УДК 004.415

А.М. Ковальчук, к.т.н.
В.Г. Левицький, к.т.н., доц.
І.І. Самолюк, аспір.
В.М. Янчук, к.т.н.
Житомирський державний технологічний університет

ПРОЕКТУВАННЯ ПІДСИСТЕМИ ВІДЛАГОДЖЕННЯ ТА ВІЗУАЛІЗАЦІЇ ДЛЯ ПРОГРАМНОЇ СИСТЕМИ АВТОМАТИЗОВАНОЇ ПОБУДОВИ КОМПІЛЯТОРІВ

Дана робота присвячена підтримці функцій візуалізації та відлагодження для компілятора компіляторів FancyCC 3.0. Детально викладено вимоги до підсистеми візуалізації, запропоновані способи кодування основних типів даних алгоритму та встановлення зв'язку між структурами даних програми користувача та відповідного візуального представлення. Наведено приклади можливої програмної реалізації підсистеми відлагодження та візуалізації.

Постановка проблеми. Створення сучасних великих програмних систем призводить до того, що використання вже розроблених середовищ генераторів компіляторів не завжди відповідає вимогам архітектури таких програмних засобів. Тому при розробці лінгвістичного забезпечення для програмної системи DSR Open Lab 1.0 [1]–[3] виникла необхідність створення нового інструментального засобу для автоматизованої побудови трансляторів, що дозволив би задовольнити специфічні вимоги одночасного використання багатьох предметно-орієнтованих мов у рамках одного програмного комплексу, забезпечив би зручний і нескладний графічний інтерфейс користувача та простоту способу описання предметно-орієнтованої мови користувача. Для цієї мети було розроблено програмне середовище генерації мовних процесорів FancyCC 3.0 [1], [4–6], яке призначене для генерації компіляторів різноманітних предметно-орієнтованих мов та мов програмування, в тому числі й мов описання математичних структур.

Дана робота присвячена вирішенню однієї з важливих проблем, які необхідно проаналізувати та вирішити на шляху вдосконалення програмного середовища генерації мовних процесорів, а саме: підтримці функцій візуалізації та відлагодження для компілятора компіляторів FancyCC 3.0.

Мета і задачі роботи. Метою роботи є проектування підсистеми візуалізації та налагодження для програмного середовища генерації мовних процесорів. Така підсистема має вирішувати наступні задачі:

- візуальне представлення процесу генерації компілятора, а саме: відображення роботи алгоритмів створення окремих функціональних частин компілятора;
- документування створеного транслятора;
- управління процесом роботи створеного компілятора (покрокове відображення етапів розбору, миттєве внесення змін в роботу транслятора тощо).

Наукова новизна роботи. Запропоновано новий для предметної області автоматизованої генерації мовних процесорів підхід до проектування підсистеми візуалізації та відлагодження.

Аналіз досліджень і публікацій. Дана робота є логічним продовженням досліджень в галузі розробки підсистеми візуалізації та відлагодження програмної системи автоматизованої побудови компіляторів FancyCC 3.0, які проводились колективом авторів раніше [1], [7]. На відміну від попередніх досліджень, які були сконцентровані на питаннях ефективного документування створеного транслятора, дана робота присвячена проектуванню більш загального підходу до візуалізації процесу створення та функціонування компілятора.

На сьогоднішній день створено значну кількість програмних систем, які не тільки реалізують функції автоматизованої розробки трансляторів, але й підтримують візуалізацію алгоритмів генерації компіляторів та ілюструють роботу окремих частин створеного компілятора. Серед цих інструментів існує ряд таких, що пов'язані з проектуванням і реалізацією сценаріїв візуалізації без використання допоміжних програмних засобів побудови анімації: LLparse та LRparse [8], Gyacc (*graphical yacc*) [9]. Вони виділяються ретельним пророблення деталей візуалізації кожної фази трансляції використовуючи анімацію для пояснень різноманітних концепцій процесу компіляції. При цьому візуальне представлення кожної фази компіляції розробляється індивідуально, а за необхідності доповнення програмних систем новими алгоритмами розширення анімаційних властивостей може відбуватися лише за рахунок попереднього накопичення бібліотек стандартних функцій, оскільки для згаданих програм не існує предметно-орієнтованих інструментів, що дозволяють автоматично синтезувати підсистеми анімації (чи їх складові).

На відміну від програм LLparse, LRparse та Gyacc, роботи зі створення ілюстрованого компілятора [10], [11] не лише запроваджують візуалізацію етапів компіляції ad hoc (для даної конкретної мети), але й висувають загальні вимоги до середовища ілюстрації програм, яке б підтримувало інструменти створення необхідних візуальних представлень. Таку тенденцію підтримує робота [12], присвячена описанню пакета програм для анімації алгоритмів синтаксичного аналізу LL(1) і SLR(1) [13]. При цьому важливим кроком є

використання авторами допоміжного інструменту XTANGO (X-window Transition-based Animation Generation) [14], [15], добре відомої системи для розробки програм з анімацією для ОС UNIX. Застосування цієї платформи для конструювання анімації алгоритмів вносить в розробку риси відкритості та гнучкості.

Таким чином, аналіз існуючих систем візуалізації та відлагодження для програм автоматизованої побудови компіляторів виявляє придатність та ефективність попередньої *автоматизації* процесу візуалізації. При цьому одне лише накопичення бібліотек стандартних алгоритмів, що часто використовуються в задачах цієї предметної області, є недостатнім засобом розробки відкритих програмних систем, придатних до швидкої модифікації. Тому необхідним є використання предметно-орієнтованого інструмента, що дозволить автоматично будувати анімаційні фрагменти.

