Číslo a název šablony	III/2 Inovace a zkvalitnění výuky prostřednictvím ICT
Číslo didaktického materiálu	EU-OPVK-VT-III/2-ŠR-214
Druh didaktického materiálu	DUM
Autor	RNDr. Václava Šrůtková
Jazyk	čeština
Téma sady didaktických materiálů	Programování v C# v příkladech II
Téma didaktického materiálu	Třídění pole
Vyučovací předmět	Seminář z informatiky
Cílová skupina (ročník)	Žáci ve věku 17–18 let
Úroveň žáků	Mírně pokročilí
Časový rozsah	1–2 vyučovací hodiny
Klíčová slova	Třídění, třídící klíč, probublávání, přímý výběr
Anotace	Studenti chápou problém třídění a učí se programovat algoritmy přímého výběru a probublávání
Použité zdroje	DRÓZD, Januš a Rudolf KRYL. *Začínáme s programováním*. 1.vyd. Praha: Grada, 1992, 306 s. ISBN 80-854-2441-X. LIBICHER, Ivan a Pavel TÖPFER. *Od problému k algoritmu a programu: sbírka řešených úloh z programování*. 1. vyd. Praha: Grada, 1992, 119 s. Educa '99. ISBN 80-854-2482-7. TÖPFER, Pavel. *Algoritmy a programovací techniky*. 1. vyd. Praha: Prometheus, 1995, 299 s. ISBN 80-858-4983-6. TÖPFEROVÁ, Dana a Pavel TÖPFER. *Sbírka úloh z programování*. Vyd. 1. Praha: Grada, 1992, 98 s. Educa '99. ISBN 80-854-2499-1. VYSTAVĚL, Radek. *Moderní programování: sbírka úloh k učebnici pro středně pokročilé*. 1. vyd. Ondřejov: moderníProgramování, 2008-2009, 2 sv. ISBN 978-80-903951-3-8. VYSTAVĚL, Radek. *Moderní programování: sbírka úloh k učebnici pro začátečníky*. 2. vyd. Ondřejov: moderníProgramování, 2008, 2 sv. ISBN 978-80-903951-5-2. VYSTAVĚL, Radek. *Moderní programování: učebnice pro středně pokročilé*. Ondřejov: moderníProgramování s.r.o, 2008. ISBN 978-80-903951-2-1. VYSTAVĚL, Radek. *Moderní programování:*

	učebnice pro začátečníky. Ondřejov: moderníProgramování s.r.o, 2007, 2 sv. ISBN 978-80-903951-0-7.
Typy k metodickému postupu učitele, doporučené výukové metody, způsob hodnocení, typy k individualizované výuce apod.	Text je možno využít ke společné práci, samostatné přípravě studentů, domácímu studiu apod. Při společné práci je vhodné nejprve obtížnější úlohy rozebrat, potom společně se studenty implementovat na počítači. (Rozbor nejlépe na tabuli, synchronní řešení s promítáním) V pracovním listu je zadání cvičení – většinou se jedná o úlohy, které by měli studenti naprogramovat samostatně. Není nutné, aby všichni zpracovali všechno, vhodné je diferencovat podle jejich zájmu a schopností. Obtížnější úlohy jsou označeny hvězdičkou. Součástí materiálu je zdrojový kód těchto příkladů. Návrh způsobu hodnocení: ohodnocení samostatné práce během hodiny např. podle volby a počtu úloh a elaborace řešení (efektivnost, komentáře…).

Metodický list k didaktickému materiálu

Prohlášení autora

Tento materiál je originálním autorským dílem. K vytvoření tohoto didaktického materiálu nebyly použity žádné externí zdroje s výjimkou zdrojů citovaných v metodickém listu.

Obrázky (schémata a snímky obrazovek) pocházejí od autora.

214. Třídění

aneb jak ukládat data, aby k nim byl snadný přístup

Problém: je dáno pole A[1]..A[n] nějakých prvků (v praxi tvorby informačních systémů nejčastěji záznamů databází)

Chceme, aby po proběhnutí třídícího algoritmu platilo:

A[1] <= A[2] <=… A[n]

Poznámka: pokud se jedná o záznamy, které se skládají z několika položek, třídíme podle jedné z nich – **třídící klíč**.

