Algoritmizácia a programovanie: 5. prednáška

Ján Grman

Obsah

1. funkcie a práca s pamäťou
2. statické jednorozmerné polia

Funkcie a práca s pamäťou

Funkcie a práca s pamäťou

- lokálne a globálne premenné
- pamäť
- funkcie

Globálne a lokálne premenné

- stanovenie kde bude premenná dostupná
  - globálne premenné
    - platnosť: od miesta definície po koniec súboru (nie programu - program sa môže skladať z viac súborov)
  - lokálne premenné
    - definované vo funkciách
    - platnosť: od definície po koniec funkcie

Príklad: globálne definície

```
#include <stdio.h>

int i;

void prva()
{...}

int j;

int druha()
{...}

void main()
{...}
```

premenná **i** je platná pre všetky 3 funkcie

premenná **j** je platná len pre funkcie: **druha()** a **main()**

Lokálne premenné

```
#include <stdio.h>

int i1, i2;

void prva()
{
   int i1, j1;
...}

int j1, j2;

int druha()
{
   int i1, j1, k1;
...}
```

globálna premenná **i1** je prekrytá lokálnou premennou **i1**

(používať sa môžu premenné: **i1**, **j1** (lokálne) a **i2** (globálna))

dve globálne premenné: **i2**, **j2** a tri lokálne premenné: **i1**, **j1**, **k1**.

Inicializácia lokálnych a globálnych premenných

- lokálne premenné:
  - nie sú automaticky inicializované
- globálne premenné:
  - automaticky inicializované na 0 (0.0, \0)

    (lepšie - nespoliehať sa na to)
  - vyhnúť sa globálnym premenným - môžu vniesť zmätok do väčších programov!

Alokácia pamäte

- každá premenná musí mať v čase svojej existencie pridelený pamäťový priestor
- akcia na vyhradenie pamäťového priestoru sa nazýva *alokácia*, ktorá môže byť
  - statická
  - dynamická

Statická alokácia pamäte

- keď vieme prekladaču vopred povedať, aké máme na premenné pamäťové nároky
  - napr. vieme, že budeme potrebovať dve prenenné typu **`double`** a jednu premennú typu **`char`**
- prekladač sám určí požiadavky pre všetky definované premenné a pri spustení programu sa pre ne alokuje miesto
- behom programu sa nemanipuluje s touto pamäťou
- premenné majú alokované miesto od začiatku programu do jeho konca
- ruší ich operačný systém

Statická alokácia pamäte

- vymedzuje miesto v dátovej oblasti
- globálne premenné - statické
- nie vždy to stačí
  - napr. rekurzia alebo do pamäte potrebujeme načítať obsah súboru
  - použiť dynamickú alokáciu, alebo vymedzenie pamäte v zásobníku

Dynamická alokácia

- vymedzenie pamäte v hromade (heap)
- za behu programu dynamicky prideliť (alokovať) oblasť pamäte určitej veľkosti
- pristupuje sa do nej prostredníctvom ukazovateľov

Vymedzenie pamäte v zásobníku

- zaisťuje kompilátor pri volaní funkcie
- väčšina lokálnych premenných definovaných vo funkciách
- existencia týchto premenných začína pri vstupe do funkcie a končí pri výstupe z funkcie
- ak chceme prenášať hodnotu premennej medzi jednotlivými volaniami funkcie - nemôže byť premenná alokovaná v zásobníku

Funkcie

- jazyk C je založený na funkciách
  - kratšie programy majú jednu funkciu **`main()`**
  - väčšina má viac funkcií
- spracovanie programu
  - začína volaním funkcie **`main()`**
  - končí opustením funkcie **`main()`**
- funkcie nemôžu byť vhniezdené
- nie procedúry - všetky funkcie vracajú hodnotu
  - dajú sa použiť aj ako procedúry (vrátia **`void`**)

Definícia funkcie

- definícia: určuje hlavičku aj telo funkcie
- deklarácia: len špecifikuje hlavičku funkcie (meno, fyp návratovej hodhoty, parametre)
  - hlavička funkcie:

```
int max(int a, int b)
```

  - definícia:

```
int max(int a, int b)
{
    return (a > b ? a : b);
}
```

`return h;` - funkcia vráti hodnotu **h**

  - volanie funkcie:

```
x = max(10 * i, j - 15);
```

Funkcia bez parametrov

– definícia funkcie:

```
int scitaj()
{
   int a, b;

   scanf("%d %d", &a, &b);
   return (a + b);
}
```

– volanie funkcie:

```
j = scitaj();
```

Procedúry a dátový typ `void`

- formálne procedúry neexistujú, dá sa to obísť:

1. funkcia návratovú hodnotu vracia, ale nepotrebujeme ju, napr. čakanie na stlačenie klávesy (bez toho, aby nás zaujímalo, aká klávesa bola stlačená)

```
getchar();
```

čakanie na stlačenie klávesy

```
(void) getchar();
```

čitateľnejšie, niektoré prekladače to vyžadujú

Procedúry a dátový typ `void`

2. funkcia sa definuje ako funkcia vracajúca typ **`void`** (nič), napr.

```
void vypis_int(int i)
{
   printf("%d", i);
}
```

