Pointer na void

- = pointer bez datového typu
- lze do něj uložit libovolný pointer

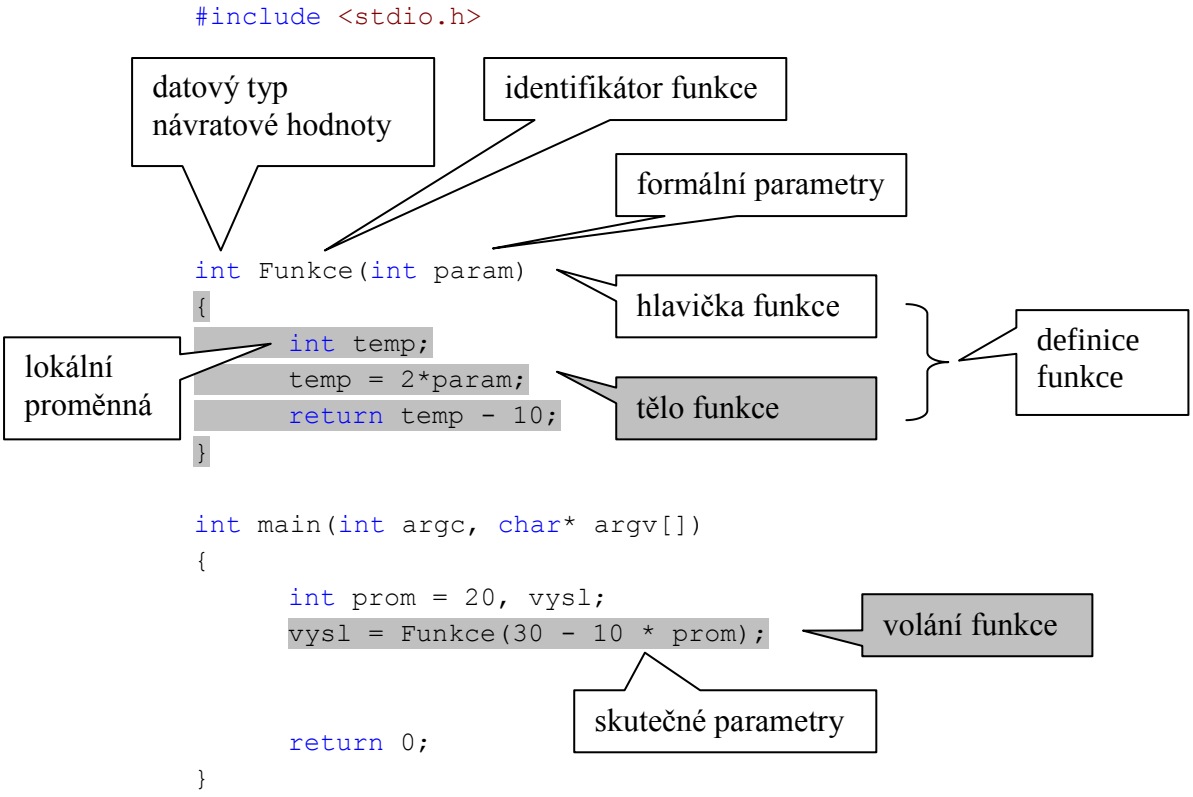
```
void *pVoid;
int promI, *pI = &promI;
double promD, *pD = &pD;

pVoid = pI;       // OK
pVoid = pD;       // OK
```

- myšlené „úrovně“ pointerů
  1. pole = adresa + datový typ + velikost pole
  2. datový pointer = adresa + datový typ
  3. pointer na void = adresa
- příklad

```
int pole[10];
int *pInt;
void *pVoid;

pInt = pole;            // v pInt neni info o delce
pInt[2] = 10;           // OK
pVoid = pInt;
//pVoid[3] = 20;        // CHYBA - no pointer arithmetics
((int *)pVoid)[3] = 22; // OK
pInt = NULL;
pInt = (int *)pVoid;    // pretypovani nutne
*((int *)pVoid+4) = 33; // OK

pInt = pole+6;          // 7. prvek
pInt[1] = 99;           // do 8. prvku
printf("%d", ++pInt);   // 8. prvek
```

Funkce

Úvod



- typy funkcí
  - bez návratové hodnoty

```
void Funkce(int param)
{
      // kod
      return;      // nebo nic
}
```

