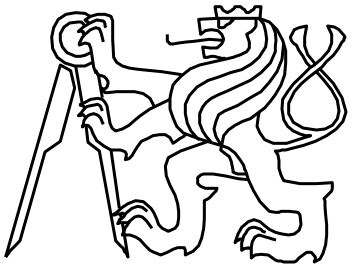
Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.
Praha & EU: Investujeme do vaší budoucnosti

Pole, řetězce



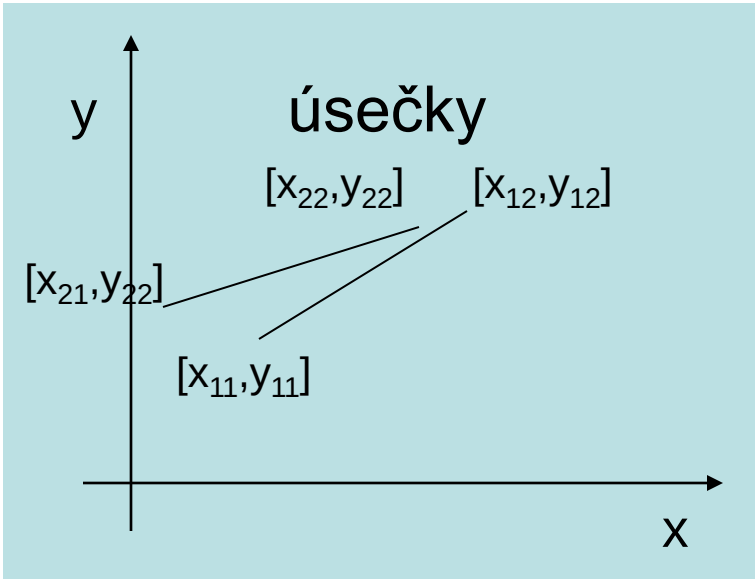
BI-PA1 Programování a algoritmizace 1, ZS 2012-2013
Katedra teoretické informatiky
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

Pole

vektory

15	2	5	4
5	10	2	3
10	-8	1	1

tabulky



- deklarace pole
- označení prvků pole
- pole pro uložení posloupnosti
- pole jako tabulka
- pole jako parametr funkce
- řetězce
- vícerozměrná pole (matice)

Pole

- Příklad: přečíst teploty naměřené v jednotlivých dnech týdnu, vypočítat průměrnou teplotu a pro každý den vypsat odchylku od průměrné teploty
- Řešení s proměnnými typu *int* (prog7-teploty1.c):

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   int t1, t2, t3, t4, t5, t6, t7, prumer;
   printf("zadejte teploty v sedmi dnech v tydnu\n");
   scanf("%d", &t1); scanf("%d", &t2); scanf("%d", &t3); scanf("%d", &t4);
   scanf("%d", &t5); scanf("%d", &t6); scanf("%d", &t7);
   prumer = (t1+t2+t3+t4+t5+t5+t6+t7)/7;
   printf("%d\n",prumer); printf("%d\n",t1-prumer); printf("%d\n",t2-prumer);
   printf("%d\n",t3-prumer); printf("%d\n",t4-prumer);
   printf("%d\n",t5-prumer); printf("%d\n",t6-prumer);
   printf("%d\n",t7-prumer); return 0;
}
```

Teploty

- Řešení je těžkopádné a bylo by ještě horší, kdyby vstupními daty byly teploty za měsíc nebo za celý rok
- Příklad vyřešíme pomocí pole
- Pole je obecně strukturovaný datový typ skládající se z pevného počtu složek (prvků) stejného typu, které se vzájemně rozlišují pomocí indexu
- V jazyku C se pole indexuje celými čísly

  0, 1, ... (počet prvků – 1),

kde počet prvků je dán při vytvoření pole a je neměnitelný.

Pole

- Řešení pomocí pole (prog7-teploty2.c)
  - pro uložení teplot vytvoříme pole obsahující 7 prvků typu *int*

```
int teploty[7]
```

  - první prvek pole má označení *teploty*[0], druhý *teploty*[1] atd.
  - vstupní data přečteme a do prvků pole uložíme cyklem

```
for (i=0; i<7; i++)
        scanf("%d", &teploty[i]);
```

  - průměrnou teplotu vypočteme jako součet prvků pole dělený 7