- volanie procedúry (funkcie):

```
vypis_int(a + b);
```

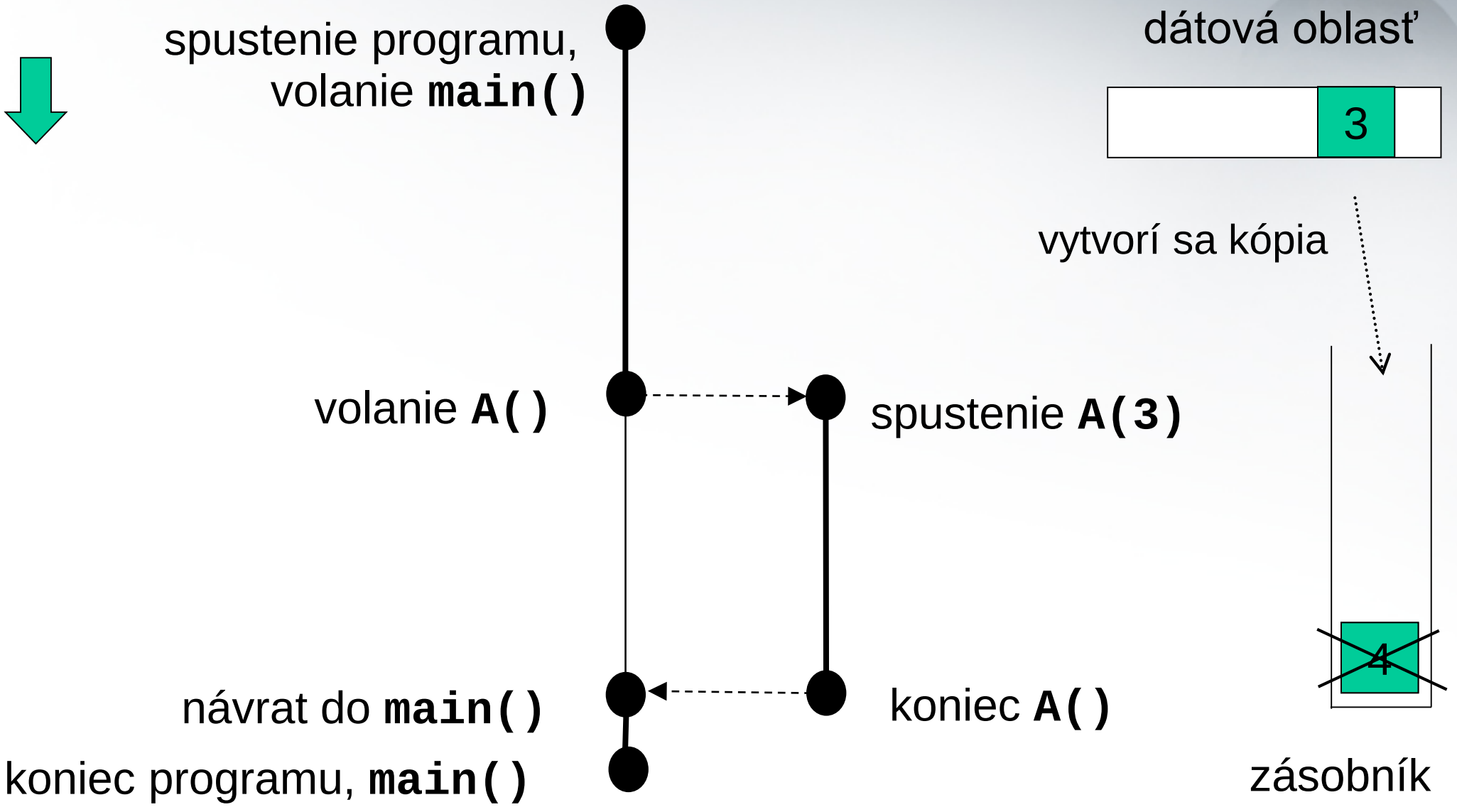
Parametre funkcií - volanie hodnotou

- predávanie parametrov hodnotou
  - parametre sú vo funkcii len čítané
  - každá zmena parametra je dočasná, je len v rámci funkcie a po jej ukončení sa stratí

- ako funguje:
  - vytvorí sa lokálna kópia premennej v zásobníku a vo funkcii sa pracuje len s ňou
  - na konci funkcie sa lokálna kópia stráca

príklad: volanie funkcie **`int A(...)`** s parametrom **3**, ktorý sa vo funkcii zmení na **4** →

Parametre funkcií - volanie hodnotou



Príklad 1: použitie funkcie v programe


```
#include <stdio.h>

int max(int a, int b) {
   return (a > b ? a : b);
}

int main() {
   int x, y;

   printf("Zadajte 2 cisla: ");
   scanf("%d %d", &x, &y);
   printf("Maximum: %d\n", max(x, y));

   return 0;
}
```

Príklad 2: použitie funkcie v programe

```
#include <stdio.h>
#define N 5

int max(int a, int b) {
   return (a > b ? a : b);
}

int main() {
   int i, x, y;

   for (i=1; i<=N; i++) {
      printf("[%d] zadajte 2 cisla: ", i);
      scanf("%d %d", &x, &y);
      printf("Maximum: %d\n", max(x, y));
   }
   return 0;
}
```

