

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**Запорізький національний технічний університет**

**МЕТОДИЧНІ ВКАЗІВКИ**

до лабораторних робіт з дисципліни

**«Технології компонентного програмного забезпечення»**

для студентів професійного напрямку

**6.050103 «Програмна інженерія»**

денне відділення

**2015**

Методичні вказівки до лабораторних робіт з дисципліни «Технології компонентного програмного забезпечення» для студентів професійного напрямку 6.050103 «Програмна інженерія» денне відділення /Укл.: І.В. Левада, Л. Ю. Дейнега – Запоріжжя: ЗНТУ, 2015. – 66 с.

Укладачі: ст. викладач І. В. Левада  
ст. викладач Л. Ю. Дейнега

Рецензент: доц., к. т. н. Г.В. Табунщик

Відповідальний за випуск: ст. викладач І. В. Левада

Затверджено на  
засіданні кафедри  
Протокол № 5 від  
24 грудня 2014 р.

## ЗМІСТ

|                                                                       |    |
|-----------------------------------------------------------------------|----|
| ПЕРЕДМОВА .....                                                       | 5  |
| 1 Лабораторна робота № 1.....                                         | 6  |
| Модульне проектування програмних систем структурним методом ....      | 6  |
| 1.1 Мета роботи .....                                                 | 6  |
| 1.2 Порядок виконання лабораторної роботи .....                       | 6  |
| 1.3 Опис середовища функціонування Design/IDEF.....                   | 7  |
| 1.4 Методичні вказівки до виконання лабораторної роботи .....         | 7  |
| 1.5 Зміст звіту .....                                                 | 7  |
| 1.6 Контрольні запитання для самоперевірки.....                       | 7  |
| 1.7 Завдання до лабораторних робіт .....                              | 8  |
| 2 Лабораторна робота № 2.....                                         | 14 |
| Модульне програмування мовою <i>Visual Prolog</i> .....               | 14 |
| 2.1 Мета роботи .....                                                 | 14 |
| 2.2 Порядок виконання лабораторної роботи .....                       | 14 |
| 2.3 Опис середовища функціонування Design/IDEF .....                  | 14 |
| 2.4 Методичні вказівки до виконання лабораторної роботи .....         | 14 |
| 2.5 Зміст звіту .....                                                 | 14 |
| 2.6 Контрольні запитання для самоперевірки.....                       | 15 |
| 3 Лабораторна робота № 3.....                                         | 16 |
| Об'єктна - орієнтоване програмування мовою <i>Visual Prolog</i> ..... | 16 |
| 3.1 Мета роботи .....                                                 | 16 |
| 3.2 Порядок виконання лабораторної роботи .....                       | 16 |
| 3.3 Опис середовища функціонування Design/IDEF .....                  | 16 |
| 3.4 Методичні вказівки до виконання лабораторної роботи .....         | 16 |
| 3.5 Зміст звіту .....                                                 | 16 |
| 3.6 Контрольні запитання для самоперевірки.....                       | 17 |
| Література.....                                                       | 18 |
| Додаток А .....                                                       | 19 |
| Основні поняття функціональної моделі IDEF0 .....                     | 19 |
| А.1 Функціональний блок і дуги .....                                  | 19 |
| А.2 Ієрархічна структура з діаграм .....                              | 20 |
| А.3 Зв'язок функцій дугами на діаграмі декомпозиції .....             | 23 |
| А.4 Порядок виконання функцій на діаграмі декомпозиції.....           | 25 |
| А.5 ICOM-коди прикордонних дуг та тунелі .....                        | 27 |
| А.6 Активації функцій.....                                            | 28 |
| А.7 Контекст моделі та контекст діаграми декомпозиції .....           | 30 |
| А.8 Правила графічного подавання компонент діаграми.....              | 30 |
| А.9 Дерево вузлів моделі. Читання моделі .....                        | 31 |
| А.10 Текст.....                                                       | 31 |
| А.11 Застосування глосарію та діаграм FEO .....                       | 32 |

