ШЛЯХИ ФОРМУВАННЯ АЛГОРИТМІЧНИХ УМІНЬ І НАВИЧОК УЧНІВ

Семенова О.І.

Хоча в шкільному курсі інформатики алгоритмічній підготовці учнів приділяється достатньо уваги, більшості школярів важко опанувати алгоритмічні вміння. Одна з причин утруднень пов'язана з початком вивчення основ алгоритмізації. Ефективність алгоритмічної підготовки учнів підвищується, якщо вона починається в молодшому шкільному віці й безперервна протягом усього процесу навчання. Іншою причиною є недостатнє використання різних методичних засобів організації алгоритмічної діяльності на уроках інформатики.

Часто в основній школі єдиним методичним засобом запису й реалізації алгоритму є мова програмування. Використання процедурної мови програмування високого рівня, безумовно, сприяє посиленню фундаментального компонента шкільного курсу інформатики, дає можливість підготувати учнів до подальшої навчальної й професійної діяльності, дозволяє перевірити на практиці виконання конкретного алгоритму. Однак для досягнення за оптимальний час установлених загальноосвітнім стандартом вимог до рівня підготовки школяра в галузі алгоритмізації необхідно в повному обсязі задіяти методичний арсенал. Доцільно використовувати такі дидактичні засоби, як навчальні виконавці, що діють в «обстановці», детально пророблять всі ланки технологічного ланцюжка «структурна схема — шкільна алгоритмічна мова — мова програмування».

Використання структурних схем (блок-схем), представлених стандартно, дозволить полегшити розуміння учнями структури алгоритму, сконцентрувати їхню увагу на знаходженні способу розв'язання завдання, а не тільки на синтаксисі мови, здійснити швидку перевірку правильності розробленого алгоритму на рівні ідеї, розібрати більше число навчальних завдань. Шкільна алгоритмічна мова (російськомовний псевдокод), який застосовується для складання структурного запису з певним ступенем формалізації природною мовою, спрощує переклад алгоритму з мови блок-схем на мову процедурного програмування.

Самостійна й усвідомлена алгоритмічна діяльність учнів можлива тоді, коли учні не переписують кимсь розроблений алгоритм, а створюють його самостійно, не боячись помилитися. Реалізовувати алгоритм на комп'ютері доцільно після того, як учитель перевірить його запис у вигляді структурної схеми (блок-схеми). Це скоротить час безпосередньої роботи учня за комп'ютером, дозволить учителеві досить швидко визначити правильність алгоритму без обліку синтаксису мови програмування, дасть можливість приділити більше уваги роботі з відстаючими учнями.

Учителеві необхідно мати додаткові завдання для диференціації навчання, ретельно продумувати дослідницьку діяльність учнів. Із цією метою вчитель може:

- внести зміни в умову завдання, які не приводять до істотної перебудови структури (наприклад, замість суми чисел знайти їхній добуток);
- накласти обмеження на використання певних засобів, ресурсів (наприклад, не використовувати в умові логічні операції).

Далі для організації алгоритмічної діяльності можна запропонувати учням такі завдання:

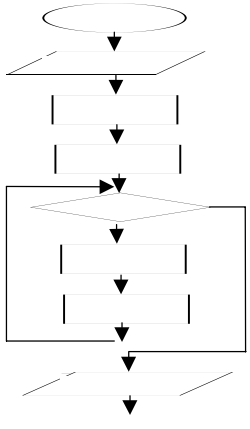
- скласти алгоритм за блок-схемою;
- визначити базові структури, що входять в алгоритм, і принцип їхнього з'єднання;
- знайти контрприклад, що вказує на логічну або семантичну помилку в алгоритмі;
- указати область правильної роботи алгоритму;
- знайти відповідність між фрагментами блок-схеми, псевдокоду й програми;
- визначити призначення фрагмента алгоритму;
- знайти результат для конкретних вхідних даних;
- оформити частину алгоритму як допоміжний, виконати параметризацію алгоритму;
- змінити алгоритм, замінивши деякі алгоритмічні конструкції (наприклад, замінити цикл «поки» циклом «до»);
- спробувати розв'язати завдання іншим способом (наприклад, більш коротким або більш простим);
- організувати в алгоритмі перевірку допустимості значень вхідних даних (наприклад, не розглядати негативні числа);
- виконати подальшу покрокову деталізацію алгоритму;
- доповнити алгоритм раніше розробленими блоками;
- звести алгоритм до єдиного блоку;
- відновити пропущені блоки алгоритму, рядка програми;
- записати коментарі в програмі у вигляді пояснень про хід виконання програми;
- створити користувацький інтерфейс програми після одержання результату розв'язання завдання (наприклад, інтерпретувати результати);
- досліджувати результати внесення змін у текст програми (наприклад, проекспериментувати із командою виведення).