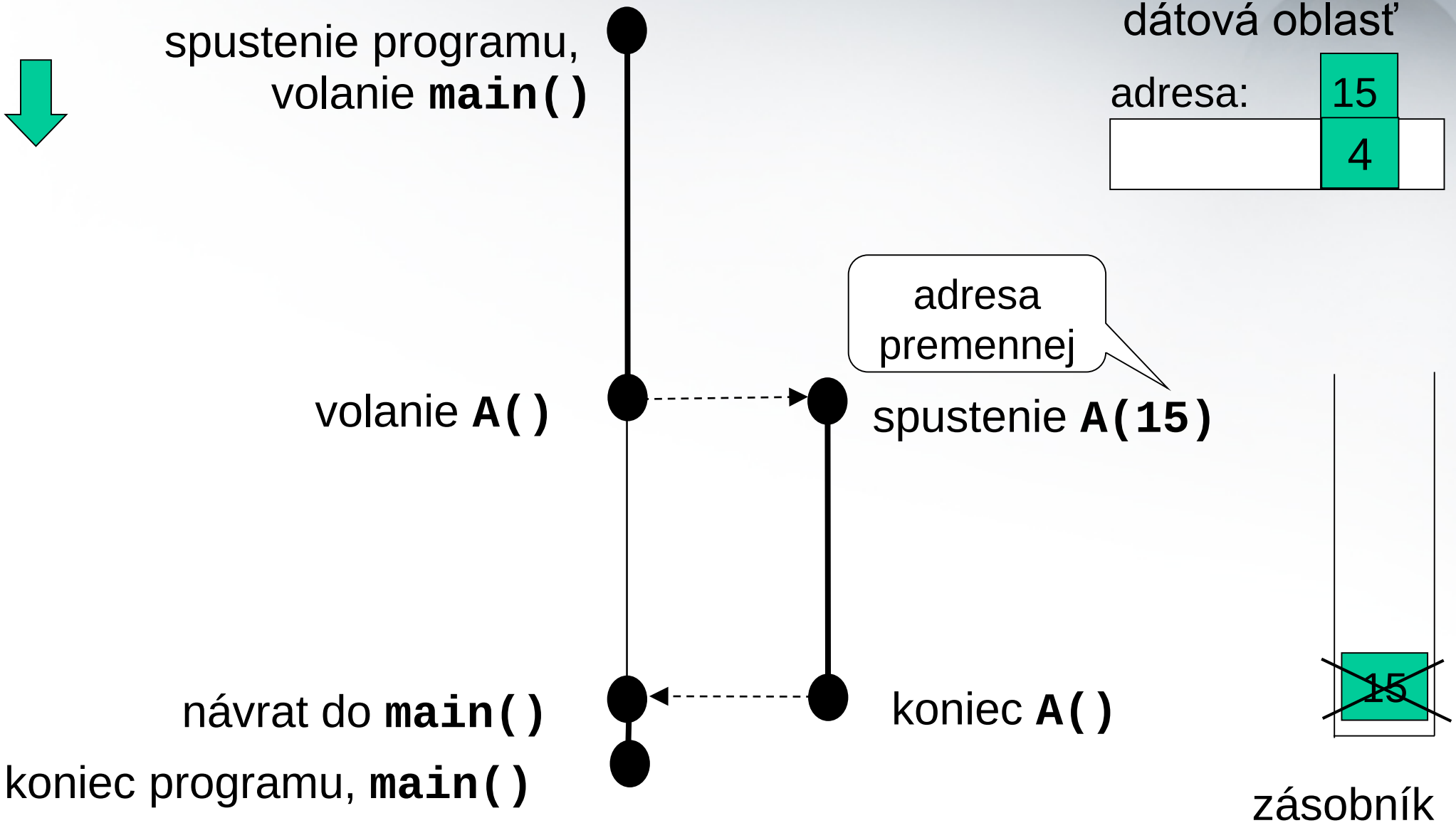
maximum z dvoch čísel - volanie funkcie viackrát

Parametre funkcií - volanie odkazom

- predávanie parametrov odkazom neexistuje v C
  - volanie odkazom by umožnilo meniť parametre v rámci funkcie
  - rieši sa pomocou ukazovateľov
  - ukazovateľ určuje, na ktorom mieste v dátovej pamäti sa má premenná zmeniť (nemení sa ukazovateľ - adresa)

príklad: volanie funkcie **`int A(...)`** s adresou premennej, ktorej hodnota je **3**, Vo funkcii sa zmení hodnota premennej na **4** →

Parametre funkcií - volanie odkazom



```c
#include <stdio.h>

#define PI 3.14
#define na_druhu(i) ((i) * (i))

void kruh(int r, float *o, float *s)
{
   *o = 2 * PI * r;
   *s = PI * na_druhu(r);
}

int main()
{
   int polomer;
   float obvod, obsah;

   printf("Zadaj polomer kruhu: ");
   scanf("%d", &polomer);

   kruh(polomer, &obvod, &obsah);
   printf("obvod: %.2f, obsah: %.2f\n", obvod, obsah);
   return 0;
}
```

Funkcia vypočíta obvod a obsah kruhu vo funkcii kruh( ). Volanie odkazom.

