вання **обчислювального мислення**, яке включає в себе розв'язання задач, проектування систем та розуміння людської поведінки, виходячи з базових понять інформатики, а безмашинні вправи вчать розпізнавати відповідні задачі у повсякденному житті. Отже, ресурси проекту Code.Org можна використовувати у процесі вивчення алгоритмізації та програмування на початковому етапі, коли важливо не стільки опанувати певну мову програмування, скільки зрозуміти її основи і сформувати інтерес до більш спеціалізованих занять у майбутньому.

★ ★ ★

Pasichnyk O. Computational thinking in the computer science lessons

Annotation. The paper examines the notion of computational thinking as a school computer science education component. According to the new curricula, algorithms and programming are taught from the second grade. As content and methods of teaching vary for different age groups, the article presents a range of materials, which can be used to develop computational thinking for the students of 2nd-9th grades.

Keywords: computational thinking, middle school programming, school computer science, Code.Org.

★ ★ ★

Пасичник О. В. Вычислительное мышление на уроках информатики

Аннотация. В статье рассматривается понятие вычислительного мышления (computational thinking) в качестве составляющей школьного курса информатики. Согласно новым программам, разделы, посвященные алгоритмизации и программированию, изучаются, начиная со второго класса. Поскольку содержание и методы обучения различаются для разных возрастных групп, в статье описано материалы, которые можно использовать для формирования вычислительного мышления у учащихся 2–9 классов.

Ключевые слова: вычислительное мышление, программирование в средней школе, информатика в школе, Code.Org.

Література

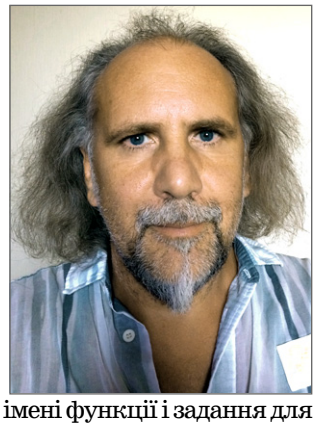
1. Computational Thinking, Jeannette M. Wing — Communications of the ACM, March 2006 [Електронний ресурс]. — Режим доступу: http://www.cs.cmu.edu/~CompThink/papers/Wing06.pdf.
2. Exploring Computational Thinking [Електронний ресурс]. — Режим доступу: http://www.google.com/edu/computational-thinking/what-is-ct.html.
3. We Can Code It! Tasneem Raja [Електронний ресурс]. — Режим доступу: http://goo.gl/E2V6j3.
4. Навчальні програми з інформатики Великобританії [Електронний ресурс]. — Режим доступу: http://www.computingatschool.org.uk/.
5. Навчальні програми з інформатики США [Електронний ресурс]. — Режим доступу: http://csta.acm.org/Curriculum/sub/K12Standards.html.
6. Навчальні програми з інформатики України [Електронний ресурс]. — Режим доступу: http://www.mon.gov.ua/ua/often-requested/educational-programs/.
7. Проект Code.Org [Електронний ресурс]. — Режим доступу: http://code.org/.
8. Відео-звернення про проект Година коду [Електронний ресурс]. — Режим доступу: http://goo.gl/6hspdN.
9. Навчальні матеріали проекту Code.Org [Електронний ресурс]. — Режим доступу: http://learn.code.org/ ; на момент написання статті доступні бета-версії нових курсів http://learn.code.org/beta.

★ ★ ★

НАВЧАННЯ ОСНОВНИХ ІДЕЙ ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ МОВИ OBJECT PASCAL

Горошко Юрій Васильович,

завідувач кафедри інформатики і ОТ Чернігівського національного педагогічного університету імені Т. Г. Шевченка, доктор педагогічних наук, доцент.



Парадигма програмування — це сукупність ідей і понять, через які визначається стиль написання програми. Парадигма, у першу чергу, визначається базовою програмною одиницею. У сучасній індустрії програмування дуже часто парадигма програмування визначається набором інструментів програміста, а точніше, мовою програмування і бібліотеками, що використовуються.

Інколи парадигму програмування ще називають стилем програмування, хоча термін «стиль» щодо програмування може мати й інший сенс. У даній статті під терміном «стиль» матимемо на увазі якраз парадигму програмування.

Відомо кілька основних парадигм програмування, найважливішими з яких на даний момент є парадигми імперативного, функціонального, декларативного, об'єктно-орієнтованого програмування.

Зауважимо, що, мабуть, найбільш незнайомою для студентів педагогічних університетів є парадигма функціонального програмування.