Структурний підхід до розробки алгоритмів визначає необхідність оформлення алгоритмічних структур так, щоб переклад запису алгоритму з однієї мови на іншу виключав внесення змін у конструкцію алгоритмічних приписів. Наприклад, якщо в блок-схемі алгоритму використовується цикл із передумовою, то серія команд повинна повторюватися у разі дотримання умови. Такий підхід не виключає можливість одночасної заміни базових конструкцій у всіх зазначених способах запису алгоритму. У деяких випадках цикл «поки» (команду повторення з передумовою) можна замінити циклом «до» (командою повторення з післяумовою) або циклом «для» (командою повторення з параметром). Од-

нак для розв'язання конкретного завдання в ланцюжку «структурна схема — навчальна алгоритмічна мова — мова програмування» повинні використовуватися такі ж конструкції, щоб перехід від одного запису дій до іншого був технологічним процесом. Для ілюстрації зазначеного підходу розглянемо алгоритм знаходження суми перших натуральних непарних чисел до n. Представимо запис алгоритму трьома способами: у вигляді блок-схеми, шкільної алгоритмічної мови й мовою програмування Паскаль (табл. 1).

Блок-схема складається з таких базових структур: двох складених команд (команди слідування й команди повторення з передумовою) і простої команди. Усі команди з'єднані послідовно. Конструкції оформлені стандартно, тому їх легко розпізнати й перевести на мову програмування. Кожна конструкція має один вхід і один вихід. Це дозволяє виконувати як наступну деталізацію алгоритму, так і його перетворення убік укрупнення блоків.

Таблиця 1

Блок-схема	Школьный АЯ		Паскаль
	алг сумма нечетных чисел	→	Program sum1;
		→	Var n, S, I: integer;
	поч	→	Begin
	уведення n	→	Readln (n);
	S:=0	→	S=0;
	i:= 1	→	i=1;
	поки i<=n	→	While i<= n do
	пц	→	Begin
	S:=S+I	→	S:= S+I;
	I:= i+2	→	I:= i+2;
	кц	→	End;
	вивід S	→	Writeln ('S=',S);
	кін	→	End.

Якщо запропонувати блок-схему з циклом «до», у якому тіло циклу виконується у випадку недотримання умови (інвертувавши умову на $i > n$ і помінявши місцями «так» і «ні»), то під час перекладу алгоритму на шкільну алгоритмічну мову й мову програмування буде потрібно внести зміни, інакше це не буде відповідати зазначеному підходу.

Пунктирні стрілки в таблиці відбивають послідовність виконання технологічного ланцюжка. Після запису алгоритму у вигляді блок-схеми кожна команда переводиться на шкільну алгоритмічну мову, а вже потім на мову програмування.

Важливо із самого початку формувати в учнів структурний підхід до побудови алгоритмів і технологічний підхід до розробки програм. Надалі (наприклад, у старшій школі) це дозволить легко перейти до іншої процедурної або об'єктно-орієнтованої мови програмування.

Модифікуємо умову попереднього завдання. Запишемо алгоритм обчислення суми перших n натуральних чисел. У всіх ланках ланцюжка поміняємо цикл «поки» на цикл «для» і наведемо приклад перекладу алгоритму з мови блок-схем на мову програмування Паскаль (табл. 2).

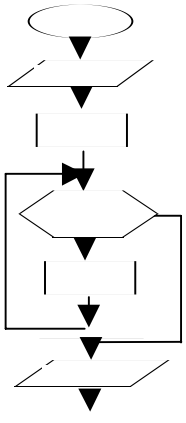
Дидактична спіраль побудови шкільного курсу інформатики визначає оволодіння знаннями, і вміннями учнів у контексті, що ускладнюється, за допомогою збагачення, розвитку й узагальнення досліджуваних понять. Під час вивчення алгоритмічної лінії курсу інформатики поняття «алгоритм» нерозривно пов'язане з поняттям «виконавець». Можна виділити такі рівні складності розгляду даних понять. Для «алгоритму» — лінійний, розгалужений, циклічний. Для «виконавця» — я сама, інша людина, комп'ютер. Алгоритмічну підготовку учнів доцільно здійснювати з урахуванням підвищення складності алгоритмічних конструкцій і ступеня відчуження алгоритму учнями.

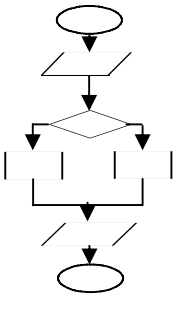
Більш глибоке розуміння учнями процесу виконання алгоритму досягається під час використання ручного тестування алгоритму, коли в ролі формального виконавця виступає не комп'ютер, а сам учень. Представити динамічну інформаційну модель покрокового виконання алгоритму можна двома способами: трасуванням алгоритму за допомогою таблиці й колонок.