За класифікацією, запропонованою в [19], системи візуального програмування поділяються на класи візуальних середовищ і візуальних мов. Потреби програмної системи FancyCC 3.0 визначають належність її підсистеми відлагодження та візуалізації до першого підкласу першого класу систематизації, а саме до середовищ візуалізації даних та інформації про дані. Деякі риси проектованої підсистеми відлагодження та візуалізації, які вказують її місце в систематиці [15], наведено в табл. 1.

Таблиця 1

Визначення рис проектованої підсистеми відлагодження та візуалізації

Пункт класифікації	**Значення пункту класифікації**
1	2
Обмеження візуалізації	
Загальність	візуалізація довільних програм в рамках визначеного далі класу (на відміну від генерації вузького набору алгоритмів або існуючих програм)
Апаратне забезпечення	обмежується операційною системою
Операційна система	родина ОС MS Windows
Мови програмування	імперативні та об’єктно-орієнтовані
Паралелізм	в розумінні застосування до мов з паралелізмом — не підтримується
Обмеження візуалізації програм	візуалізації довільних програм для мов, які підтримуються
Спеціалізація	відсутня
Підтримка великих елементів	обмежена системними ресурсами
Предмет візуалізації	
Візуалізація програмної системи	програмні дані та потік даних
Час збирання даних	дані збираються як під час компіляції, так і під час виконання програми користувача
Як співвідносяться програмний час та час візуалізації	динамічний збір інформації та динамічна візуалізація
Час генерації візуалізації	як наперед заданий хід візуалізації (стиль “post mortem”), так і інтерактивна побудова візуалізації
Форма візуалізації	
Середовище	програмна система
Кольори	обмежені системними ресурсами
Розмірність	2D

Продовження таблиці 1

Пункт класифікації	**Значення риси підсистеми**
1	**2**
Використання анімаційних технологій	рівень “стерти”/“перемалювати”
Адаптоване до користувача представлення великих масивів даних	підтримується
Декілька вікон візуалізації	підтримується
Синхронізація вікон візуалізації	підтримується
Метод візуалізації	
Стиль специфікації візуалізації	автоматизація візуалізації за рахунок динамічного зв’язку між даними програми та підсистемою візуалізації
Риси штучного інтелекту	не підтримується
Здатність настроювати візуалізацію	підтримується
Зв’язування візуалізації та програмної системи	використання сервісів віртуальної машини підсистеми візуалізації та відповідних типів даних в програмному коді системи користувача
Ступінь спареності між візуалізацією та програмним кодом	використання сервісів віртуальної машини підсистеми візуалізації та відповідних типів даних тісно інтегровані з програмним кодом системи користувача
Взаємодія з користувачем	
Стиль	базується на стандартних елементах управління графічного інтерфейсу користувача операційної системи
Навігація	базується на стандартних елементах управління графічного інтерфейсу користувача операційної системи

Керування швидкістю візуалізації	підтримка покрокової візуалізації та призупинення виконання програми користувача
Візуалізація в протилежному напрямку часу	якщо користувач реалізує алгоритми самостійно
Вбудовані функції запису демонстрацій	не підтримується

Проведений огляд сучасних інструментів візуалізації програмного забезпечення [15]–[18] демонструє велику кількість претендентів на роль засобу автоматизації побудови підсистеми візуалізації та відлагодження для FancyCC 3.0. На жаль, той факт, що існуючі безкоштовні програмні інструменти не задовольняють наведеним в табл. 1 рисам проектованої підсистеми, а також наявність специфічних вимог до згаданого компілятора компіляторів [1], [7] як частини великого комплексу програмних засобів чисельного аналізу та математичного моделювання, не дозволяють використати в даному випадку існуючі предметно-орієнтовані інструменти автоматичної побудови анімації. Тому наступні розділи даної роботи присвячені проектуванню деяких важливих рис нової подібної програмної системи (системи візуалізації алгоритмів VIAL), яка підтримувала б автоматизовану розробку анімації даних і відповідних алгоритмів полегшуючи таким чином побудову підсистеми візуалізації та відлагодження для FancyCC 3.0.

Загальні риси системи візуалізації алгоритмів. Далі розглянемо вимоги до функціонального навантаження системи візуалізації алгоритмів, не обмежуючись предметною областю побудови мовних процесорів. В цьому випадку серед різних властивостей алгоритмів нас будуть цікавити лише здатність алгоритму працювати з деяким, заздалегідь визначеним набором структур даних, змінюючи дані в ході роботи. В подальшому ми відволікаємось від будь-яких інших рис алгоритмів в рамках їх роботи в програмній системі і вважаємо візуалізацією алгоритму відображення поточного стану відповідних структур даних в різні моменти часу: перед початком роботи алгоритму, після закінчення або після важливих внутрішніх етапів його роботи.

Головними проблемами, що постають перед розробкою системою візуалізації алгоритмів у цьому випадку є:

- встановлення зв’язку між структурами даних, якими маніпулює алгоритм, та їх візуальним представленням;
- визначення способу та стилю відображення структур даних у візуальному вигляді;
- визначення способу розміщення візуального представлення структур даних відносно один одного та відносно вже існуючої для користувача візуальної картини.