У функціональному програмуванні наголошується на застосуванні функцій, на відміну від імперативного програмування, у якому наголошується на змінах у стані й виконанні послідовностей команд. Іншими словами, функціональне програмування є способом створення програм, у яких єдиною дією є виклик функції, єдиним способом поділу програми є створення нового імені функції і задання для цього імені виразу, за яким обчислюється значення функції, а єдиним правилом композиції є оператор суперпозиції функцій. Жодних комірок пам'яті, операторів надання значень, циклів, чи, тим більше, блок-схем чи передавання управління [1].

Користь в оволодінні функціональною парадигмою програмування полягає в тому, що під час написання програми у такому стилі, потрібно розбити програму на невеликі частини, які є простими й надійними у той же час.

В основу функціонального програмування покладено дві ідеї:

1. Дані повинні бути незмінними, тобто під час написання програми у функціональному стилі не змінюють дані, як це роблять в імперативному програмуванні, а на основі одних незмінних даних конструюють інші, з іншими значеннями, наприклад, якщо щось потрібно змінити в масиві, створюють новий масив, з даними, які відрізняються від даних, що є у початковому масиві.

2. Програма, написана у функціональному стилі, не повинна враховувати своє минуле, тобто не містить відомостей про те, що могло або не могло відбутися у програмі раніше. Кожну структурну одиницю такої програми (функцію) треба сприймати так, що вона не має відомостей про все інше у програмі, крім інших функцій, тобто функція не повинна мати стану. Кожна функція оперує тільки тими даними, які передані їй як параметри. Іншими словами, кожна функція програми не повинна мати побічного ефекту.

Оволодіння таким стилем опису функцій є корисним і під час написання програм з імперативною парадигмою. Тому навчання студентів такого опису функцій є дуже важливим.

Існують мови програмування, які повністю базуються на функціональній парадигмі, наприклад мова Haskell, але освоїти основи функціонального програмування можна і з використанням, наприклад, мов об'єктно-орієнтованого програмування, таких як Object Pascal. У даній статті якраз і буде йти мова про написання програм у функціональному стилі з використанням мови Object Pascal.

Також характерною особливістю функціональної парадигми є відмова від циклів. Оскільки у функціональній парадигмі всі дії реалізовано через виклик функцій, то і циклічні дії організовано через рекурсивний виклик функцій.

Відзначимо також, що надалі ми матимемо справу з так званими динамічними масивами у мові Object Pascal. Описуючи такі масиви, не вказують діапазон для значень індекса. Пам'ять під динамічний масив розподіляється після того, як вказано його розмір за допомогою процедури SetLength(масив, розмір), а елементи динамічного масиву номеруються, починаючи з нуля. Для роботи з динамічними масивами визначено низку функцій, наприклад:

- Length(масив), за якою повертається кількість елементів масива;
- Copy(масив, початковий_індекс, кількість_елементів), за якою повертається масив, що складається з елементів початкового масиву за вказаними початковим індексом і кількістю елементів;
- High(масив), за якою повертається найбільше значення для індекса масиву.

Особливості роботи з динамічними масивами розглянуто на такому прикладі:

```
var a,b: array of integer;
begin
 setlength(a,5);
 a[0]:=0;
 a[1]:=1;
 a[2]:=2;
 a[3]:=3;
 a[4]:=4;
 b:=copy(a,1,3);
 writeln(b[0]);
 writeln(b[1]);
 writeln(b[2]);
 writeln(Length(b));
...
```

За наведеним прикладом отримаємо:

```
1
2
3
3
```

Розглянемо тепер задачу: необхідно знайти середнє арифметичне елементів масиву цілих чисел.

Якщо розв'язувати цю задачу з використанням імперативної парадигми, можемо отримати такий код:

```
type DynIntArr=array of integer;
var a: DynIntArr;
Sum, i: Integer;
begin
 SetLength(a,5);
 a[0]:=0;
 a[1]:=1;
 a[2]:=2;
 a[3]:=3;
 a[4]:=4;
 Sum:=0;
 for i:=0 to Length(a)−1 do
   Sum:=Sum+a[i];
 writeln(Sum/Length(a));
end.
```

Розв'яжемо тепер цю задачу з використанням функціональної парадигми. Оскільки, як уже наголошено вище, у функціональній парадигмі не використовуються цикли, спочатку опишемо рекурсивну функцію, за якою можна знайти суму елементів масиву. Ця функція матиме два параметри: часткове значення суми елементів певної частини масиву, яку вже розглянуто, й частина масиву, яку розглянути залишилось. Створюючи код цієї функції, будемо використовувати такий підхід: додамо до часткового значення суми значення початкового елемента залишку масиву, знайдемо новий залишок масиву без початкового елемента, якщо цей залишок непорожній, то рекурсивно викличемо цю функцію з новим значенням часткової суми і новим залишком масиву, у протилежному випадку — виходимо з рекурсії, повернувши часткове значення суми як результат функції. Отримаємо такий код.

```
function SumOfArray(PartSum:integer;
a: DynIntArr):Integer;
var b:DynIntArr;
begin
 PartSum:=PartSum+a[0];
 b:=Copy(a,1,Length(a)−1);
 if (Length(b)>0) then
```

```
  SumOfArray:=SumOfarray(PartSum,b)
 else
  SumOfArray:=PartSum;
end;
```