Príklad funkcie: výmena premenných

```
void vymen(int *p_x, int *p_y)
{
    int pom;

    pom = *p_x;
    *p_x = *p_y;
    *p_y = pom;
}
```

i: 7

j: 5

- volanie funkcie: **`vymen(&i, &j)`**

Príklad funkcie: výmena premenných

- volanie funkcie: `vymen(&i, &j)`

```
vymen(i, j);
```

chyba: vymieňa obsah adries, daných obsahom `i`, `j`: vymieňa hodnoty na adresách 5 a 7

```
vymen(*i, *j);
```

chyba: vymieňa adresy adries z obsahu `i`, `j`: z adries 5 a 7 sa zoberú hodnoty a tie sa použijú ako adresy

Vzájomné volanie funkcií

Príklad: funkcie – násobenie a dielenie

program načíta celé číslo, potom umožní používateľovi v cykle číslo násobiť dvoma, deliť troma, vypísať číslo - pokým používateľ program neukončí

```c
#include <stdio.h>

int nasob_2(int x);
int del_3(int x);

int main() {
   int i, c;
   printf("Zadajte cele cislo: ");
   scanf("%d", &i);

   do {
      printf("\ncislo ma hodnotu: %d\n\n", i);
      printf("stlacte N na vynasobie cisla dvoma.\n");
      printf("stlacte D na vydelenie cisla troma.\n");
      printf("stlacte K na ukoncenie programu.\n");
      c = getch();

      if (c == 'n' || c == 'N')
         i = nasob_2(i);
      else if (c == 'd' || c == 'D')
         i = del_3(i);
   } while (c != 'k' && c != 'K');
   return 0;
}
```

```c
int nasob_2(int x) {
   return x * 2;
}

int del_3(int x) {
   return x / 3;
}
```

Príklad: navzájom sa odkazujúce funkcie

program vypočíta hodnotu funkcií **p(x)** a **q(x)** pre dané **x**.

$$p(x) = \begin{cases} p(x-1) + q(x/2) & \text{ak } x > 1 \\ 2 & \text{ak } x <= 1 \end{cases}$$

$$q(x) = \begin{cases} q(x-3) + p(x-5) & \text{ak } x > 3 \\ x/3 & \text{ak } x <= 3 \end{cases}$$

```c
#include <stdio.h>

float q(float x);

float p(float x) {
   if (x <= 1)
      return 2.0;
   return (p(x-1) + q(x/2));
}

float q(float x) {
   if (x <= 3)
      return x / (float) 3.0;
   return (q(x-3) * p(x-5));
}
```

```c
int main() {
   float x;

   do {
      printf("Zadajte realne cislo (konec pri zadani -1.0)\n");
      scanf("%f", &x);
      if (x == -1.0) break;
      printf("\np(%.3f) = %.3f\n", x, p(x));
      printf("q(%.3f) = %.3f\n\n", x, q(x));
   } while (1);
}
```

Príklad: faktoriál - iteratívne

Faktoriál - iteratívne

```
#include <stdio.h>

... /* definicia funkcie
       faktorial */

void main ()
{
   int n;

   printf("Zadajte cele cislo: ");
   scanf("%d", &n);
   printf("%d! = %d\n", n, faktorial(n));
}
```

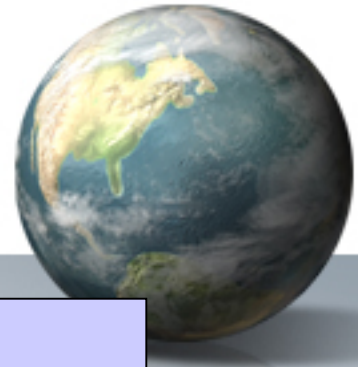
```
long faktorial(long n)
{
   if (n <= 0)
      return 1;
   else {
      int i, f=1;

      for(i=1; i<=n; i++)
         f *= i;
      return f;
   }
}
```

Príklad: výpis súboru po stránkach

program opisuje text zo súboru **`subor.txt`** na obrazovku s tým, že po vypísaní jednej stránky čaká na stlačenie klávesy **`<Enter>`**

Príklad: výpis súboru po stránkach



```
#include <stdio.h>

#define RIADKY_OBR  20
#define MENO "subor.txt"

void vypis(FILE *fr);

int main(void) {
   FILE *fr;

   if ((fr = fopen(MENO, "r")) == NULL) {
      printf("Subor %s nebol otevoreny.\n", MENO);
      return 1;
   }

   vypis(fr);
   if (fclose(fr) == EOF)
      printf("Subor %s nebol zatvoreny.\n", MENO);
   return 0;
}
```

Úplný funkčný prototyp

Príklad: výpis súboru po stránkach

pokračovanie:

```
void vypis(FILE *fr) {
  int c, pocet = 0;

  while ((c = getc(fr)) != EOF) {
    putchar(c);
    if (c == '\n') {
      if (++pocet >= RIADKY_OBR) {
        pocet = 0;
        while (getchar() != '\n')
          ;
      }
    }
  }
}
```