```
int prumer=0;
for (i=0; i<7; i++)
        prumer += teploty[i];
prumer = prumer/7;
```

  - na závěr pomocí cyklu vypíšeme odchylky od průměru

```
for (i=0; i<7; i++)
printf("%d ", teploty[i]-prumer);
```

Pole v jazyku C

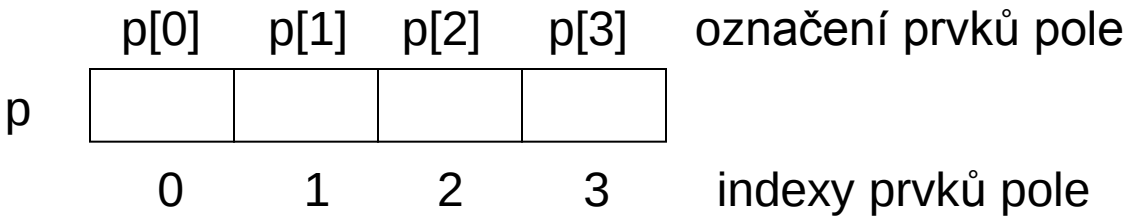
- Pole *p* obsahující *n* prvků typu *T* vytvoříme deklarací
  - T p[n];

  kde *T* může být libovolný typ a *n* musí být celočíselný konstantní výraz s nezápornou hodnotou
- Je-li deklarace pole na úrovni souboru (globální, mimo funkce), pak prvky pole mají nulové počáteční hodnoty
- Je-li deklarace pole lokální ve funkci, pak prvky pole mají nedefinované počáteční hodnoty
- Zápis
  - p[i]

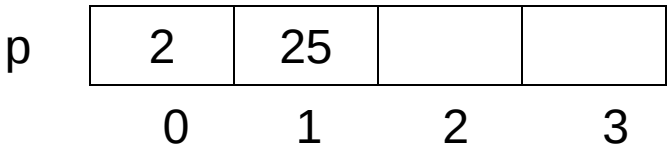
  kde *i* je celočíselný výraz, má vlastnosti proměnné typu *T*
  - jestliže hodnota výrazu *i* je nezáporná a menší než počet prvků, pak *p*[*i*] označuje prvek pole *p* s indexem *i*
  - v opačném případě *p*[*i*] označuje nedefinované místo v paměti (nedovolený index se při běhu programu netestuje!!)

Pole v jazyku C

- Příklad:
  - int p[4];



  - p[0] = 2;
  - p[1] = 25



- Příklad použití špatného indexu: prog7-pole.c

Pole pro uložení posloupnosti

- Příklad teplot zobecníme: z klávesnice zadáme počet dní a pak teploty
- Jak velké pole máme deklarovat?

V jazyku C musí být počet prvků deklarovaného pole dán konstantním výrazem, proto musíme počet dní shora omezit

```
/* prog7-teploty3.c*/
#include <stdio.h>
#include <stdlib.h>
#define MAXPOCET 100
int main(void) {
    int teploty[MAXPOCET], pocet, prumer=0, i;
    printf("zadejte pocet dni (cislo od 1 do %d): ", MAXPOCET);
    scanf("%d", &pocet);
    printf("zadejte teploty ve %d. dnech \n", pocet);
    for (i=0; i<pocet; i++) scanf("%d", &teploty[i]); /* cteni teplot do pole */
    for (i=0; i<pocet; i++) prumer += teploty[i]; /* vypocet prumerne teploty */
    prumer = prumer/pocet;
    printf("prumerna teplota je %d\n", prumer); /* vypis odchylek od prum. teploty */
    printf("odchylky od prumerne teploty:\n");
    for (i=0; i<pocet; i++) printf("%d ", teploty[i]-prumer); printf("\n");
    return 0;
}
```

Teploty ještě jednou

- Program prog7-teploty4.c kontroluje (pomocí funkce), zda zadaný počet dní není větší než maximum