Для загальності функціонального підходу можна замінити оператор додавання **PartSum:=PartSum+a[0]** функцією додавання двох цілих чисел, яку описано нижче:

```
function AddInt(a,b: Integer):integer;
begin
 AddInt:=a+b;
end;
```

Отримаємо такий код:

```
function SumOfArray(PartSum:integer;
a: DynIntArr):Integer;
var b:DynIntArr;
begin
 PartSum:=AddInt(PartSum,a[0]);
 b:=Copy(a,1,Length(a)−1);
 if (Length(b)>0) then
  SumOfArray:=SumOfarray(PartSum,b)
 else
  SumOfArray:=PartSum;
end;
```

Залишилося описати функцію знаходження середнього арифметичного елементів масиву:

```
function AveragefArray(a: DynIntArr): Real;
begin

AverageOfArray:=SumOfArray(0,a)/Length(a);
end;
```

Тепер наведемо всю програму:

```
program FuncDemo;
{$APPTYPE CONSOLE}
uses
 SysUtils;
type DynIntArr=array of integer;
function AddInt(a,b: Integer): integer;
begin
 AddInt:=a+b;
end;
function AveragefArray(a: DynIntArr): Real;
begin

AverageOfArray:=SumOfArray(0,a)/Length(a);
end;
function SumOfArray(PartSum:integer;
a:DynIntArr):Integer;
var b:DynIntArr;
begin
 PartSum:=AddInt(PartSum,a[0]);
 b:=Copy(a,1,Length(a)−1);
 if (Length(b)>0) then
  SumOfArray:=SumOfarray(PartSum,b)
 else
  SumOfArray:=PartSum;
end;
var a: DynIntArr;
begin
 setlength(a,5);
 a[0]:=0;
 a[1]:=1;
 a[2]:=2;
 a[3]:=3;
 a[4]:=4;
 writeln(AverageOfArray(a));
 readln;
end.
```

Далі доцільно розглянути функцію, за якою кожний елемент масиву змінюється, наприклад збільшується на якусь константу, написавши її з врахуванням особливостей функціональної парадигми, а саме незмінність даних і відмову від циклів.

Для описання такої функції спочатку опишемо додаткову функцію *f*, за якою, власне, і здійснюється рекурсивне збільшення елементів початкового масиву *a* на вказану константу і з розміщенням результату у масиві *b*:

```
function f(a,b:DynIntArr; i: Integer):DynIntArr;
var c:DynIntArr;
begin
 Setlength(b, AddInt(Length(b), 1));
 b[High(b)]:=AddInt(a[0], i);
 c:=Copy(a,1,Length(a)−1);
 if (Length(c)>0) then
  f:=f(c,b,i)
 else
  f:=b;
end;
```

Потім опишемо кінцеву функцію IncArr для приховування використання допоміжного масиву:

```
function IncArr(a:DynIntArr;
Increment: Integer): DynIntarr;
var b: DynIntArr;
begin
 IncArr:=f(a,b,Increment);
end;
```

У головній програмі, щоб збільшити всі елементи масиву a, наприклад на 5, розмістивши результат у масиві b, достатньо застосувати такий оператор:

```
b:=IncArr(a,5);
```

Розгляд наведених вище прикладів дозволить викладачеві не тільки ознайомити студентів з важливими ідеями функціональної парадигми, але збагатить їхні навички у використанні рекурсії, буде служити пропедевтикою щодо вивчення мови ПРОЛОГ у дисципліні «Основи штучного інтелекту», оскільки у цій мові також для організації циклів використовують рекурсію.

Література

1. Функціональне програмування. — Режим доступу : uk.wikipedia.org/wiki/Функціональне_програмування — 5.09.2012 р.
2. *Jonathon Morgan*. Don't Be Scared Of Functional Programming. — Режим доступу : http://www.smashingmagazine.com/2014/07/02/dont-be-scared-of-functional-programming/ — 5.09.2012 р.
3. *Горошко Ю.В*. Система інформаційного моделювання у підготовці майбутніх учителів математики та інформатики : дис. на здобуття наук. ступ. доктора пед. наук за спец. 13.00.02 «Теорія та методика навчання (інформатика)» / Ю. В. Горошко. — К., 2013.