Встановлення зв’язку між структурами даних, якими маніпулює алгоритм, та їх візуальним представленням. Дана проблема зводиться до забезпечення системи візуалізації алгоритмів можливістю перегляду та модифікації зовнішніх значень програмного коду алгоритму. Складність цієї проблеми викликана орієнтацією системи візуалізації алгоритмів не стільки на інтерпретуючі реалізації мов програмування, скільки на взаємодію з алгоритмами, які попередньо відкомпільовані в машинний код.

В загальному випадку для користувачів, зацікавлених у візуальному представленні алгоритму, представляють цікавість, звичайно, далеко не всі змінні його програмної реалізації, а лише такі, що відповідають структурам даних, важливих для сприйняття та пізнання особливостей роботи алгоритму. Задача визначення набору таких змінних повністю повинна бути покладена на розробника (або кодувальника) алгоритму. Завдання ж системи візуалізації алгоритмів в цьому випадку є забезпечення уніфікованого способу доступу до змінних різного типу під час виконання програмного коду алгоритму. Складність цієї задачі полягає, по-перше, в забезпеченні підтримки необмеженого набору конкретних (таких, що мають унікальне ім’я в програмному коді) типів змінних, які беруть участь у візуалізації, а, по-друге, в необхідності розробки такої надбудови над системою підтримки часу виконання (run-time), яка дозволила б за допомогою попередньо визначеного функціонального відношення зв’язати бітові образи змінних в пам’яті комп’ютера з їх візуальним представленням. Згадане функціональне відношення, очевидно, має бути побудоване під час компіляції програмного коду алгоритму.

Реалізація підтримки необмеженого набору конкретних типів змінних в рамках системи візуалізації алгоритмів має базуватися на розумінні таких фактів:

- набір конкретних скалярних типів мов програмування є обмеженим;
- набір найменувань складених типів мов програмування є обмеженим, а побудова кожного конкретного складеного типу є процедурою структурування множини конкретних скалярних і складених типів за правилами, визначеними найменуванням даного конкретного складеного типу;
- набір найбільш поширених в алгоритмах структур даних є обмеженим і включає такі типи, як число, рядок, дерево, стек, вектор, матриця тощо.

Очевидно в розглядуваному випадку можливо реалізувати обмежений опис нескінченого числа конкретних типів змінних, призначених для візуалізації. Прикладом такого опису є сам компілятор відповідної мови програмування, однак для задачі створення системи візуалізації алгоритмів такий спосіб описання типів структур даних є занадто загальним і потужним. Альтернативним придатним описом змінних в цьому випадку повинно виступити функціональне відношення між бітовими образами

визначеної структури та їх візуальним представленням. Способом ітеративної побудови такого функціонального відношення є множина послідовних викликів функцій системи візуалізації алгоритмів, які декларують типи змінних (в рамках термінології, заздалегідь узгодженої між віртуальною машиною системи та програмною реалізацією алгоритму і вказують на ділянку пам'яті, де зберігається значення змінної.

Визначення способу та стилю відображення структур даних у візуальному вигляді. Оскільки набір структур даних, поширених в цікавих для користувача алгоритмах, є обмеженим, можливим є попереднє створення способу візуального відображення кожної структури даних і методу застосування схеми візуального відображення до змінних конкретних типів. Базовими типами даних, які обробляються віртуальною машиною системи візуалізації алгоритмів будемо вважати такі структури: логічне значення; ціле число; дійсне число; комплексне число; рядок; вектор; матриця; багатовимірний масив; множина; стек; список; черга; дерево; хеш.

Візуальне представлення елементів означеної множини даних залежить від множини зареєстрованих в системі шаблонів візуалізації. Шаблони візуалізації можуть формуватись, наприклад, у вигляді текстового або бінарного опису, що пов'язує конкретний тип даних його візуальним представленням. Для кожної структури даних має бути зареєстровано мінімум один вид візуального представлення. Також декілька різних типів даних можуть бути пов'язані з одним видом візуального представлення. Приклади декількох видів візуального представлення для одного типу даних представлені у табл. 2.

Таблиця 2

Приклади візуального представлення даних

Тип даних	Візуальне представлення
Матриця	Математичний стиль
	Табличний стиль
Дерево	Стиль "MS Windows Explorer"
	Стиль теорії графів
Стек	Стилі в залежності від напрямку зростання стеку

Окремим цікавим випадком, який зумовлює стиль відображення змінних алгоритму, є складені типи даних, елементами яких є інші складені типи даних. Так, наприклад, стиль відображення стеку цілих чисел суттєво відрізняється від стеку дерев. В першому випадку можливим є компактне візуальне представлення в області малого розміру. В другому випадку неможливо заздалегідь стверджувати, що в загальному випадку детальне відображення стеку буде досить компактним. Навпаки, в цьому випадку слід очікувати швидкого зростання площі візуалізації, необхідної для відображення складеної структури даних. Тому можливим є символічне зведення другого типу відображення до першого шляхом символічного представлення вкладених до стеку структур даних (у нашому випадку дерева) у вигляді тексту, піктограми тощо. Для повноцінної візуалізації складених структур даних в такому випадку слід реалізувати також спосіб візуального перегляду вкладених до стеку дерев як додаткового тимчасового або постійного зображення.

Визначення способу розміщення візуального представлення структур даних. Загалом проблема включає в себе спосіб вирішення питання про ефективний тип відображення набору структур даних для алгоритму. Окремими задачами тут є визначення критерію оптимальності відображення набору структур даних, методу прийняття рішення, визначення впливу фактичного набору структур даних на індивідуальне відображення кожного елементу з набору (наприклад яку площу слід відвести під дану змінну, в якому вікні або в якій частині вікна її відображати, які зображення слід зробити напівпрозорими та, можливо, сумістити їх тощо).