Перевірку виконання команд проходження й розгалуження доцільно оформити у вигляді таблиці, рядки якої відповідають командам алгоритму, а стовпці містять значення величин, що зберігаються в пам'яті виконавця, результати перевірки умов і очікувану учнями інформацію на екрані комп'ютера. Заповнимо трасувальну таблицю (рис. 1), що ставиться до алгоритму знаходження більшого з двох чисел, представленого у вигляді блок-схеми (див. рис. 1).

Кожна команда блок-схеми виконується учнями для конкретних вхідних даних, результат її виконання відбивається в трасувальній таблиці. Для запобігання надмірності інформації не слід дублювати значення рядків у таблиці. Якщо комірка пам'яті набуває нового значення, то його варто записати у відповідному рядку таблиці, а попереднє значення закреслити для демонстрації процесу зникнення інформації.

Таблиця 2

Блок-схема	Шкільний АЯ		Паскаль
	**алг сума чисел**	→	Program sum1;
		→	Var i, n, S:integer;
	поч	→	Begin
	уведення n	→	Readln(n);
	S:=0	→	S:=0;
	для i **від** 1 **до** n	→	For i:=1 to n do
	пц	→	S:=S+i;
	S:=S+i	→	
	кц	→	
	вивід S	→	Writeln(S);
	кін	→	End.



Вид экрана	Ячейки памяти			Проверка условия
	a	b	max	
5,8	5	8		
				5 > 8 (нет)
			8	
8				

Рис. 1

Доцільно запропонувати учням розробити інший алгоритм для рішення цього ж завдання з використанням команди розгалуження в неповній формі. З огляду на дане обмеження, в алгоритм можна ввести наступні зміни. Після уведення двох змінних *a* й *b*, змінній max присвоїти значення першої змінної *a*. Якщо значення змінної max менше значення *b*, то змінній max присвоїти значення другої змінної *b*. Команду виведення максимального значення залишити без змін.

Тестування команди повторення можна відобразити у вигляді колонок, число яких буде відповідати кількості повторень циклу. Це дасть можливість уникнути багаторазового виправлення результатів покрокового виконання тіла циклу. Дана форма подання процесу розглядається після трасувальної таблиці, оскільки, з одного боку, передбачається вивчати команду повторення після команди проходження й розгалуження, а з іншого боку, запис у вигляді колонок менш очевидний, ніж у вигляді таблиці.

На рис. 2 показано, як заповнити трасувальні колонки до алгоритму знаходження остачі від ділення націло двох натуральних чисел.

Учням можна запропонувати запитання на розуміння й додаткове завдання: яке значення містить змінна *b* після виконання програми? Яку команду треба додати, щоб на екрані комп'ютера побачити значення змінної *b* після виконання програми? Змінити алгоритм таким чином, щоб крім остачі від ділення націло була знайдена частка від ділення націло (тобто підрахувати кількість повторень циклу).

Таким чином, ручне тестування алгоритму дозволяє учневі виступити в ролі виконавця алгоритму, продемонструвати зміну даних у комірках пам'яті комп'ютера й виконання команд залежно від поставлених умов. За допомогою трасувальних таблиць і колонок можна виконати ручне тестування алгоритму, представленого різними способами, у тому числі й на мовах програмування різних рівнів.

Підбір тестів для перевірки правильності розробки алгоритму, верифікація й інтерпретація результатів формального виконання алгоритму є важливими етапами підготовки й розв'язання завдань із використанням комп'ютера. Однак на уроках інформатики або зовсім не реалізуються зазначені етапи й наявність працюючої програми для учня означає досягнення результату, або вчитель бере на себе завдання аналізу отриманих результатів, у найкращому разі залучаючи учнів до пошуку семантичних і логічних помилок у програмі. Такі підходи не розкривають до кінця сутність комп'ютерного експерименту. Учням необхідно самостійно продумувати різні варіанти вихідних даних і перевіряти кінцеві результати роботи уведеної програми. Для цього на стадії розробки алгоритму, тобто до набору учнями тексту програми за комп'ютером, варто визначи-