```
#define MAXPOCET 100
int ctiInt(int min, int max) {
    int x; scanf("%d", &x);
    while (x<min || x>max) {
            printf("spatne, zadejte znovu: ");
           scanf("%d",&x);
    }
    return x;
}
int main(void) {
    int teploty[MAXPOCET], pocet, prumer=0, i;
    printf("zadejte pocet dni (cislo od 1 do %d): ", MAXPOCET);
    pocet = ctiInt(1, MAXPOCET);
    printf("zadejte teploty ve %d dnech \n", pocet);
    ...
```

Obracení posloupnosti pomocí pole

- Program, který přečte posloupnost čísel zakončenou nulou a vypíše ji obráceně (tentokrát pomocí pole)

```
/* prog7-obrat.c*/
#include <stdio.h>
#include <stdlib.h>
#define MAXPOCET 100
int main(void) {
    int pole[MAXPOCET], dalsi, i=0;
    printf("zadejte posloupnost nanejvys %d cisel "
      "zakoncenou nulou\n", MAXPOCET);
    do {
      scanf("%d", &dalsi); pole[i] = dalsi;
      i++;
    } while (dalsi!=0 && i<MAXPOCET);
    for (i=i-1; i>=0; i--) printf("%d ", pole[i]);
    printf("\n");
    return 0;
}
```

Pole jako tabulka

- Předchozí příklady ilustrovaly použití pole pro uložení posloupnosti
- Pole lze použít též pro realizaci tabulky (zobrazení), která hodnotám typu indexu (v jazyku C to je pouze interval celých čísel počínaje nulou) přiřazuje hodnoty nějakého typu
- Příklad: přečíst řadu čísel zakončených nulou a vypsat tabulku četnosti čísel od 1do 100 (ostatní čísla ignorovat)
- Tabulka četnosti bude pole 100 prvků typu *int*, počet výskytů čísla *x*, kde $1 \leq x \leq 100$, bude hodnotou prvku s indexem *x*-1
- Tabulku deklarujeme jako globální pole (počáteční hodnoty prvků budou 0) a s pomocí dvou funkcí:
  - *void ctiCisla*() přečte čísla a vytvoří tabulku četnosti
  - *vypisTabulku*() vypíše tabulku četnosti

Pole jako tabulka

```
/*prog7-cetnost1.c*/
int tabulka[100];
void ctiCisla(void) {
    int cislo;
    printf("zadejte posloupnost celych cisel zakoncenou nulou\n");
    scanf("%d", &cislo);
    while (cislo!=0) {
      if (cislo>=1 && cislo<=100) tabulka[cislo-1]++;
      scanf("%d", &cislo);
    }
}
void vypisTabulku(void) {
    int i;
    for (i=1; i<=100; i++)
      if (tabulka[i-1])
         printf("pocet vyskytu cisla %2d je %d\n",i, tabulka[i-1]);
    printf("\n");
}
```

Pole jako tabulka

- Nejmenší číslo je dáno konstantou *MIN*, největší konstantou *MAX* (jména konstant jsou zavedena direktivou *define)*

```
/* prog7-cetnost2.c*/
#include <stdio.h>
#include <stdlib.h>
#define MIN 1
#define MAX 100
int tabulka[MAX-MIN+1];
void ctiCisla(void) {
    int cislo;
    printf("zadejte posloupnost celych cisel zakoncenou nulou\n");
    scanf("%d", &cislo);
    while (cislo!=0) {
      if (cislo>=MIN && cislo<=MAX) tabulka[cislo-MIN]++;
      scanf("%d", &cislo);
    }
}
void vypisTabulku(void) {
    int i;
    for (i=MIN; i<=MAX; i++) if (tabulka[i-MIN]) printf( .....
```

Pole jako parametr funkce

- Program, který přečte dva vektory s *n* složkami (*n* se přečte z klávesnice) a vypíše skalární součin těchto vektorů (prog7-vektory.c)
- Složky vektorů uložíme do polí, jejichž počet prvků zadáme konstantou MAX (zavedeme direktivou *define*). Tím bude omezen počet složek vektorů
- Pro čtení vektoru zavedeme funkci *ctiVektor,* parametrem bude pole, kam mají být uloženy složky vektorů, a počet složek

```
void ctiVektor(int v[], int n) {
    int i;
    printf("zadejte %d celych cisel\n", n);
    for (i=0; i<n; i++) scanf("%d", &v[i]);
}
```

skutečným parametrem bude pole prvků typu *int*

Pole jako parametr funkce

- Pro výpočet skalárního součinu dvou vektorů zavedeme funkci, které dodáme vektory jako parametry, třetím parametrem bude počet složek