Для поточної реалізації підсистеми візуалізації алгоритмів пропонується вважати, що кожна структура даних відображається в окремій області (можливо, заданого розміру). При цьому необхідною частиною підсистеми візуалізації є методи поєднання окремих (компактних) областей для структур даних алгоритму в єдине зображення, яке і буде вважатися візуальним представленням алгоритму.

Короткий перелік інтерфейсів системи візуалізації алгоритмів. За функціональним спрямуванням можна виділити такі інтерфейси (групи функцій) системи візуалізації алгоритмів.

1. Клієнтський інтерфейс алгоритму:
   - встановити адресу і тип змінної, яка буде відображатися;
   - встановити тип візуального відображення (лише для перегляду або перегляду та редагування з подальшою перевіркою коректності);
   - інформувати систему про початок роботи алгоритму;
   - інформувати систему про зміну алгоритмом значення змінної;
   - інформувати систему про призупинення алгоритму;
   - інформувати систему про повне зупинення алгоритму;
   - інформувати систему про статус закінчення роботи алгоритму (нормальне чи помилкове).
2. Інтерфейс з боку системи управління візуалізацією алгоритмів:
   - отримати список усіх візуалізованих змінних для даного алгоритму;

- освіжити візуальне представлення змінних алгоритму за їх поточним значенням;
- режим анімації (постійне оновлення візуального представлення змінних алгоритму через встановлений проміжок часу);
- інформувати алгоритм про можливість початку роботи;
- інформувати алгоритм про можливість продовження роботи;
- інформувати алгоритм про необхідність паузи;
- інформувати алгоритм про необхідність повного зупинення.

3. Внутрішній інтерфейс підтримки паралельного виконання кількох алгоритмів (на спільних або окремих структурах даних).

4. Внутрішній інтерфейс відображення структур даних:

- відобразити або редагувати структуру даних за типом змінної, типом її розміщення (наприклад: додаткове вікно, частина вікна, зображення поверх вікна) тощо.

5. Внутрішній інтерфейс підтримки "асистента користувача":

- моніторинг роботи користувача;
- імітування роботи користувача;
- прийняття рішення та поради щодо визначення способу розміщення візуального представлення структур даних відносно один одного та відносно вже існуючої для користувача візуальної картини.

Можливий спосіб програмного кодування типів структур даних алгоритмів користувача. Розглянемо далі стиль кодування змінних, визначених у фрагментах програмного коду C++, для використання в програмній підсистемі відлагодження та візуалізації. Наступний приклад пояснює загальні принципи структурування інформації щодо типів структур даних та конкретних екземплярів змінних відповідних типів. Отже, на рис. 1 зображено декларування змінних користувача, які потрібно візуалізувати; на рис. 2 наведений фрагмент протоколу передачі типів структур даних та конкретних екземплярів змінних системі візуалізації алгоритмів; на рис. 3 зображено результуючий програмний код, який забезпечує відображення задекларованих раніше змінних у визначеному користувачем стилі.

```
...
struct LRTableCharActionItem {
       unsigned short value;
       char* action;
};
class LRActionT : public LRTableData {
...
       long rows, columns;
       LRTableActionItem *Table;
...
};
LRActionT LALR_action_table;
...
```

Рис. 1. Декларація змінних користувача

```
...
// Binary coding of supported types
struct VIAL_Type {
  string name;                 // (mandatory): type of data structure,
                               // one of the following fixed set:
                               // {"scalar", "struct", "vector", "table", "tree"}
  string alias;                // (mandatory): type unique alias
                               // (uniquely defines drawing form)
  struct VIAL_DataStyle style; // (optional): style of data representation
  struct VIAL_Type *sub_type;  // (optional): array of sub-types
  long sub_type_number;        // (optional): number of sub-types number
};

// Binary coding of the data structure
struct VIAL_Data {
  string var;                 // data (variable) name: optional
  string caption;             // data caption: optional
  void* address;              // data address: mandatory
  string type_alias;          // (mandatory): type unique alias
  struct VIAL_Data *sub_data; // (optional): array of sub-data
  long sub_data_number;       // (optional): number of sub-data number
};
...
```