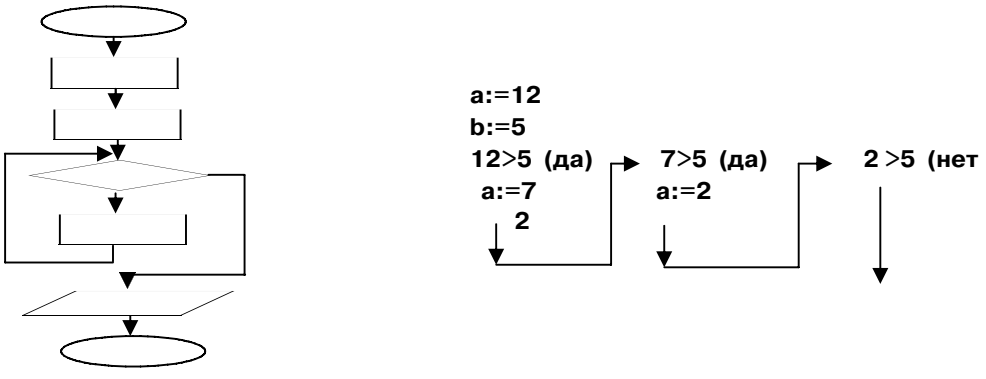


Рис. 2

ти перевірювані події і відповідні їм результати виконання програми. Використання системи тестів під час проведення комп'ютерного експерименту дозволить учням усвідомлено й цілеспрямовано проводити тестування й налагодження програми.

Під час складання системи тестів необхідно передбачити перевірку основних окремих випадків розв'язання конкретного завдання й основних типів даних, у тому числі й неприпустимих. Систему тестів доцільно оформити у вигляді таблиці із вказівкою номера тесту, вхідних і вихідних даних.

Розглянемо алгоритм обчислення периметра трикутника, у якому передбачається відстежити введення некоректних даних. Блок-схема одного з можливих алгоритмів розв'язання даного завдання (рис. 3) містить дві команди слідування, три команди розгалуження в скороченій формі, з'єднані за допомогою вкладення, і команду розгалуження в повній формі.

Додаткова змінна (ознака) до дозволяє уникнути багаторазового дублювання в алгоритмі однієї й тієї ж команди виведення повідомлення на екран монітора: «Трикутник не існує». Латинськими буквами A, B, C, D на блок-схемі відзначені можливі варіанти проходження по орієнтованому графу. Оскільки кожний варіант виконання алгоритму необхідно перевірити хоча б одним тестом, то система тестів буде містити як мінімум чотири випадки, що перевіряються (табл. 4).

Варіанту A відповідають тести № 1, 2 і 3, варіанту B — № 4, варіанту C — № 5, варіанту D — № 6. Як завдання можна запропонувати учням доповнити систему тестів, для кожного випадку, що перевіряються, підібрати свої вхідні й вихідні дані.

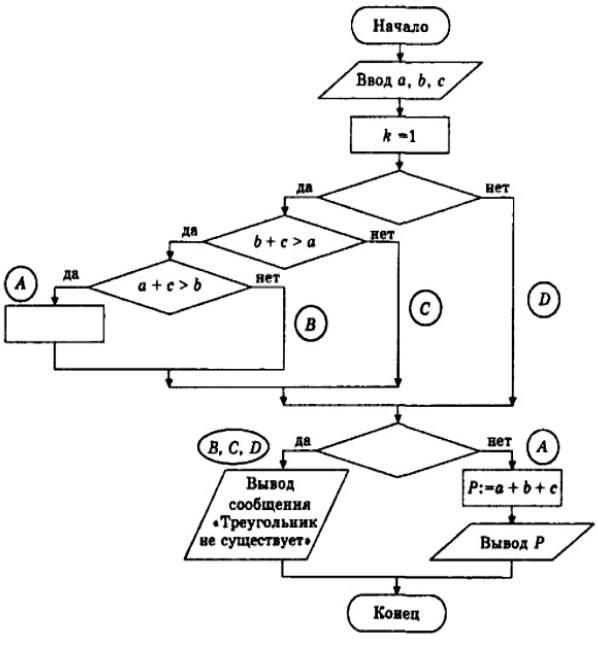


Рис. 3

Таким чином, на підставі вищесказаного можна стверджувати, що використання вчителем блок-схем з урахуванням структурного підходу до розробки алгоритмів, ручного тестування алгоритму за допомогою трасувальних таблиць або колонок, системи тестів під час проведення комп'ютерного експерименту буде сприяти більш ефективному формуванню алгоритмічних умінь і елементів алгоритмічної культури школярів. Вивчення алгоритмізації за запропонованою технологією дає можливість раннього виявлення обдарованих дітей.

Таблиця 4

№	Випадок, що перевіряється	Дані			Результат
		a	b	c	
1	Рівносторонній трикутник	3	3	3	9
2	Рівнобедрений трикутник	3	3	5	11
3	Прямокутний трикутник	3	4	5	12
4	Неприпустимі значення	1	3	-1	Трикутник не існує
5	Трикутник вирожденний	2	0	0	Трикутник не існує
6	Трикутник вирожденний	3	3	6	Трикутник не існує