```
int skalarniSoucin(int x[], int y[], int n) {
        int i, s=0;
        for (i=0; i<n; i++)
                s += x[i]*y[i];
        return s;
}
```

Pole jako parametr funkce

```
/* prog7-vektory.c*/
...
#define MAX 100
int ctiInt(int min, int max) {...}
void ctiVektor(int v[ ], int n) {...}
int skalarniSoucin(int x[], int y[], int n) {...}
int main(void) {
    int x[MAX], y[MAX], n;
    printf("zadejte pocet slozek vektoru (cislo od 1 do %d): ", MAX);
    n = ctiInt(1, MAX);
    printf("vektor x\n");
    ctiVektor(x, n);
    printf("vektor y\n");
    ctiVektor(y, n);
    printf("skalarni soucin vektoru x a y je %d\n",  skalarniSoucin(x, y, n));
    return 0;
}
```

Pole jako parametr funkce - shrnutí

- Deklarace parametru *p*, kterým má být pole prvků typu *T*, má v hlavičce funkce tvar:
  - *T p*[ ]
- Takto deklarovaný parametr můžeme ve funkci používat jako označení pole prvků typu *T*, např.
  - p[i] = ... ;
- Přípustným skutečným parametrem je jméno pole prvků typu *T*
- Obvykle je třeba předat funkci jako další parametr počet (definovaných) prvků pole (parametrem *p* je funkci předána pouze adresa prvního prvku pole, nic víc!!)
- Zadáme-li ve volání funkce špatný parametr (např. pole prvků jiného typu), překladač vypíše pouze varovné hlášení a funkce bude pracovat špatně (prog7-spatnyparametr.c)
- Ne tak v interpretu PROC!

Řetězce

- Literál typu řetězec je v jazyku C posloupnost znaků uzavřená do uvozovek
- Příklady: "\n" "Pepa z depa" " Kocka sedi na bezu"
- V paměti je řetězec reprezentován posloupností bytů obsahujících kódy znaků řetězce, která je zakončena nulovým bytem
- Proměnné pro uložení řetězců jsou pole prvků typu *char*

```
char txt[10]; /* lze ulozit az 9-znakovy retezec */
```

- Operace s řetězci provádějí funkce deklarované v <*string.h*>
- Pro přiřazení řetězce do pole znaků slouží knihovní funkce *strcpy*(*kam*,*co*):
  - strcpy(txt, “abcd”);

Nekontroluje se, zda pole je dostatečně velké!

- Pro porovnání řetězců slouží knihovní funkce *strcmp*:

strcmp(*x*, *y*) < 0 když *x* < *y*
= 0 když *x* = *y*
> 0 když *x* > *y*

Délku řetězce vrátí funkce *strlen*(*x*)

Řetězce – výpis většího slova I

- Z klávesnice přečteme řetězec pomocí funkce *scanf* a konverze %s (ignoruje počáteční mezery a oddělovače řádků, čtení skončí na mezeře nebo oddělovači řádků)
- Na displej řetězec vypíšeme pomocí funkce *printf* a konverze %s

```
/* prog7-retezce1.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXDELKA 100
int main(void) {
    char slovo1[MAXDELKA], slovo2[MAXDELKA], vetsi[MAXDELKA];
    printf("zadejte prvni slovo: ");
    scanf("%s", slovo1);
    printf("zadejte druhe slovo: ");
    scanf("%s", slovo2);
```

Řetězce – výpis většího slova I

- Z klávesnice přečteme řetězec pomocí funkce *scanf* a konverze %s (ignoruje počáteční mezery a oddělovače řádků, čtení skončí na mezeře nebo oddělovači řádků)
- Na displej řetězec vypíšeme pomocí funkce *printf* a konverze %s

```
/*slova nactena v promennych slovo1 a slovo2*/
printf("slovo %s ma %d znaku\n", slovo1, strlen(slovo1));
printf("slovo %s ma %d znaku\n", slovo2, strlen(slovo2));
if (strcmp(slovo1, slovo2)>0)
        strcpy(vetsi, slovo1);
else
        strcpy(vetsi, slovo2);
printf("vetsi je %s\n", vetsi);
return 0;
}
```

Řetězce – přečtení, otočení a výpis slova