Рис. 2. Фрагмент протоколу передання типів структур даних та конкретних екземплярів змінних системі візуалізації алгоритмів

```
...
int VIALAPI_DeclareType(string alg_name, struct VIAL_Type*);  // declares a new type
int VIALAPI_DeclareData(string alg_name, struct VIAL_Data*);  // declares a new data structure

struct VIAL_Type tp_VIAL_unsigned_short = {"scalar", "unsigned short"};
struct VIAL_Type tp_VIAL_char_ptr = {"scalar", "char*"};
struct VIAL_Type tp_VIAL_LRTableCharActionItem_SubTypes[ 2 ] = {
  tp_VIAL_unsigned_short,
  tp_VIAL_char_ptr
};

struct VIAL_Type tp_VIAL_LRTableCharActionItem;
tp_VIAL_LRTableCharActionItem.name  = "struct";          // predefined type of data structure
tp_VIAL_LRTableCharActionItem.alias = "LRTableCharActionItem";
tp_VIAL_LRTableCharActionItem.style.default_style = 1;   // default table style
tp_VIAL_LRTableCharActionItem.sub_type_number = 2;       // always 1 for "table" type
tp_VIAL_LRTableCharActionItem.sub_type =
               (struct VIAL_Type*)tp_VIAL_LRTableCharActionItem_SubTypes;

struct VIAL_Type tp_VIAL_LRTableActionItem;
tp_VIAL_LRTableActionItem.name  = "table";          // predefined type of data structure
tp_VIAL_LRTableActionItem.alias = "LRTableActionItem*";
tp_VIAL_LRTableActionItem.style.default_style = 1;  // default table style
tp_VIAL_LRTableActionItem.sub_type_number = 1;      // always 1 for "table" type
tp_VIAL_LRTableActionItem.sub_type = &tp_VIAL_LRTableCharActionItem;

VIALAPI_DeclareType( "LALR(1) parsing", &tp_VIAL_LRTableActionItem );

struct VIAL_Data dt_LALR_action_table__rows;
struct VIAL_Data dt_LALR_action_table__columns;

dt_LALR_action_table__rows.type_alias = "long";
dt_LALR_action_table__rows.address = &LALR_action_table.rows;
dt_LALR_action_table__columns.type_alias = "long";
dt_LALR_action_table__columns.address = &LALR_action_table.columns;

struct VIAL_Data dt_LALR_action_table__Table_SubData[] = {
  dt_LALR_action_table__rows,
  dt_LALR_action_table__columns
};

struct VIAL_Data dt_LALR_action_table__Table;
dt_LALR_action_table__Table.var = "Table";
dt_LALR_action_table__Table.caption = "LALR(1) parsing table";
dt_LALR_action_table__Table.address = &LALR_action_table.Table;   // getting real address
dt_LALR_action_table__Table.type_alias = "LRTableActionItem*";    // above defined alias
dt_LALR_action_table__Table.sub_data_number = 2;                  // rows & columns
dt_LALR_action_table__Table.sub_data =
    (struct VIAL_Data*) dt_LALR_action_table__Table_SubData;      // sub-data: table size

VIALAPI_DeclareData( "LALR(1) parsing", &dt_LALR_action_table__Table );
...
```

Рис. 3. Програмний код, який забезпечує візуальне відображення задекларованих раніше змінних

Зв'язок алгоритму з підсистемою кодування типів даних та підсистемою візуалізації.

Головними підсистемами, які беруть участь у візуалізації алгоритмів, є підсистема кодування типів даних, підсистема візуалізації алгоритмів та сам візуалізований алгоритм. Розглянемо далі схему взаємодії між цими підсистемами:

1. Алгоритм декларує необхідні для візуалізації типи даних.

2. Виклик функції підсистеми кодування типів даних додає до таблиці визначених типів структур даних відповідні записи, що можуть бути однозначно ідентифіковані за іменами.

3. Підсистема кодування типів даних будує шаблон візуалізації (рисування та редагування) задекларованої структури даних. Цей шаблон визначає загальний вигляд форми для відображення даного типу та реакцію підсистеми візуалізації на системні події, які мають відношення до форми (активація та деактивація форми; початок, продовження та закінчення рисування; наведення та зняття фокусу; клік мишкою; натиснення та вивільнення клавіш мишки; наведення, рух та зняття мишки; зміни розмірів; горизонтальний та вертикальний скролінг; зменшення та збільшення масштабу). Для візуального відображення форми шаблону не вистачає лише доступу до реального значення змінної в пам'яті комп'ютера та в ряді випадків – застосування складених структур даних, конкретних розмірів змінної, кількості вкладених змінних тощо. Наприклад, форма введення та редагування рядкової змінної може бути зображена за шаблоном, а форма візуалізації деякої конкретної рядкової змінної потребує доступу до адреси в пам'яті відповідного рядка.

3. Алгоритм декларує необхідні для візуалізації змінні задекларованих попередньо типів даних.

4. Виклик функції підсистеми кодування змінних зареєстрованих типів даних доповнює існуючий шаблон візуалізацію необхідною інформацію, генерує потік команд візуалізації та передає керування підсистемі візуалізації.

5. Підсистема візуалізації інтерпретує базові команди рисування примітивів, що призводить до відображення відповідної форми.

Подібний характер роботи дозволяє задовольнити вимоги, сформульовані на початку роботи, забезпечивши візуалізацію та відлагодження алгоритмів різного виду, в тому числі алгоритмів, використаних в програмному середовищі генерації компіляторів FancyCC 3.0.

Розробка програмного прототипу. Для підтвердження придатності викладеної вище методики до побудови необхідного програмного забезпечення, авторами статті та магістром Житомирського державного технологічного університету Кухарчуком Д.В. була розроблена базова частина програмного прототипу системи автоматизованої побудови візуалізації програм користувача. Вхідний та вихідний його інтерфейси реалізовано поверхнево (рис. 1), основний акцент зроблено на перевірку запропонованої в роботі програмної архітектури. Розробка проводилася мовою C++ за допомогою інструментального засобу MS Visual C++ 6.0.

Згідно з викладеною вище схемою для прототипу системи автоматизованої побудови візуалізації програм користувача були розроблені декілька прикладів, які демонструють його базові можливості. На рис. 4–6 наведено відповідно: простий приклад візуалізації основних типів даних; приклад покрокової візуалізації сортування масиву; приклад візуалізації складених структур даних (зокрема управляючої таблиці синтаксичного розбору SLR(1) в форматі програмної системи FancyCC 3.0).