Při programování algoritmů, které se budou často používat, je důležitá jejich časová **efektivnost,** která se měří počtem porovnatelných operací při jejich realizaci. (Provádění časově efektivnějšího algoritmu na počítači by mělo trvat kratší dobu) Při třídění porovnáváme obvykle počet operací srovnání.

Pro začátek si uvedeme dva algoritmy, které sice nejsou nejefektivnější, ale relativně snadno se programují.

Na formulář umístíme dvě víceřádková textová pole, v jednom se zobrazí naše nesetříděné pole, do druhého pak budeme zobrazovat pole přerovnané.

Přímý výběr

Nejprve najdeme minimum, vyměníme ho s prvním prvkem a pak pokračujeme dál:

zabýváme se vždy úsekem pole od **i** do **n**, vyhledáme v něm minimum a vyměníme ho s i-tým prvkem.

Protože pro umístění každého prvku potřebujeme řádově n operací srovnání, celkový počet operací srovnání bude řádově n*n.

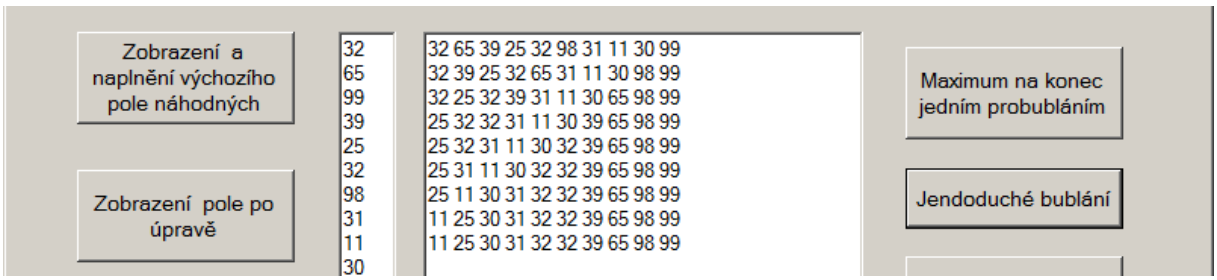
(Rychlost při spuštění na počítači vyplývá z toho, že údaje pro for-cykly se připravují do cache procesoru, čímž se zpracování zrychlí)

Třídění probubláváním

Procházíme pole po dvojicích a je-li větší prvek před menším, (inverze) vyměníme je. Po prvním průchodu se dostane největší prvek nakonec. Po druhém průchodu můžeme podobně zařadit na předposlední místo druhý největší prvek atd. Počet operací srovnání je opět řádově n*n

Vylepšení: Při každém průchodu evidujeme, zda jsme vyměňovali a algoritmus ukončíme, když se nevyměňovalo. (setříděné pole)

Algoritmus si můžeme kázat na výpisu programu



Příklad 1.

Naprogramujte první průchod polem s vyměňováním sousedních prvků, jsou-li v inverzi.

```
private void buttonMaxKonec_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < 9; i++)
            {
                if (a[i] > a[i + 1]) //pokud je menší prvek před větším, vyměním je
                {
                    int pom = a[i]; a[i] = a[i + 1]; a[i + 1] = pom;
                }
            }
```

```
    }
```

Příklad 2.

Naprogramujte setřídění pole tak, že minulý příklad rozšíříte o opakovaný průchod pole – počet opakování je dán počtem prvků.

```
private void buttonJednoducheBubl_Click(object sender, EventArgs e)
{
   for (int j=0;j<9;j++)
       //vnější cyklus udává, kolikáté maximum se dostane na své místo
    for (int i = 0; i < 9; i++)
    {
        if (a[i] > a[i + 1]) //pokud je menší prvek před větším, vyměním je
        {
            int pom = a[i]; a[i] = a[i + 1]; a[i + 1] = pom;
        }

    }
}
```

Příklad 3.