```
/* prog7-retezce2.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXDELKA 100
void obrat(char slovo[]);
int main(void) {
    char slovo[MAXDELKA];
    printf("zadejte slovo: ");
    scanf("%s", slovo);
    obrat(slovo);
    printf("obracene slovo je %s\n", slovo);
    system("PAUSE");
    return 0;
}
```

Řetězce

- Funkce pro obrácení slova v poli

```
void obrat(char slovo[]) {
    int delka = strlen(slovo),
    char pom;
    posledni = delka-1, delka2 = delka/2, i;
    for (i=0; i<delka2; i++) {
            pom = slovo[i];
            slovo[i] = slovo[posledni-i];
            slovo[posledni-i] = pom;
    }
}
```

Vícerozměrné pole

- Vícerozměrným polem se obecně rozumí takové pole, k jehož prvkům se přistupuje pomocí více než jednoho indexu
- V jazyku C se s vícerozměrnými poli pracuje jako s poli, jejichž prvky jsou opět pole
- Příklad dvojrozměrného pole (matice) prvků typu *int*:
  - deklarace dvojrozměrného pole (3 řádky, 4 sloupce) prvků typu *int*

```
int mat[3][4];
```

součet všech prvků pole mat

```
int suma = 0, i, j;
for (i=0; i<3; i++)
    for (j=0; j<4; j++) suma += mat[i][j];
```

Má-li být dvojrozměrné pole parametrem funkce, musí být v deklaraci parametru uveden počet sloupců

- Vše ukážeme na následujícím příkladu

Příklad – součet matic

- Vstupní data:

  m n kde *m* je počet řádků a *n* je počet sloupců matice

  prvky první matice *m* x *n*

  prvky druhé matice *m* x *n*

- Výstup:

  součet matic

- Vzhledem k tomu, že počet řádků a počet sloupců musí být dán konstantním výrazem, omezíme počet řádků konstantou *MAXR* a počet sloupců konstantou *MAXS* (zavedeme je direktivou *define*)
- Řešení: prog7-matice.c

Příklad – součet matic

```
/* prog7-matice.c*/
#include <stdio.h>
#include <stdlib.h>
#define MAXR 10
#define MAXS 10

int ctiInt(int min, int max) { ... } /* precte cislo, zkontroluje, zda je v
                                        intervalu min az max a kdyz ne, cteni se opakuje */

void ctiMatici(int mat[][MAXS], int m, int n) { ... }

void soucetMatic(int x[][MAXS], int y[][MAXS], int z[][MAXS], int m, int
    n) { ... }

void vypisMatice(int mat[][MAXS], int m, int n) { ... }

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

Příklad – součet matic

```
...
int main(void) {
    int x[MAXR][MAXS], y[MAXR][MAXS], z[MAXR][MAXS], m, n;
    printf("zadejte pocet radku matice (max. %d): ", MAXR);
    m = ctiInt(1, MAXR);
    printf("zadejte pocet sloupcu matice (max. %d): ", MAXS);
    n = ctiInt(1, MAXS);
    printf("prvni matice\n");
    ctiMatici(x, m, n);
    printf("druha matice\n");
    ctiMatici(y, m, n);
    soucetMatic(x, y, z, m, n);
    printf("matice souctu\n");
    vypisMatice(z, m, n);
    return 0;
}
```

Příklad – součet matic

- Čtení matice

```
void ctiMatici(int mat[][MAXS], int m, int n) {
    int r, s;
    printf("zadejte %d x %d celych cisel\n", m, n);
    for (r=0; r<m; r++)
            for (s=0; s<n; s++) scanf("%d", &mat[r][s]);
}
```

- Výpis matice

```
void vypisMatice(int mat[][MAXS], int m, int n) {
    int r, s;
    for (r=0; r<m; r++) {
            for (s=0; s<n; s++)
                    printf("%5d ", mat[r][s]);
            printf("\n");
    }
}
```

Příklad – součet matic

```
void soucetMatic(int x[][MAXS], int y[][MAXS], int z[][MAXS], int m, int n) {
    int r, s;
    for (r=0; r<m; r++)
        for (s=0; s<n; s++)
            z[r][s] = x[r][s] + y[r][s];
}
```

