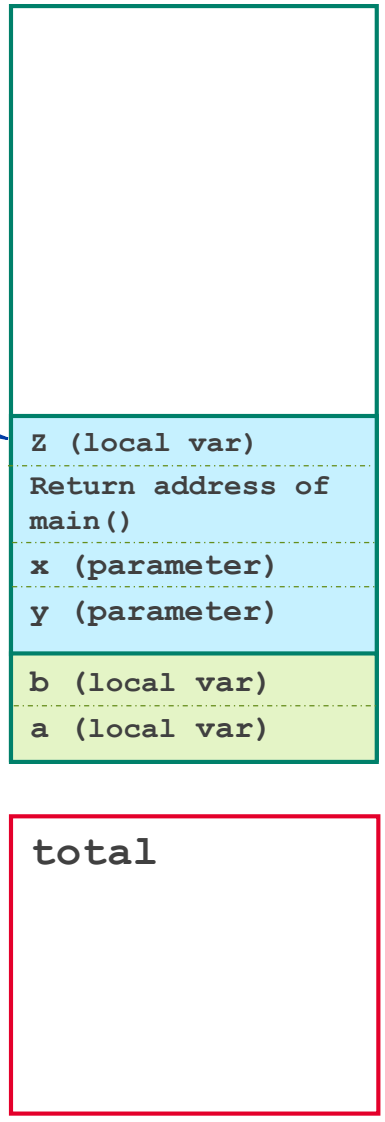
Kdykoliv v rámci běhu programu je vždy spouštěna funkce na vrcholu zásobníku. Ostatní čekají.

Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```

```
2. pop x ; clean the stack
          (za to zodpovida
          volany podprogram)
   ; návrat do main
   pop z
   pop return addr of main()
```

SquareOfSum()



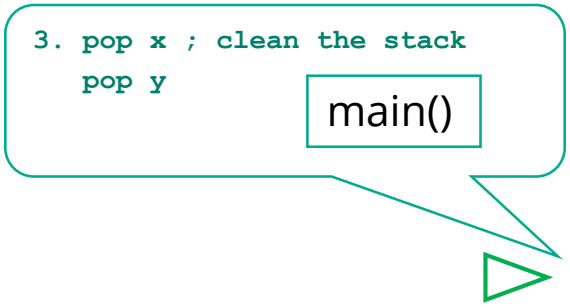
Stack

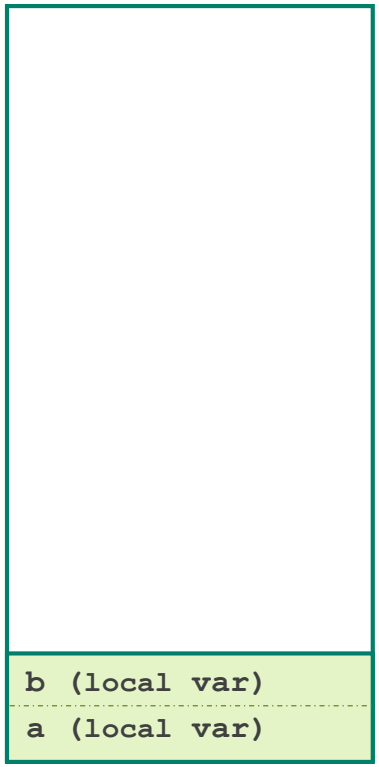
Data

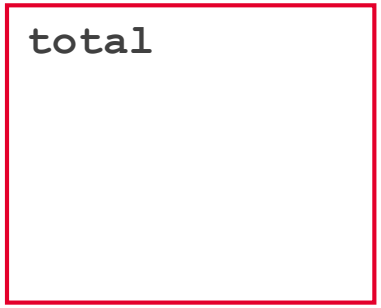
Kdykoliv v rámci běhu programu je vždy spouštěna funkce na vrcholu zásobníku. Ostatní čekají.

Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```







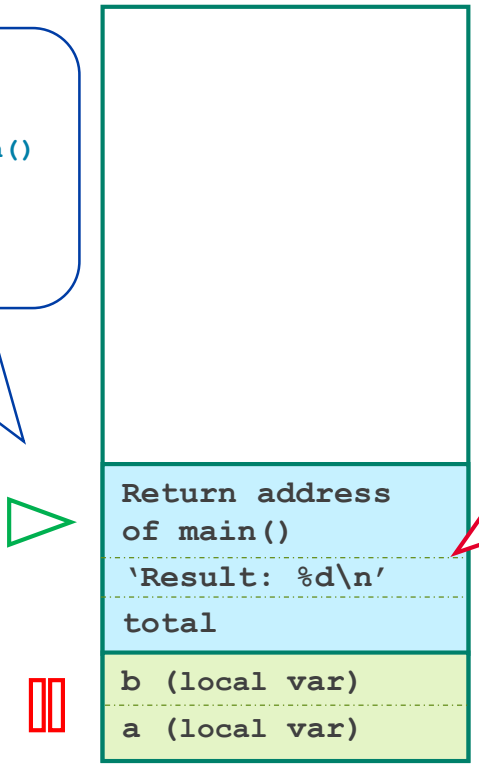
Kdykoliv v rámci běhu programu je vždy spouštěna funkce na vrcholu zásobníku. Ostatní čekají.

Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```

```
4. push par2 ; total
   push par1 ; string
   push return addr of main()
```

main()



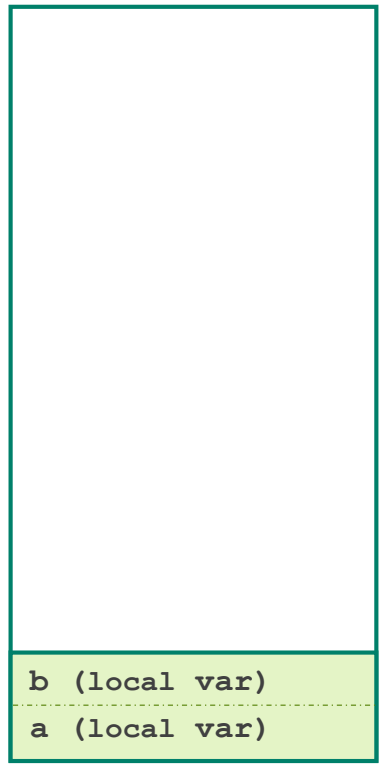
Kdykoliv v rámci běhu programu je vždy spouštěna funkce na vrcholu zásobníku. Ostatní čekají.

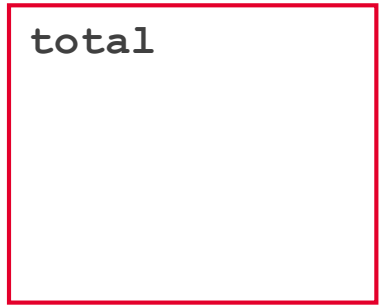
Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```

```
5. pop par1 ; clean the stack
   pop par2
```

main()

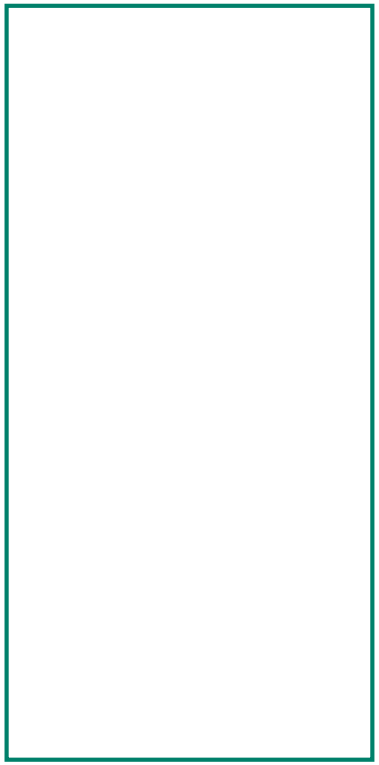




Kdykoliv v rámci běhu programu je vždy spouštěna funkce na vrcholu zásobníku. Ostatní čekají.

Stack frame - příklad

```
int total;

int Square(int x)
{
    return x*x;
}

int SquareOfSum(int x, int y)
{
    int z = Square(x+y);
    return z;
}

int main()
{
    int a = 2;
    int b = 5;
    total = SquareOfSum(a, b);
    printf('Result: %d\n', total);
}
```



Stack

Data

Kdykoliv v rámci běhu programu je vždy spouštěna funkce na vrcholu zásobníku. Ostatní čekají.

Vytvoření/zničení zásobníkového rámce

- Vytvoření zásobníkového rámce

```
push ebp    ; ulozeni EBP (používala ho volající funkce)
mov ebp, esp  ; EBP=ESP ukazuje na vrchol zasobniku
sub esp, X    ; alokace prostoru pro lokální proměnné
              ; na zásobníku, X odpovídá 4*počet_proměnných
```

- Zničení zásobníkového rámce

```
mov esp, ebp    ; obnova ESP
pop ebp    ; obnova původního EBP
ret
```

Vytvoření/zničení zásobníkového rámce

- Vytvoření zásobníkového rámce

```
enter X,0       ; local variable size, nesting level
```

- Zničení zásobníkového rámce

```
leave
ret
```

Ukázka

- cv8_C.asm

Úkoly

- cv8_4.asm
- cv8_5.asm

THE END
... for this week