Naprogramujte vylepšení – pole budeme procházet jen, pokud nebude setříděno, pak algoritmus skončí.

```
private void buttonChytrBubl_Click(object sender, EventArgs e)
{
    bool menil;
    do //bubláni budeme opakovat, dokud pole nebude setříděné
     // setříděné pole se pozná tak, že při jeho průchodem nebylo třeba
     // vyměňovat
    {
        menil=false;//logická proměnná, pokud se nic nevyměňovalo, bude false
        for (int i = 0; i < 9; i++)
         {
          if (a[i] > a[i + 1]) //pokud je menší prvek před větším, vyměním je
           {
            int pom = a[i]; a[i] = a[i + 1]; a[i + 1] = pom; menil=true;
           }
         }
    }
     while (menil); //totéž jako menil=true;
    }
```

Důležité

Časová efektivnost algoritmu se měří počtem porovnatelných operací při jeho realizaci.

Jednoduché třídící algoritmy jako je přímý výběr nebo probublávání mají kvadratickou složitost – tedy úměrnou n*n, kde n je počet tříděných prvků.

Pracovní list

Cvičení

1. Naplňte pole reálnými čísly, (`3.14, 5.7, 0, -2.3, 5.6`) zobrazte ho, setřiďte a opět zobrazte.

2. Nalezněte v poli celých čísel nejmenší prvek a vyměňte ho s prvním prvkem. Upravené pole opět zobrazte.

3. Doplňte do třídění probubláváním výpis pole po každém průchodu.

4.(*) Naprogramujte třídění pole celých čísel přímým výběrem, obě pole zobrazte.

Řešení

1.

```
private void buttonRealna_Click(object sender, EventArgs e)
        {
            double[] b = new double[5] {3.14, 5.7, 0, -2.3, 5.6}; ;
            for (int i = 0; i < 5; i++) //zobrazení
                textBoxVych.Text+=b[i].ToString()+Environment.NewLine;
            bool menil;//třídění
            do
            {
                menil=false;//logická proměnná, pokud se nic nevyměňovalo, bude false
                for (int i = 0; i < 4; i++)
                 {
                  if (b[i] > b[i + 1]) //pokud je menší prvek před větším, vyměním je
                    {
                     double pom = b[i]; b[i] = b[i + 1]; b[i + 1] = pom; menil=true;
                    }
                 }
            }
             while (menil); //totéž jako menil=true;

         for (int i = 0; i < 5; i++)
                textBoxVyst.Text+=b[i].ToString()+Environment.NewLine;
        }
```

2.

```
private void buttonMinZac_Click(object sender, EventArgs e)
```

```
        {
           int imin = 0; //index minima – pro výměnu potřebujeme vědět, kde se v poli
       nachází
            //a[imin] je potom ono minimum, pro začátek a[0]
            for (int i = 0; i < 10; i++)
                if (a[i] < a[imin]) imin = i;
            //minimum vyměníme s 1 prvkem
            int pom = a[imin]; a[imin] = a[0]; a[0] = pom;
        }
```

3.

```
private void buttonJednoducheBubl_Click(object sender, EventArgs e)
        {
            for (int j = 0; j < 9; j++)
            {
                //vnější cyklus udává, kolikáté maximum se dostane na své místo
                for (int i = 0; i < 9; i++)

                    if (a[i] > a[i + 1])//pokud je menší prvek před větším, vyměním je
                    {
                        int pom = a[i]; a[i] = a[i + 1]; a[i + 1] = pom;
                    }
                //po každém probublávacím cyklu přidáme výpis řádku

                for (int k = 0; k < 10; k++)
                    textBoxVyst.Text += a[k].ToString() + " ";

                textBoxVyst.Text += Environment.NewLine; //a následné odřádkování
            }
```

4.

```
private void buttonPrVyb_Click(object sender, EventArgs e)
        {


            for (int j = 0; j < 10; j++)
            {
                //hledáme minimum  úseku posloupnosti od a[j] po a[9] a toto
                //minimum vyměňujeme vždy s prvkem a[j]
                int imin = j; //index minima – pro výměnu potřebujeme vědět, kde se v
poli nachází
                for (int i = j; i < 10; i++)
                    if (a[i] < a[imin]) imin = i;
                //minimum vyměníme s j prvkem
                int pom = a[imin]; a[imin] = a[j]; a[j] = pom;
            }
        }
```