Čaká na odriadkovanie, až potom vypisuje ďalšiu stránku

Jednorozmerné polia

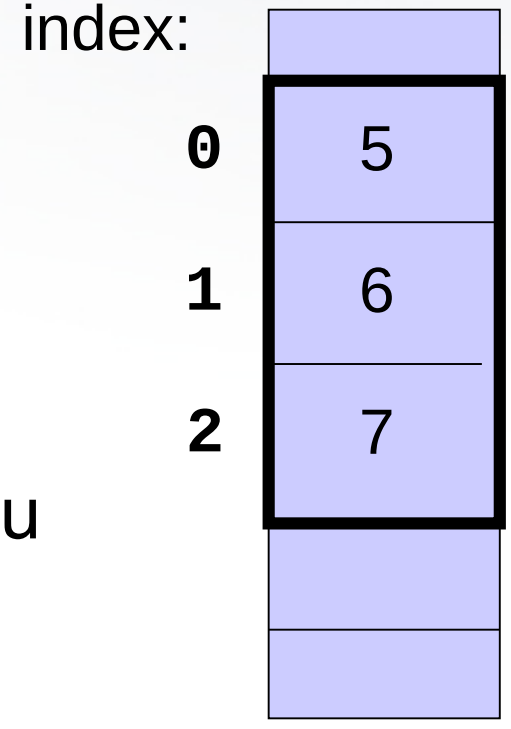
Základy práce s poliami

- pole je štruktúra zložená z niekoľkých prvkov rovnakého typu (blok prvkov)

```
TYP x[N];
```

statická definícia poľa

N: 3
TYP: **int**



  - pole obsahuje **N** prvkov
  - dolná hranica je vždy **0**
    ⇒ horná hranica je **N-1**
  - číslo **N** musí byť známe v čase prekladu
  - hodnoty nie sú inicializované na 0

Príklady definícií statického poľa

```
#define N 10

int x[N], y[N+1], z[N*2];
```

x má 10 prvkov poľa, od indexu 0 po index 9
y má 11 prvkov poľa, od indexu 0 po index 10
z má 20 prvkov poľa, od indexu 0 po index 19

Prístup k prvkom poľa

```
#define N 10

...
   int x[N], i;

   x[0] = 1;

   for (i = 0; i < N; i++)
      x[i] = i+1;

   for (i = 0; i < N; i++)
      pritnf("x[%d]: %d\n", i, x[i]);
```

priradenie hodnoty do prvého prvku poľa

v cykle priradenie hodnoty postupne všetkým prvkom poľa

výpis prvkov poľa

Príklad statického poľa: histogram písmen v reťazci

```
#include <stdio.h>
#include <stdlib.h>
#define N ('Z' - 'A' + 1)

int main() {
   int i;
   char hist[N], slovo[100];

   scanf("%s", slovo);          /* nacitanie slova */
   for (i = 0; i < N; i++)      /* inicializacia hist */
      hist[i] = 0;

   i = 0;
   /* naplnenie hist */
   while (i < 100 && slovo[i] != '\0') {
      hist[toupper(slovo[i]) - 'A']++;
      i++;
   }
```

pokračovanie

Príklad statického poľa: histogram písmen v reťazci

pokračovanie:

```
    for(i = 0; i < N; i++)    /* vypis hist */
       if(hist[i] != 0)
          printf("%c: %d\n", i+'A', hist[i]);

    return 0;
}
```

Inicializácia poľa v definícii

```
...
int x[3] = { 1, 2, 3 };
int y[] = { 1, 2, 3 };
int z[5] = { 1, 2, 3 };
...


int n = 5;
int z[n];
```

Počet prvkov poľa je daný počtom hodnôt

Hodnoty **z[3]** a **z[4]** sú inicializované na 0.

Nie je povolené, keďže **n** je premenná a nie číslo alebo konštanta (t.j. hodnota nemusí byť známa v čase prekladu)

Zistenie veľkosti poľa

```
int x[10];
```

- **x** je statický ukazovateľ

  sizeof(x) == 10 * sizeof(int) (napr. 20)

```
int i;
int pole[]= { 3, 6, 9, 12, 15 };

for (i=0; i< (sizeof(pole)/sizeof(int)); i++) {
  printf("%d", pole[i] );
}
```

Počet prvkov poľa

Pole ako parameter funkcie

vráti maximum z prvkov poľa **`pole`**

- identifikátor nasledovaný zátvorkami:

```
int pole[]
```

```
int maximum(int pole[], int n)
{
    int i, max = pole[0];
    for (i = 1; i < n; i++) {
        if (pole[i] > max)
            max = pole[i];
    }
    return max;
}
```

nepozná veľkosť poľa, preto ju treba uviesť

Pole ako parameter funkcie: veľkosť poľa

```
int maximum(int pole[], int n)
```

- vo funkcii sa nedá zistiť veľkosť poľa aj keď:

```
int maximum(int pole[10])
```

parameter sa dá vo funkcii meniť (lebo sa vytvorila jeho lokálna kópia (nezáleží na tom, či ide o statické alebo dynamické pole)