  - bez parametrů

```
Typ Funkce(void)
```

  - s proměnným počtem parametrů (později)
- datový ty návratové hodnoty
  - všechny jednoduché typy (včetně pointerů)
  - struktury, výčet, union
- (ne)využití návratové hodnoty

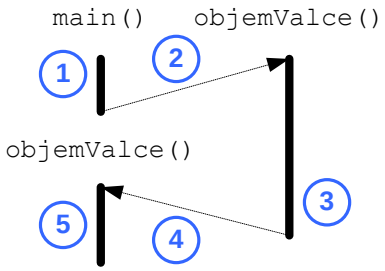
```
Funkce(30 - 10 * prom);
(void)Funkce(30 - 10 * prom);
```

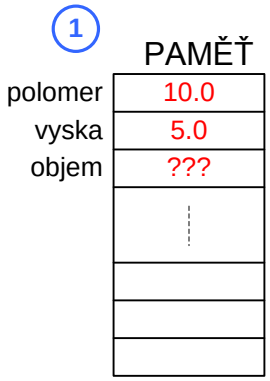
- parametry funkce – datový typ skutečného parametru = datový typ formálního parametru (popř. přetypování)

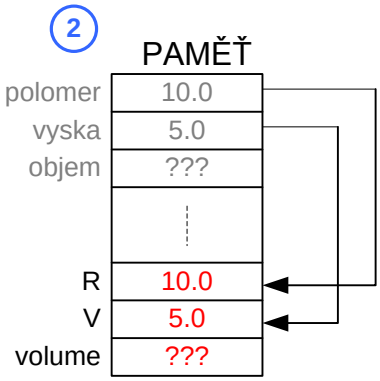
Volání hodnotou

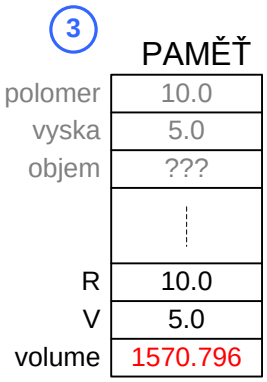
- = způsob předávání parametrů
- jazyk C = výhradní způsob volání
- Příklad a postup:

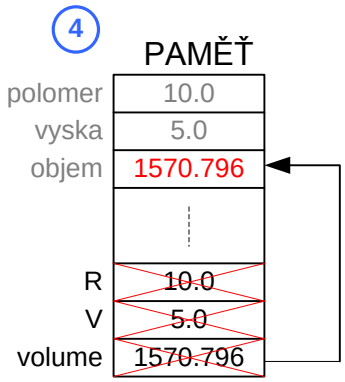
```
#include <stdio.h>
#define PI 3.141592

double ObjemValce(double R, double V)
{
      double volume;
      volume = PI*R*R*V;
      return volume;
}
int main(void)
{
      double polomer = 5, vyska = 10, objem;
      objem = ObjemValce(polomer,vyska)
      return 0;
}
```