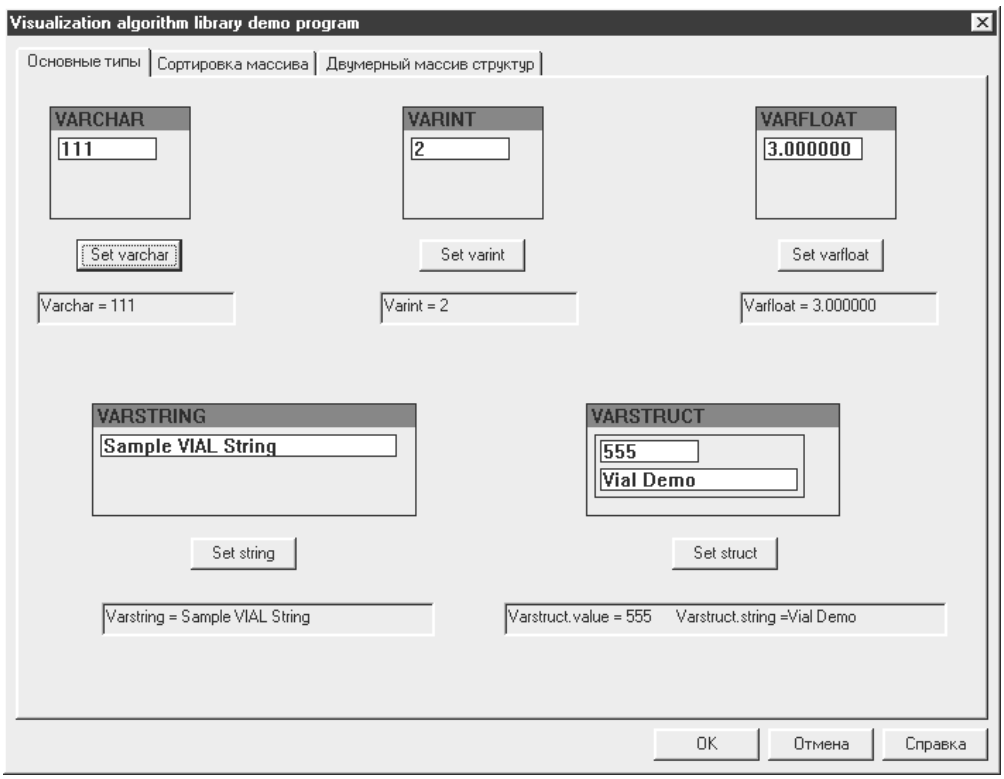

Рис. 4. Робота прототипу системи автоматизованої побудови візуалізації програм користувача: приклад візуалізації основних типів даних

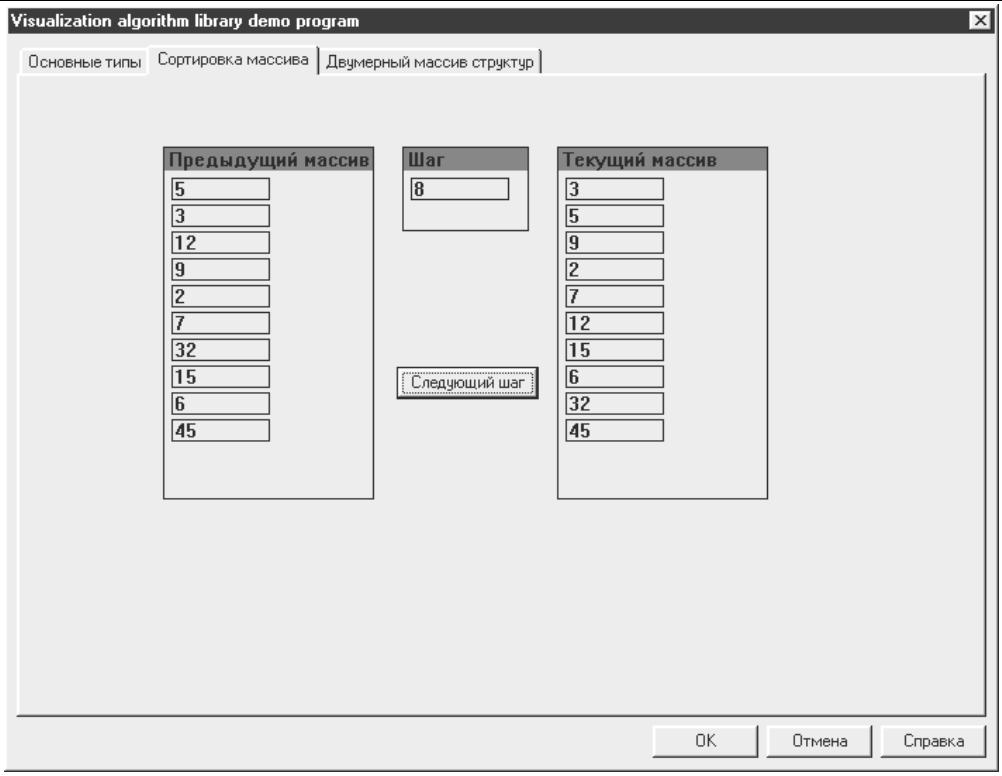

Рис. 5. Робота прототипу системи автоматизованої побудови візуалізації програм користувача: приклад покрокової візуалізації сортування масиву