- parameter bude stále považovaný za **`pole[]`**

Pole ako parameter funkcie

```
int pole[]
```

Volanie funkcie s poľom ako parametrom:

```
max = maximum(pole, 10);
```

Príklady

Výpočet študijného priemeru

```
#include<stdio.h>
#define MAX 10

int main() {
   int i;
   float znamky[MAX];
   float priemer = 0;

   for (i = 0 ; i < MAX; i++) {
      printf("Aka je znamka z %d-teho predmetu?", i+1);
      scanf("%f", &znamky[i]);

      priemer += znamky[i];
   }
   priemer /= MAX;
   printf("Priemer znamok je %.2f.", priemer);

   return 0;
}
```

Nájdenie výskytu hodnoty v poli.

```
#include<stdio.h>

int main() {
   int x[] = {12,67,56,60,88,34,123};
   int hodnota, velkost, i, najdene = -1;
   scanf("%d", &hodnota);
   velkost = sizeof(x) / sizeof(int);

   i = 0;
   while ( najdene < 0 && i < velkost ) {
      if ( x[i] == hodnota )
         najdene = i;
      else
         i++;
   }

   if (najdene != -1)
      printf ("%d je na pozicii %d.\n", hodnota, i);
   else
      printf ("%d sa v poli nenachadza.\n", hodnota);
   return 0;
}
```

Nájdenie všetkých výskytov hodnoty v poli.

```
#include<stdio.h>

int main() {
   int x[] = {12,67,56,60,88,34,123};
   int hodnota, velkost, i, najdene = -1;
   scanf("%d", &hodnota);
   velkost = sizeof(x) / sizeof(int);

   for(i=0; i<velkost; i++)
      if(x[i] == hodnota)
         if(najdene == -1) {
            printf("%d je na pozicii %d", hodnota, i);
            najdene = 1;
         }
         else
            printf(", %d", i);


   if (najdene == -1)
      printf("%d sa v poli nenachadza.\n", hodnota);
   else
      printf("\n");
   return 0;
}
```

Príklad: porovnanie polí znakov

Funkcia zistí, či dva reťazce (polia znakov) sú rovnaké. Pri porovnaní sa ignoruje veľkosť písmen.

```
int rovnake_retazce(char s1[], int n1, char s2[], int n2) {
   int i = 0;

   if(n1 != n2)
      return 0;
   while(i < n1) {
      if(toupper(s1[i]) != toupper(s2[i]))
         break;
      i++;
   }

   if (i == n1) return 1;
   else return 0;
}
```

Príklad: otočenie veľkého čísla

Napíšte program, ktorý otočí číslo (napr. číslo 123 po otočení je 321), pričom uvažujte veľmi dlhé čísla, ktoré sa nezmestia ani do long int (max. 1000 číslic)

```c
#include<stdio.h>
#define MAX 1000

int main(){
   char num[MAX];
   int i=0, j, flag=0;

   printf("Nacitajte kladne cislo: ");
   scanf("%s", num);
   while(num[i] != '\0'){
      if(num[i] < '0' || num[i] > '9'){
         printf("Nespravne cislo.");
         return 0;
      }
      i++;
   }

   printf("Obratene cislo: ");
   for(j=i-1; j>=0; j--)
      if(flag == 0 &&  num[j] == '0')
         ;
      else {
         printf("%c", num[j]);
         flag = 1;
      }
   return 0;
}
```

Príklad: histogram

Program vytvorí histogram výskytov písmen v súbore (pre každé písmeno - počet jeho výskytov)

```c
#include<stdio.h>
#include<stdlib.h>

#define SUBOR "pismena.txt"
#define N 'Z' - 'A' + 1    /* pocet pismen abecedy */

int main()
{
   int c, i, hist[N];
   FILE *fr;

   if((fr = fopen(SUBOR, "r")) == NULL) {
      printf("Subor sa nepodarilo otvorit.\n");
      return 1;
   }

   for (i=0; i<N; i++)
      hist[i] = 0;

   while ((c = toupper(getc(fr))) != EOF) {
      if(c >= 'A' && c <= 'Z')
         hist[c - 'A']++;
   }
```

→ pokračovanie

Príklad: histogram

pokračovanie:

```
    for (i=0; i<N; i++)
       if (hist[i] != 0)
          printf("%c: %2d\n", 'A' + i, hist[i]);

    if(fclose(fr) == EOF) {
       printf("Subor sa nepodarilo zatvorit.\n");
       return 1;
    }

    return 0;
}
```

Príklad: vloženie prvku do poľa

Program načíta do poľa celé čísla a
vloží zadané číslo na zadanú pozíciu

```
#include<stdio.h>

   int main(){
   int a[50], velkost, cislo, i, poz;

   printf("Zadajte velkost pola: ");
   scanf("%d", &velkost);
   printf("Zadajte %d prvkov pola:\n", velkost);
   for(i=0; i<velkost; i++)
      scanf("%d", &a[i]);

   printf("Zadajte poziciu a cislo na vlozenie: ");
   scanf("%d %d", &poz, &cislo);

   i = velkost++;
   while(i > poz) {
      a[i] = a[i-1];
      i--;
   }
   a[i] = cislo;

   printf("Pole po vlozeni prvku:\n");
   for(i=0; i<velkost; i++)
      printf(" %d",a[i]);
   return 0;
}
```