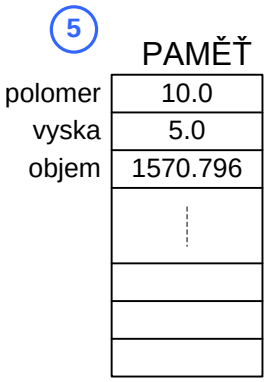








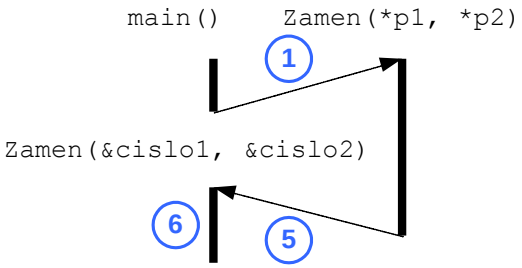


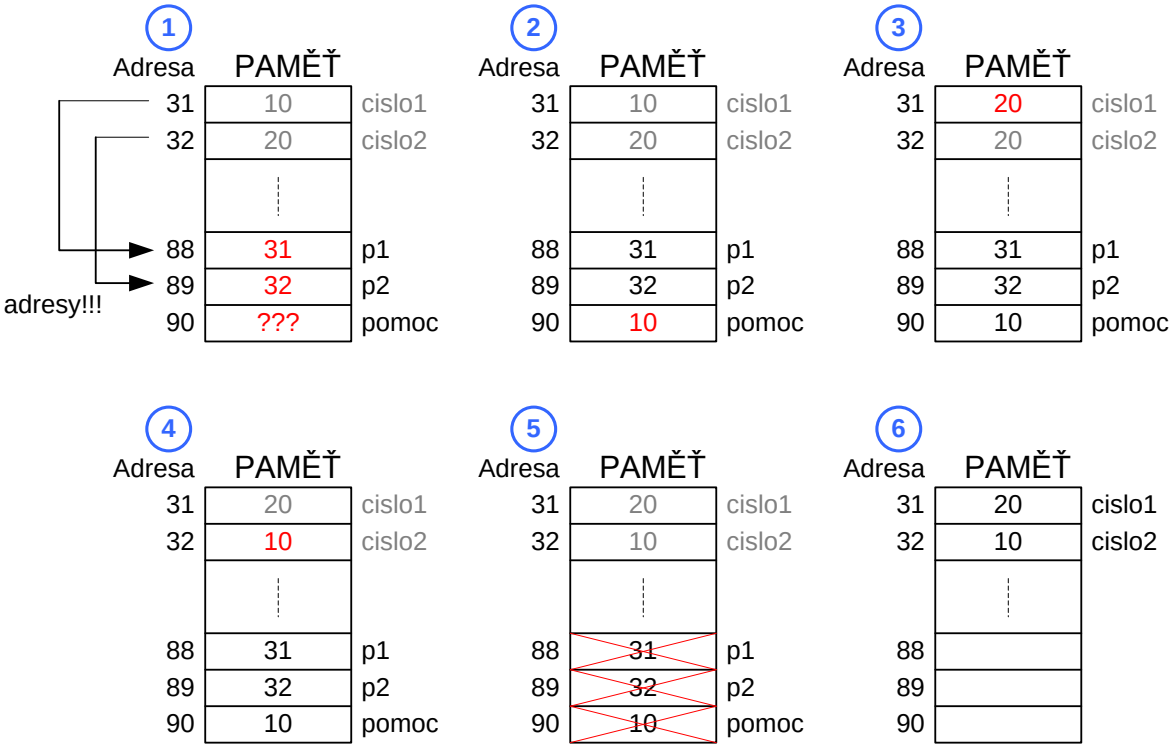


Volání „odkazem“

- pouze trik s pointery – ve skutečnosti stále volání hodnotou
  - někdy název „volání adresou“
- využití → změna hodnoty skutečného parametru ve funkci
- příklad a princip – záměna hodnot dvou proměnných

```
#include <stdio.h>

void Zamen(int *p1, int *p2)
{
        int pomoc;
        pomoc = *p1;
        *p1 = *p2;
        *p2 = pomoc;
}

int main(void)
{
        int Cislo1=10, Cislo2=20;
        Zamen(&Cislo1,&Cislo2);
        return 0;
}
```





Deklarace funkce

- překladač při kompilaci narazí na volání funkce – musí znát:
  - počet a typ vstupních parametrů
  - typ návratové hodnoty
- před prvním voláním nutno funkci:
  - definovat – viz příklady výše
  - deklarovat
- deklarace funkce

```
#include <stdio.h>

double ObjemValce(double R, double V);

int main(void)
{
	double polomer, vyska, objem;
	Objem = ObjemValce(polomer,vyska)
	return 0;
}

double ObjemValce(double R, double V)
{
	// telo funkce
}
```

úplný funkční prototyp

- obsah hlavičkových souborů (`#include <???.h>`) → převážně deklarace funkcí
- neznámá funkce v místě volání → překladač uvažuje
  - typ a počet formálních parametrů dle hlavičky volání
  - návratová hodnota `int`
- neznámá funkce příklad:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
	double vysl = Funkce(10);
	return 0;
}

double Funkce(float param)
{
	printf("%f", param);
	return 10.0;
}
```

```
warning C4013:
'Funkce' undefined;
assuming extern returning int
```

překlad volání jako `int Funkce(int)`

8

4.204e-045

Funkce a pole

- Pole jako návratová hodnota – nelze