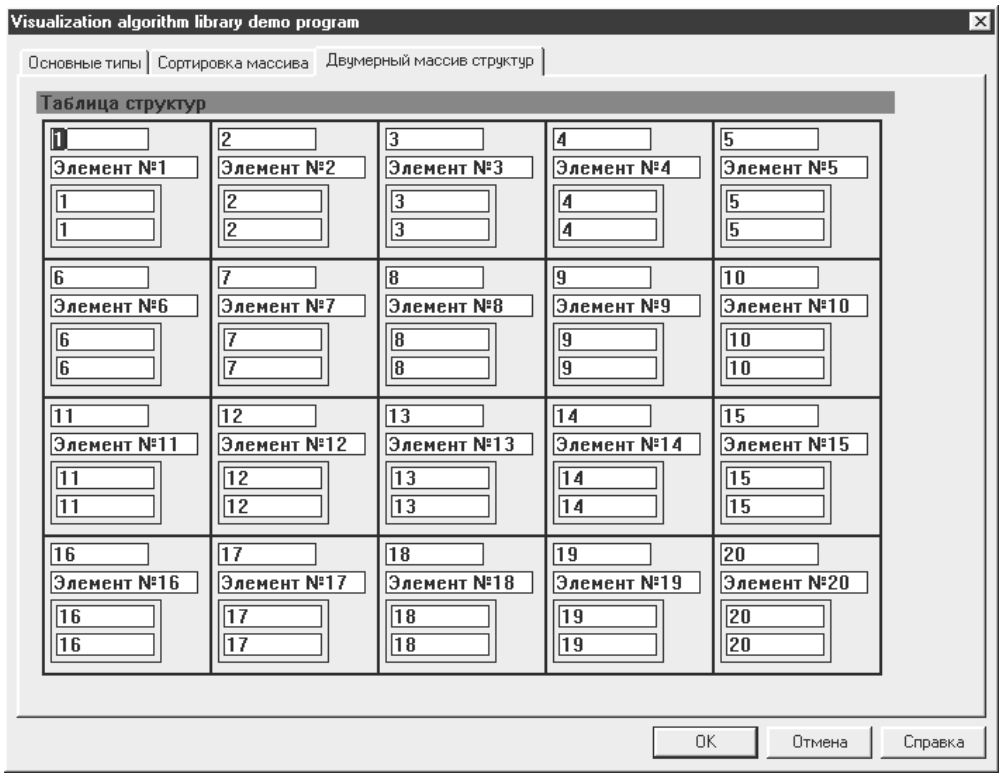

Рис. 6. Робота прототипу системи автоматизованої побудови візуалізації програм користувача: приклад візуалізації складених структур даних

Висновки. Практична перевірка викладених в роботі методів автоматизованої розробки програмних фрагментів візуалізації та відлагодження програм, дозволяє стверджувати, що запропонована схема є життєспроможною та доступною для пересічного користувача. Розробка такого інструментального засобу є важливим фактором успіху програмного середовища генерації компіляторів FancyCC 3.0,

оскільки анімація алгоритмів побудови компілятора підвищує як продуктивність праці програміста, так і рівень його розуміння та сприйняття процесу компіляції.

Застосування запропонованого підходу пропонує, звичайно, ширші перспективи, ніж забезпечення анімацією лише конкретної програмної системи. Зокрема до намірів авторського колективу входить подальше дослідження цієї теми в рамках підвищення якості навчального процесу, тому що активне залучення студентів до інтерактивної взаємодії з програмними системами підвищує їх мотивацію та забезпечує краще збереження здобутих знань.

ЛІТЕРАТУРА:

1. *Левицький В.Г.* Розробка лінгвістичного забезпечення спеціалізованої програмної системи чисельного аналізу // Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 01.05.03: Математичне та програмне забезпечення обчислювальних машин і систем. — Інститут проблем моделювання в енергетиці НАН України. – Київ, 2001.
2. *Kolodnytsky M., Kovalchuk A., Kuryata S., Levitsky V., Samolyuk I.* Interactive Visual Software System for Problem Solving and Teaching in Finite Group Theory // Proceedings of the 23rd International Conference on Information Technology Interfaces. Pula, Croatia, 2001. – P. 409–416.
3. *Kolodnytsky M., Kovalchuk A., Kuryata S., Levitsky V.* The Mathematical Software Implementation for Computational Algebra and Number Theory // Computer Mathematics. Proceedings of the 4th Asian Symposium. Chiang Mai, Thailand, December 17–21, 2000 / Lecture Notes Series on Computing. Volume 8. – World Scientific, Singapore. – P. 291–294.
4. *Ковальчук А.М., Левицький В.Г.* Розробка програмного забезпечення навчального курсу “Проектування компіляторів” // Вестник Херсонского Государственного Технического Университета. – 2001. – № 10. – С. 231–235.
5. *Левицький В.Г.* Опис мови користувача в програмному середовищі генерації мовних процесорів // Вісник ЖІТІ. – 2000. – № 14. – С. 190–197.
6. *Левицький В.Г.* Мова визначення вхідних даних програмного середовища генерації мовних процесорів “FANCYCC 3.0” // Вісник ЖІТІ. – 2000. – № 12 – С. 226–233.
7. *Левицький В.Г., Колодницький М.М., Ковальчук А.М.* Сучасні програмні засоби автоматизованої побудови компіляторів: Навчальний посібник. – Житомир: ЖІТІ, 2002. – 192 с.
8. *Stephen A. Blythe, Michael C. James and Susan H. Rodger*. LLparse and LRparse: visual and interactive tools for parsing. Selected papers of the twenty-fifth annual SIGCSE symposium on Computer science education, 1994. – P. 208–212.
9. *Mona E. Lovato, Michael F. Kleyn*. Parser visualizations for developing grammars with yacc. Papers of the 26th SISCSE technical symposium on Computer science education, 1995. – P. 345–349.
10. *K. Andrews, R. R. Henry, W. K. Yamamoto*. Design and implementation of the UW Illustrated compiler. Proceedings of the SIGPLAN'88 conference on Programming Language design and Implementation, 1988. – P. 105–114.
11. *Robert R. Henry, Kenneth M. Whaley and Bruce Forstall*. The University of Washington illustrating compiler. Proceedings of the conference on Programming language design and implementation, 1990. – P. 223–233.
12. *S. Khuri, Y. Sugono*. Animating parsing algorithms. Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, 1998, Pages 232-236.
13. *A.V.Aho, R.Sethi, J.D.Ullman*. Compilers: Principles, Techniques and Tools, Addison-Wesley Publication, Reading, Massachusetts, 1986.
14. John T. Stasko. The path-transition paradigm: a practical methodology for adding animation to program interfaces. Journal of Visual Languages and Computing, Volume 1. – N. 3. – 213–236, 1990.
15. *B.A. Price, R.M. Baecker, I.S. Small*. A principled taxonomy of software vizualization. Journal of Visual Languages and Computing 4(3):211–266.
16. *Stefan Bruckner*. Data structures in the visualization toolkit. Seminar paper. The Institute of Computer Graphics and Algorithms. Vienna University of Technology, Austria. (http://dods.ipsl.jussieu.fr/prism/wp4a/vtk/vtk_ref1.pdf)
17. *Olivier Esteban, Stephane Chatty, Philippe Palanque*. Visual construction of highly interactive applications. VDB, P. 304–316. – 1995. (http://www.tls.cena.fr/divisions/ PII/presentation/publis.html)
18. Jeffrey L. Korn, Andrew W. Appel. Traversal-based visualization of data structures. IEEE Symposium on Information Visualization. Proceedings {IEEE} Symposium on Information Visualization, P. 11–18. – 1998.
19. *N.C. Shu*. Visual programming. New York: Van Nostrand Reinhold, 1988.