Príklad: zmazanie prvku poľa

Program načíta do poľa celé čísla a vymaže číslo zo zadanej pozície

```
#include<stdio.h>

int main(){
   int a[50], i, poz, velkost;
   printf("Zadajte pocet prvkov pola (<=50): ");
   scanf("%d", &velkost);

   printf("Zadajte %d prvkov pola:\n", velkost);
   for(i=0; i<velkost; i++)
      scanf("%d", &a[i]);

   printf("Zadajte poziciu na vymazanie prvku: ");
   scanf("%d",&poz);

   i = poz;
   while(i < velkost-1){
      a[i] = a[i+1];
      i++;
   }

   velkost--;

   printf("Pole po vymazani prvku:\n");
   for(i=0; i<velkost; i++)
      printf("%d ",a[i]);

  return 0;
}
```

Príklad: zmazanie duplicít v poli

Program načíta do poľa celé čísla a vymaže čísla tak, aby sa žiadne v poli neopakovalo

```
#include<stdio.h>

   int main(){
   int a[50], int i, j, k, velkost;
   printf("Zadajte pocet prvkov pola: ");
   scanf("%d", &velkost);
   printf("Zadajte %d pvrkov pola:\n", velkost);
   for(i=0; i<velkost; i++)
   scanf("%d", &a[i]);

   for(i=0; i<velkost; i++){
      j=i+1;
      while(j<velkost){
         if(a[i] == a[j]){
            k=j;
            velkost--;
            while(k < velkost){
               a[k] = a[k+1];
               k++;
            }
         }
         else
            j++;
     }
}
```

```
   printf("Pole po zmanazi duplicit:\n");
   for(i=0; i<velkost; i++)
      printf("%d ",a[i]);
   return 0;
}
```

Príklad: súčet veľkých čísel

Program vypočíta súčet veľkých čísel (zapísaných v poli - uvažujte veľmi dlhé čísla, ktoré sa nezmestia ani do long int (max. 1000 číslic)

```
#include<stdio.h>
#define MAX 50

   int sucet(int cislo1[], int cislo2[], int vysledok[],
      int m, int n);

int main() {
   int i, n, m, p, cislo1[MAX], cislo2[MAX], vysledok[MAX];
   char c;
   printf("Zadajte prve cislo: ");
   m=0;
   while((c=getchar()) != '\n')
      cislo1[m++] = c - '0';

   printf("Zadajte druhe cislo: ");
   n=0;
   while((c=getchar()) != '\n')
      cislo2[n++] = c - '0';

   p = sucet(cislo1, cislo2, vysledok, m, n);
   printf("Sucet je: ");
   for(i=0; i<p; i++)
      printf("%d", vysledok[i]);
   printf("\n");
   return 0;
}
```

```
int sucet(int cislo1[], int cislo2[], int vysledok[],
   int m, int n) {
   int i, j, k, prenos, c1, c2, pom[MAX];

   i = m-1;
   j = n-1;
   k = 0;
   prenos = 0;
   while(i >= 0 || j >= 0) {
      if(i >= 0) c1 = cislo1[i];
      else c1 = 0;
      if(j >= 0) c2 = cislo2[j];
      else c2 = 0;
      pom[k++] =  (c1 + c2 + prenos) % 10;
      prenos = (c1 + c2 + prenos) / 10;
      i--;
      j--;
   }
   if (prenos != 0)
      pom[k++] = prenos;

   for(i=0; i<k; i++)
      vysledok[k-i-1] = pom[i];

   return k;
}
```

Príklad: počet výskytov podpostupnosti

Zistiť počet výskytov podpostupnosti v postupnosti

```
#include <stdio.h>
#define MAX 50

int pocet_vyskytov(int post[], int n, int podpost[], int m)ô

int main() {
   int i, n, m, post[MAX], podpost[MAX];
   printf("Zadajte pocet cisel postupnosti: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
      scanf("%d", &post[i]);

   printf("Zadajte pocet cisel hladanej podpostupnosti: ");
   scanf("%d", &m);
   for(i=0; i<m; i++)
      scanf("%d", &podpost[i]);

   printf("Pocet vyskytov podpostupnosti v postupnosti: %d\n",
      pocet_vyskytov(post, n, podpost, m));
   return 0;
}
```

Príklad: počet výskytov podpostupnosti

```
int pocet_vyskytov(int post[], int n, int podpost[], int m) {
   int i=0, j, vyskyt=0;

   while(i+m < n) {
      for(j=i; j<i+m; j++)
         if(post[j] != podpost[j-i]) break;
      if(j==i+m)
         vyskyt++;
      i++;
   }
   return vyskyt;
}
```

Príklad: počet výskytov podpostupnosti

Program zistí, či načítaná postupnosť
je rastúca, klesajúca, konštantná
alebo aj rastúca aj klesajúca

```
#include <stdio.h>
#define N 5

int main() {
   int i, pole[N];
   int rast = 1, kles = 1, konst = 1;

   printf("Zadajte %d prvkov pola:\n", N);
   for(i=0; i<N; i++)
      scanf("%d", &pole[i]);
   for(i=1; i<N; i++) {
      if(pole[i] != pole[i-1]) konst = 0;
      if(pole[i] <= pole[i-1]) rast = 0;
      if(pole[i] >= pole[i-1]) kles = 0;
   }

   printf("Funkcia je ");
   if(konst) printf("konstantna.\n");
   if(rast) printf("rastuca.\n");
   if(kles) printf("klesajuca.\n");
   if(!konst && !rast && !kles)
      printf("aj rastuca aj klesajuca.\n");

   return 0;
}
```