|                                                                                                       |    |
|-------------------------------------------------------------------------------------------------------|----|
| Додаток Б.....                                                                                        | 33 |
| Реалізація функціональних моделей засобами <i>Design/IDEF</i> .....                                   | 33 |
| Б.1 Створення функціональної модель IDEF0 .....                                                       | 33 |
| Б.2 Додавання назви функції до блоку.....                                                             | 34 |
| Б.3 Створення дуги блоку .....                                                                        | 34 |
| Б.4 Створення мітки дуги.....                                                                         | 35 |
| Б.5 Прикріплення мітки до дуги.....                                                                   | 35 |
| Б.6 Створення діаграми декомпозиції для деталізації блоку.....                                        | 36 |
| Б.7 Розміщення на діаграмі декомпозиції кількох блоків.....                                           | 36 |
| Б.8 Створення відгалуження від дуги .....                                                             | 37 |
| Б.9 Злиття дуги в одну.....                                                                           | 37 |
| Б.10 Створення тунелю.....                                                                            | 37 |
| Б.11 Рух за моделлю IDEF0.....                                                                        | 37 |
| Додаток В.....                                                                                        | 39 |
| Застосування моделі IDEF0 до створення функціональної структури<br>модульної програмної системи ..... | 39 |
| В.1 Роль моделювання при розробці функціональної структури<br>модульної програмної системи .....      | 39 |
| В.2 Побудова функціональної моделі програмної системи.....                                            | 40 |
| В.2.1 Вибір точки зору на вміст моделі, мету створення та<br>контекст моделі .....                    | 40 |
| В.2.2 Розробка контекстної діаграми A-0 .....                                                         | 41 |
| В.2.3 Розробка діаграми декомпозиції A0 .....                                                         | 42 |
| В.3 Семантичні критерії якості функціональних моделей .....                                           | 47 |
| В.3.1 Типи семантичних критеріїв.....                                                                 | 47 |
| В.3.2 Зв'язок функцій діаграми дугами.....                                                            | 48 |
| В.3.3 Зчеплення модулів програми.....                                                                 | 48 |
| В.3.4 Змістова зв'язність процедур модулів програми.....                                              | 50 |
| Додаток Д.....                                                                                        | 53 |
| Модульне програмування мовою <i>Visual Prolog</i> .....                                               | 53 |
| Д.1 Структура модульної програми мовою Visual Prolog .....                                            | 53 |
| Д.2 Описи глобальних предикатів.....                                                                  | 54 |
| Д.3 Приклад модульної програми.....                                                                   | 55 |
| Д.4 Створення модульної програми експертом застосування.....                                          | 56 |
| Д.5 Команди компіляції та побудови програми.....                                                      | 59 |
| Додаток Е.....                                                                                        | 60 |
| Об'єктна – орієнтоване програмування мовою <i>Visual Prolog</i> .....                                 | 60 |
| Е.1 Класи та об'єкти .....                                                                            | 60 |
| Е.2 Створення об'єкту та робота з ним .....                                                           | 62 |
| Е.3 Успадкування класів .....                                                                         | 63 |
| Е.4 Абстрактні класи .....                                                                            | 64 |

## ПЕРЕДМОВА

Методичні вказівки вміщують у стислому вигляді матеріал з дисципліни «Технології компонентного програмного забезпечення», знання якого необхідні при виконанні лабораторних робіт, а також методичні вказівки по виконанню цих робіт.

Матеріал розміщується у додатках до лабораторних робіт.

В процесі виконання лабораторних робіт студенти повинні:

- вивчити основні поняття методології функціонального моделювання систем IDEF0, яка базується на структурному методі;
- одержати навички проектування функціональних моделей програмних систем інструментальним засобом підтримки концептуального моделювання DESIGN/IDEF;
- навчитися розробляти модульні програми засобами мови програмування VISUAL PROLOG, застосовуючи функціональні моделі;
- навчитися розробляти програми засобами мови VISUAL PROLOG об'єктно-орієнтованим методом;
- навчитися обирати методологію програмування та методи створення програм відповідно задачі, що розв'язується.

Знання та вміння з цієї дисципліни, використовуються при розробці складних автоматизованих програмних систем і можуть бути застосовані при розробці дипломних проектів.

## **1 ЛАБОРАТОРНА РОБОТА № 1**

### **МОДУЛЬНЕ ПРОЕКТУВАННЯ ПРОГРАМНИХ СИСТЕМ СТРУКТУРНИМ МЕТОДОМ**

#### **1.1 Мета роботи**

Мета лабораторної роботи:

- засвоїти основні поняття методології IDEF0 необхідні для побудови функціональної моделі модульної програмної системи;
- навчитись використовувати можливості середовища DESIGN/IDEF для розробки функціональних моделей програмних систем;
- навчитися створювати функціональну модель програмної системи, що складається з кількох модулів;
- навчитися застосовувати методи контролю якості функціональної моделі методології IDEF0 для контролю функціональної структури програмної системи та її модулів.

#### **1.2 Порядок виконання лабораторної роботи**

1.2.1 Вивчити основні поняття необхідні для розробки функціональної моделі з додатку А «Основні поняття моделі IDEF0».

1.2.2 Вивчити матеріал «Реалізація функціональних моделей засобами DESIGN/IDEF» додаток Б та навчитися його застосовувати. Додаток містить основні команди інструментального засобу DESIGN/IDEF.