КОВАЛЬЧУК Андрій Михайлович – кандидат технічних наук, доцент кафедри АіКТ Житомирського державного технологічного університету.

Наукові інтереси:
– комп'ютерні інформаційні технології;
– архітектура програмних систем;
– графічні системи та візуалізація даних;
– програмне забезпечення математичного моделювання технічних та екологічних систем;
– використання обчислювальної техніки в навчальному процесі.

ЛЕВИЦЬКИЙ В'ячеслав Георгійович – кандидат технічних наук, доцент кафедри ПЗОТ Житомирського державного технологічного університету.
Наукові інтереси:
– комп'ютерні інформаційні технології;
– розробка мов програмування та предметно-орієнтованих мов, побудова компіляторів;
– програмне забезпечення математичного моделювання технічних та екологічних систем;
– використання обчислювальної техніки в навчальному процесі.

САМОЛЮК Ігор Іванович — аспірант, асистент кафедри АіКТ Житомирського державного технологічного університету.
Наукові інтереси:
– комп'ютерні інформаційні технології;
– програмне забезпечення математичного моделювання технічних систем;
– теорія автоматичного керування;
– використання обчислювальної техніки в навчальному процесі.

Янчук Валентин Миколайович – кандидат технічних наук, доцент кафедри АіКТ Житомирського державного технологічного університету.
Наукові інтереси:
– комп'ютерні інформаційні технології;
– інформаційно-довідникові системи та бази даних;
– програмне забезпечення математичного моделювання технічних та екологічних систем;
– використання обчислювальної техніки в навчальному процесі.

Подано 15.01.2004

Ковальчук А.М., Левицький В.Г., Самолюк І.І., Янчук В.М. Проектування підсистеми відлагодження та візуалізації для програмної системи автоматизованої побудови компіляторів.

Ковальчук А.М., Левицкий В.Г., Самолюк И.И., Янчук В.Н. Проектирование подсистемы отладки и визуализации для программной системы автоматизированного построения компиляторов.

Kovalchuk A.M., Levitsky V.G., Samolyuk I.I., Yanchuk V.M. Design of debugging and visualization subsystem for the compiler construction software.

УДК 004.415

Проектування підсистеми відлагодження та візуалізації для програмної системи автоматизованої побудови компіляторів / Ковальчук А.М., Левицький В.Г., Самолюк І.І., Янчук В.М.

Дана робота присвячена підтримці функцій візуалізації та відлагодження для компілятора компіляторів FancyCC 3.0. Детально викладені вимоги до підсистеми візуалізації, запропоновані способи кодування основних типів даних алгоритму та встановлення зв'язку між структурами даних програми користувача та відповідного візуального представлення. Наведено приклади можливої програмної реалізації підсистеми відлагодження та візуалізації.

УДК 004.415

Проектирование подсистемы отладки и визуализации для программной системы автоматизированного построения компиляторов / Ковальчук А.М., Левицкий В.Г., Самолюк И.И., Янчук В.Н.

В данной работе рассматриваются функции визуализации и отладки для компилятора компиляторов FancyCC 3.0. Детально изложены требования к подсистеме визуализации, предложен способ кодирования основных типов данных алгоритма, способ установления связи между структурами данных программы пользователя и соответствующего визуального представления. Приведены примеры возможной программной реализации подсистемы отладки и визуализации.

УДК 004.415

Design of debugging and visualization subsystem for the compiler construction software / Kovalchuk A.M., Levitsky V.G., Samolyuk I.I., Yanchuk V.M.

We consider debugging and visualization subsystem for the compiler construction software FancyCC 3.0. Detailed features of the subsystem are discussed, including: coding of main data types, links between user data structures and their visual representation. Some examples of development of the software subsystem are given.