```
#include <stdio.h>

int * Funkce()
{
      int temp[] = {1, 2, 3, 4};
      return temp;
}

int main(int argc, char* argv[])
{
      int *pole = Funkce();
      return 0;
}
```

konec funkce → `temp[]` zaniká

  - lze obejít → dynamická pole (později)

Pole jako argument funkce

- Pole jako argument funkce – předává se pointer na začátek
  - = volání „odkazem“ → pole lze ve funkci měnit

1D pole

- specifikace formálního parametru:

pointer na `pole[0]`

počet prvků

doporučeno

```
void TiskPole(int pole[], int prvku)
void TiskPole(int *pole, int prvku)
```

- příklad: kopie obsahu jednoho pole do druhého

```
#include <stdio.h>

#define POCET_PRVKU 5

void ZkopirujPole(int source[], int target[], int length)
{
      int i;
      for(i = 0; i < length; i++)
            target[i] = source[i];
}

int main(void)
{
      int zdroj[POCET_PRVKU] = {1, 2, 3, 4, 5};
      int cil[POCET_PRVKU];
      ZkopirujPole(zdroj, cil, POCET_PRVKU);
      return 0;
}
```

Vícerozměrná pole

- druhá a další dimenze pole **musí být** uvedeny jako **konstanta**
- příklad: kopie 2D pole

```
#include <stdio.h>

#define RADKU 2
#define SLOUPCU 3

void Kopie2D (int z[][SLOUPCU], int c[][SLOUPCU], int rad, int slo)
{
      int i, j;
      for(i = 0; i < rad; i++)
            for(j = 0; j < slo; j++)
                  c[i][j] = z[i][j];
}

int main(void)
{
      int zdroj[RADKU][SLOUPCU] = {{11,12,13}, {21,22,23}};
      int cil[RADKU][SLOUPCU] = {0};
      Kopie2D(zdroj, cil, 2, 2);
      return 0;
}
```

počet řádků variabilní

práce s částí pole

- důvod omezení:

```
int pole[RADKU][SLOUPCU];
pole[i][j] ↔ *(*(pole + i*SLOUPCU) + j)
```

Pointery na funkce

- ojedinělé využití (win32 API – dynamické knihovny, vícevláknové aplikace)
- příklady definic:

```
double (*pF1)(int, int);// na double f(int par1, int par2)
int *(*pF2)(void);      // na funkci int *f(void)
```

- princip:
  - definice pointeru
  - získání adresy funkce a její uložení do pointeru
  - volání funkce prostřednictvím pointeru
- Příklad

```
#include <stdio.h>

double F(int a, int b)
{
      return (double)a/b;
}

int main(void)
{
      double (* pFunkci)(int, int); // pointer na funkci
      double vysledek;
      pFunkci = F;                  // prirazeni adresy do pointeru
      vysledek = pFunkci(5,2);      // 1. zpusob pouziti
      vysledek = (* pFunkci)(10,3); // 2. zpusob pouziti
      return 0;
}
```

Složité definice

- pomůcka pro čtení (dle [Herout]):
  1. nalezneme identifikátor
  2. čteme doprava až do `)`. Po jejím nálezu přesun k odpovídající `(` a potom opět doprava → vše již přečtené přeskočit
  3. ukončující `;` → z nejlevějšího dosud přečteného místa čteme doleva
- poznámky
  - prázdné `()` nebo `()` s deklaracemi = funkce vracející …
  - `[neco]` = pole prvků typu ...
  - `*` = pointer na ...
- příklady:

```
int *x[2];        // x je | pole | pointerů na | int
double *f(int a); // f je | funkce vracející | pointer na | double
void (*n)();      // n je | pointer na | funkci vracející | void
```

- vytváření = inverze čtení
  1. zachovávat základní pravidla:

- `int funkce();` // návratová hodnota vlevo od ident.
- `int pole[];` // pole vpravo od ident.
- `int *a;` // pointer vlevo od ident

2. závorkovat

- příklady:

```
char (*a[10])();        // a je pole pointerů na funkce vracející char
double (*f())[];        // f je funkce vracející pointer na pole prvků
                        double
```

Bonus – změna pointeru ve funkci

- příklad: Záměna dvou pointerů

```
#include <stdio.h>

void Zamen(int **p1, int **p2)
{
      int *pomoc;
      pomoc = *p1;
      *p1 = *p2;
      *p2 = pomoc;
}

int main(void)
{
      int a = 1, *px = &a;
      int b = 10, *py = &b;
      Zamen(&px, &py);
      return 0;
}
```