Dynamická alokace paměti - malloc

```
void *malloc(unsigned int size)
```

- Příklad alokace proměnné typu double

```
double *p;
if ((p = (double*)malloc(sizeof(double)))==NULL) {
    printf("Nedostatek paměti");
    exit (1);
}
 ...
*p=4.0;
printf("%f", *p);
```

Dynamická alokace paměti – calloc, free

```
/* prog7-alokace.c*/
double *p;
int i;
p = (double *)calloc(10, sizeof(double));

//vynulování pole

for (i=0; i<10; i++) p[i]=0;
```

Totéž pomocí malloc

```
p = (double *)malloc(10 * sizeof(double));
```

- uvolnění paměti

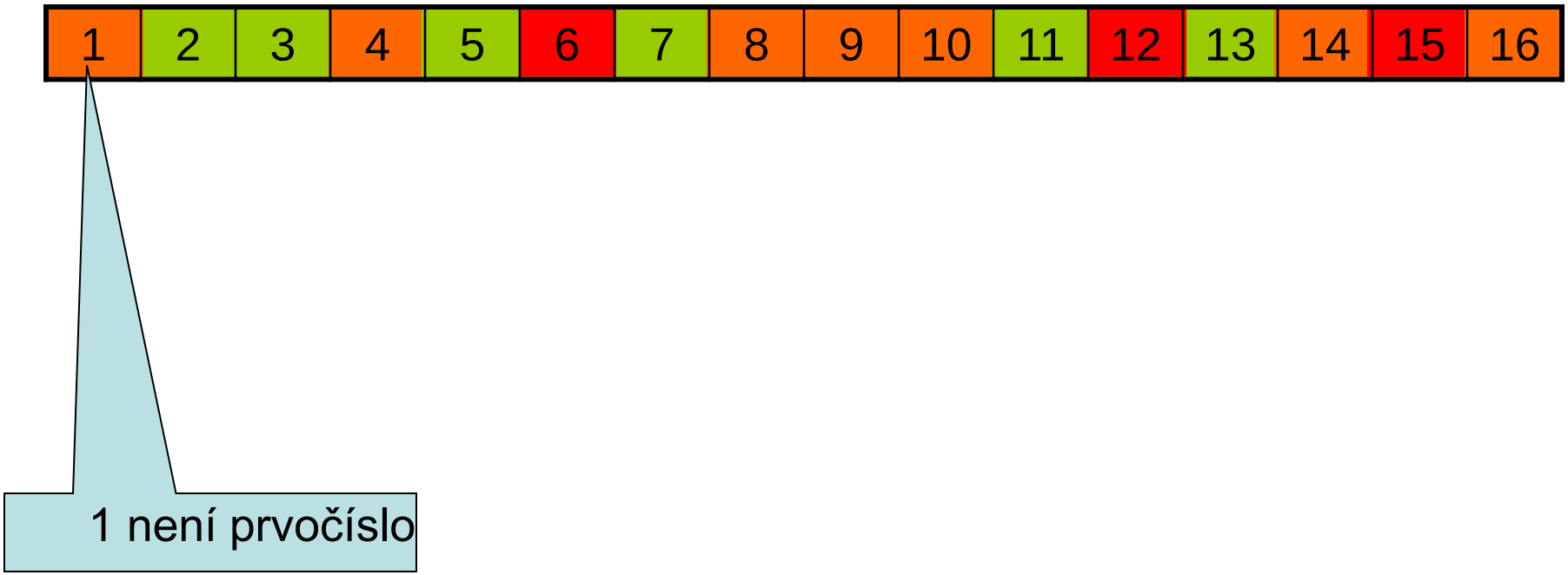
```
void free(void *ptr);
```

Eratosthénovo síto

- Eratosthen z Kyrény, 276–194 př. n. l.

Úkol: Najděte všechna prvočísla menší než zadané N, např. 16

Kandidáti



Eratosthénovo síto

Algoritmus:

1. vytvoř seznam čísel menších než N
2. najdi prvočíslo
3. vyškrtej jeho násobky
4. dokud je co škrtat, pokračuj od bodu 2.