1.2.3 Вивчити матеріал «Застосування моделей IDEF0 до створення функціональної структури модульної програмної системи» додаток В.

1.2.4 Створити засобами DESIGN/IDEF функціональну модель «Облік наукової роботи студентів» з додатку В.

1.2.5 Розробити функціональну модель системи, яку треба розробити за вашим варіантом у завданні на лабораторну роботу (завдання подані нижче).

1.2.6 Аналізувати створену модель системи за критеріями якості моделі. Корегувати модель, якщо необхідно, надрукувати вашу модель та зберегти.

### **1.3 Опис середовища функціонування DESIGN/IDEF**

Система DESIGN версія 3.1 функціонує в ОС WINDOWS 98/XP/NT на IBM та IBM сумісних комп'ютерах.

### **1.4 Методичні вказівки до виконання лабораторної роботи**

1.4.1 Пункти 1.2.1-1.2.3 виконуються з метою познайомити студента з прикладом подаванням функціональної структури системи на базі методології IDEF0 та навчити створювати модель засобом підтримки методології DESIGN/IDEF.

1.4.2 У пункті 1.2.4 треба ознайомитися з завданням до лабораторної роботи. Складне завдання можна виконувати 2 студентам. Складність залежить не від завдання, а від його реалізації. Одержати додатковий матеріал у викладача для докладнішого опису предметної області. На реалізацію моделі відводиться 4 тижні.

1.4.3 Виконати аналіз створеної моделі програмної системи за критеріями якості - сполучення та зв'язність модулів системи.

### **1.5 Зміст звіту**

Звіт містить наступні матеріали.

1.5.1 Мету роботи

1.5.2 Сторінки моделі

1.5.3 Аналіз моделі за критеріями контролю якості - сполучення та зв'язність модулів системи.

### **1.6 Контрольні запитання для самоперевірки**

1.6.1 Як створюють назву функції на діаграмі А-0?

1.6.2 Які властивості повинні мати модулі системи?

1.6.3 Чому на діаграмі декомпозиції не зображується алгоритм?

1.6.4 Назвіть типи зв'язностей функцій на діаграмі декомпозиції вашої моделі.

1.6.5 Які ви знаєте типи зчеплень модулів системи?

### 1.7 Завдання до лабораторних робіт

Подані нижче завдання застосовуються для лабораторних робіт № 1-3. Варіант завдання обирається за номером у списку у журналі. Крім основного завдання програма повинна: пропонувати типи робіт: корегувати дані тесту або файли, що застосовують в системі; виводити вміст файлів.

1 Скласти програму, що тестує знання студента про групу стандартних предикатів Прологу, які оброблюють рядки. Програма виводить на екран контрольне питання про дію певного предикату і відповіді - одну вірну і три невірних. Студент вводить номер вірної, на його думку, відповіді. Програма перевіряє вірність відповіді, якщо відповідь невірна, то виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Обсяг предметної області студент обирає сам.

2 Скласти програму, що тестує знання студента про історичні факти з життя України. Програма виводить на екран контрольне питання і відповідні питанню відповіді - одну вірну і три невірних. Студент вводить номер вірної, на його думку, відповіді. Програма перевіряє вірність відповіді, якщо відповідь невірна, то виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Предметну область і її обсяг студент обирає сам.

3 Скласти програму, що тестує знання студента про групу стандартних функцій мови Lisp, які оброблюють списки. Програма виводить на екран питання про дію функції і відповіді - одну вірну і три невірних. Студент вводить номер вірної, на його думку, відповіді. Програма перевіряє вірність відповіді, якщо вона невірна, то виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Обсяг предметної області студент обирає сам.

4 Скласти програму, що тестує знання студента з української граматики. Програма виводить на екран контрольне питання і відповідні питанню відповіді - одну вірну і три невірних. Студент вводить номер вірної, на його думку, відповіді. Програма перевіряє вірність відповіді, якщо вона невірна, то виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Предметну область і обсяг предметної області студент обирає сам.

5 Скласти програму, що тестує знання студента з англійської мови. Програма виводить на екран слово англійською мовою. Студент вводить його переклад українською мовою. Програма перевіряє вірність відповіді, якщо відповідь невірна, то виводить вірну відповідь. У завданні треба передбачити кілька синонімів для кожного слова. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Обсяг предметної області студент обирає сам.

6 Скласти програму, що перевіряє знання студента з української мови. Програма виводить на екран слово українською мовою. Студент вводить його синонім. Програма перевіряє вірність відповіді, якщо вона невірна, то виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Обсяг предметної області студент обирає сам.