Ak pole používame ešte na niečo iné.

```c
#include <stdio.h>
#define N 5

int main() {
   int i, akt, pred;
   int rast = 1, kles = 1, konst = 1;

   printf("Zadajte %d prvkov pola:\n", N);
   scanf("%d", &pred);
   for(i=1; i<N; i++) {
      scanf("%d", &akt);
      if(akt != pred) konst = 0;
      if(akt <= pred) rast = 0;
      if(akt >= pred) kles = 0;
      pred = akt;
   }

   printf("Funkcia je ");
   if(konst) printf("konstantna.\n");
   if(rast) printf("rastuca.\n");
   if(kles) printf("klesajuca.\n");
   if(!konst && !rast && !kles)
      printf("aj rastuca aj klesajuca.\n");

   return 0;
}
```

To isté bez použitia poľa.

Príklad: otočenie obsahu poľa

Funkcia otočí reťazec (pole znakov).

```
void reverse(char data[], int size) {
   int i, pom;

   for (i=0; i<size/2; i++) {
      pom = data[i];
      data[i] = data[size-i-1];
      data[size-i-1] = pom;
   }
   printf("%s\n", data);
}
```

Príklad: Eratostenovo sito

- algoritmus na nájdenie prvočísel v poli
  - vyškrtáva všetky násobky prvočísel, počnúc 2 (vyškrtnutie ~ prepísanie čísla na 0)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	3	0	5	0	7	0	0	0	11	0	13	0	0	0

`i:` 2
`prv[i]:` 0

v cykle, **`i`** od 0 do 2 poľa **`prv`**:
  v cykle, **`k`** od **`i+1`** do 14: vyškrtneme všetky násobky čísla **`prv[i]`**

Príklad: Eratostenovo sito

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX 100

void main()
{
   int n, i, k, odm;
   int prv[MAX];

   printf("Do ktoreho cisla hladat prvocisla? (< MAX) ");
   scanf("%d", &n);

   n--;     /* znizime n o 1, nevyskrtavame nasobky jednotky */

   for(i = 0; i < n; i++)  /* inicializacia */
      prv[i] = i+2;
```

⟶ pokračovanie

pokračovanie:

```
   odm = (int) sqrt(n)-1; printf("odmocnina: %d\n", o);

   for(i = 0; i < odm; i++) {
      if(prv[i] != 0) {     /* ak sme predtym cislo nevyskrtli */
         for(k = i+1; k < n; k++) { /* vyskrtni vsetky nasobky */
            if(prv[k] != 0) {     /* ak este nie je vyskrtnute */
               if(prv[k] % prv[i] == 0)     /* ak je delitelne */
                  prv[k] = 0;
            }
         }
      }
   }

   printf("Prvocisla: "); /* vypisanie prvocisel */
   for(i = 0; i < n; i++)
      if(prv[i] != 0)
         printf("%d, ", prv[i]);

   putchar('\n');
   return 0;
}
```

Príklad: kontrola sumy + ladenie

Program zistí, či prvé číslo v súbore **`suma.txt`** je súčtom čísel, ktoré sú za ním. Použitie základného a podrobného ladenia (príklad z cvičení).

```c
#include <stdio.h>
#include <stdlib.h>
#define LADENIE_ZAKLADNE
#define LADENIE_PODROBNE

int main() {
   FILE *fr;
   float cena, suma, sucet = 0.0;

   #if defined(LADENIE_ZAKLADNE) || defined(LADENIE_PODROBNE)
      printf("Otvorenie suboru\n");
   #endif

   if((fr = fopen("suma.txt", "r")) == NULL) {
      printf("Nepodarilo sa otvorit subor.\n");
      exit(1);
   }
   #ifdef LADENIE_PODROBNE
      printf("Subor otvoreny\n");
   #endif
```

```
#if defined(LADENIE_ZAKLADNE) || defined(LADENIE_PODROBNE)
   printf("Kontrola sumy\n");
#endif

fscanf(fr, "%f", &sucet);

while (fscanf(fr, "%f", &cena) != EOF) {
   suma += cena;
   #ifdef LADENIE_PODROBNE
      printf("Suma: %.2f\n", suma);
   #endif
}

if(suma == sucet)
   printf("Suma je spravna\n");
else
   printf("Suma je nespravna\n");

#ifdef LADENIE_PODROBNE
   printf("Suma skontrolovana\n");
#endif
```

```
    #if defined(LADENIE_ZAKLADNE) || defined(LADENIE_PODROBNE)
       printf("Zatvaranie suboru\n");
    #endif

    if(fclose(fr) == EOF) {
       printf("Nepodarilo sa zatvorit subor.\n");
       exit(1);
    }

    #ifdef LADENIE_PODROBNE
       printf("Subor zatvoreny\n");
    #endif

    return 0;
}
```