Který je první násobek jež budeme škrtat pro číslo 13?

Je to 2*13?

Kdy budeme škrtat naposledy, když N je 1000000?

Poslední prvočíslo menší než 1000 je 997

Složitost algoritmu - O(N*log(log N))

Eratosthénovo síto

```
/* prog7-poleersito.c*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define boolean char
#define true 1
#define false 0

#define MAX 1000000

void sito(boolean mnozina[], int max) { ... }
void vypis(boolean mnozina[], int max) { ... }
void vypisPocet(boolean mnozina[], int max) { ... }
int main(void) { ... }
```

Eratosthénovo síto

```
void vypisPocet(boolean mnozina[], int max) {
    int i, pocet = 0;
    for (i = 2; i < max; i++)
        if (mnozina[i])pocet++;
    printf("%d\t", pocet);
}

int main(void) {
    boolean mnozina[MAX];
    sito(mnozina, MAX);
    printf("Prvocisla od 2 do %d:", MAX);
    vypisPocet(mnozina, MAX);
    system("PAUSE");
    return 0;
}
```

Eratosthénovo síto

```
void sito(boolean mnozina[], int max) {
    int i, p, pmax;
    for (i = 2; i < max; i++)
        mnozina[i] = true;
    p = 2;
    pmax = (int) sqrt(max);
    do { // vypuštění všech násobků prvočísla p
        for (i = p * p; i < max; i += p)
            mnozina[i] = false;
        do { // hledání nejbližšího prvočísla po p
            p++;
        } while (!mnozina[p]);
    } while (p <= pmax);
return;
}
```

min a max v poli

- Najděte min a max v poli, použijte, co nejméně porovnání.

```
/*prog7-minmax.c*/
#include <stdio.h>
#include <stdlib.h>
void vypis(int mnozina[], int max) {
    int i;
    for (i = 0; i < max; i++) printf("[%2d]=%5d\n", i, mnozina[i]);
}
int main(void) {
    int min, max;
    int p[] = {1, 4, 5, 78, -100, 45, 28, 34, 5};
    vypis(p, 9);
    minMax(p, 9, &min, &max);
    printf("Nejvetsi prvek je %d, \n nejmensi prvek je %d\n", max, min);
    system("PAUSE");
    return 0;
}
```

min a max v poli – I

- Úkol: najděte min a max v poli, použijte, co nejméně porovnání.

```
void minMax(int pole[], int maxIndex, int* min, int *max) {
    int i; *min = *max = pole[0];
    for (i = 1; i < maxIndex; i++) {
        if (pole[i] > *max) *max = pole[i];
        if (pole[i] < *min) *min = pole[i];
    }
    return;
}
```

- Kolik bude potřeba porovnání v nejlepším případě?
- Kolik bude potřeba porovnání v nejhorším případě?
- Co se stane pokud bude maxIndex = 0?

min a max v poli - II

```
void minMax2(int pole[], int maxIndex, int* min, int *max) {
    int i; *min = *max = pole[0];
    for (i = 1; i < maxIndex; i++) {
        if (pole[i] > *max) *max = pole[i];
        else if (pole[i] < *min) *min = pole[i];
    }
    return;
}
```

- Kolik bude potřeba porovnání v nejlepším případě?
- Kolik bude potřeba porovnání v nejhorším případě?
- {1,5,12,123,455}
- {45,25,3,-100}

min a max v poli - III

```
void minMax3(int pole[], int maxIndex, int* min, int *max) {
    int i;
    if (pole[0] > pole[1]) { *max = pole[0]; *min = pole[1]; }
    else { *max = pole[1]; *min = pole[0]; }
    i = 2 - maxIndex % 2;
    for (; i < maxIndex - 1; i+=2) {
        if (pole[i] > pole[i + 1]) {
            if (pole[i]>*max)*max = pole[i];
            if (pole[i + 1]<*min) *min = pole[i + 1]; }
        else {
            if (pole[i + 1]>*max)*max = pole[i + 1];
            if (pole[i]<*min) *min = pole[i];
        }
    }
    return;
}
```

- Kolik bude potřeba porovnání v nejlepším případě?
- Kolik bude potřeba porovnání v nejhorším případě?
- Jak to bude fungovat pro pole liché délky?