7 Скласти програму, що дозволяє оцінювати характер кандидата при прийомі на роботу. Оцінка виконується на базі двох тестів, що характеризують: працездатність та організованість людини. Тести знаходяться у додаткових матеріалах на сервері.

8 Скласти програму, що дозволяє оцінювати характер робітника для підвищення. Оцінка виконується на базі двох тестів, що характеризують: вміння вести діла та творчий потенціал робітника. Тести знаходяться у додаткових матеріалах на сервері.

9 Скласти програму, що дозволяє тестувати вашу можливість стати бізнесменом. Опит складається з двох тестів, що характеризують: хист вести ділове обговорювання з партнерами і шанс стати мільонером. Тести знаходяться у додаткових матеріалах на сервері.

10 Скласти програму, що навчає дитину змістовим зв'язкам між словами. Тест на зв'язки треба розробити самому на основі семантичних зв'язків: клас - елементи класу, об'єкт - частини об'єкту, дія - інструменти дії, тощо. Програма видає наступні рядки з слів типу:

11 «**Олівець**» відноситься до «**Ручка**», як «**Мотоцикл**» відноситься до...

12 Дитина вводить номер вірної відповіді. з вказаних:

13 «1 **Дорога**» 2 «**Автомобіль**» 3 «**Руль**»

14 Програма перевіряє вірність відповіді, якщо вона невірна виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Предметну область і обсяг предметної області студент обирає сам.

15 Скласти програму, яка створює частотний словник. Програма вводить текст, розбирає його на слова, сортує словник і виявляє різні форми кожного слова. Для сортування опитуйте корені слів для кожної групи. Після чого для кожної групи форм одного слова підраховується частота, з якою зустрічається в тексті це слово. Частота зберігається разом з групою форм. Частотні словники застосовуються для швидкості вибору слів при перевірці їх написання.

16 Скласти програму, що тестує знання студента про стандартні предикати Прологу, які працюють з динамічними фактами. Програма виводить на екран контрольне питання про дію певного предикату і відповіді - одну вірну і три невірних. Студент вводить номер вірної відповіді. Програма перевіряє вірність відповіді, якщо вона невірна виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню

програма видає кількість вірних і невірних відповідей. Обсяг предметної області студент обирає сам.

17 Скласти програму, що перевіряє знання студента про групу стандартних предикатів мови Lisp. Програма виводить на екран питання про дію предикату і відповіді - одну вірну і три невірних. Студент вводить номер вірної відповіді. Програма перевіряє вірність відповіді, якщо вона невірна виводить вірну відповідь. Програма запам'ятовує питання з невірними відповідями і наприкінці питань знову задає їх. По закінченню програма видає кількість вірних і невірних відповідей. Обсяг предметної області студент обирає сам.

18 Скласти програму, що дозволяє швидко знайти у тлумачному словнику за словом зміст статті. Програма повинна дозволяти шукати слово за набором кількох букв і продовжити пошук, якщо слово кілька разів входить у словник. При відсутності слова програма поповнює словник.

19 Скласти програму, що веде бесіду з користувачем на тему «Програмування». Програма задає перше питання і аналізує відповідь користувача. Залежно від належності у відповіді користувача певного ключового слова програма шукає певний варіант наступного свого питання. Якщо у відповіді користувача немає ключового слова, то програма задає наступне питання, що нав'язує тему бесіди. Бесіда продовжується до виходу з програми. Програма повинна передбачити різні виходи з бесіди. Такий метод спілкування називають метод ключових слів. Д. Марселлус. Программирование экспертных систем на Турбо Прологе. С.183-190.

20 Скласти програму, що веде бесіду з користувачем на тему «Музика». Програма задає перше питання і аналізує відповідь користувача. Залежно від належності у відповіді користувача певного ключового слова програма шукає певний варіант наступного свого питання. Якщо у відповіді користувача немає ключового слова, то програма задає наступне питання, що нав'язує тему бесіди. Бесіда продовжується до виходу з програми. Програма повинна передбачити різні виходи з бесіди. Такий метод спілкування називають метод

ключових слів. Д. Марселлус. Программирование экспертных систем на Турбо Прологе. С.183-190.

21 Скласти програму, що за групою введених текстів створює частотний словник можливих сполучень приголосних букв, що зустрічаються в українській мові. Аналогічний словник існує в текстовому редакторі WORD. Такий словник дозволяє виявляти орфографічні помилки в словах. Сполучення букв, що мають більшу частоту, розташовуються попереду, що збільшує швидкість знаходження сполучення.

22 Написати програму «Телефонний довідник». Програма повинна працювати з телефонами підрозділів установ та квартирними телефонами. Програма повинна вміти поповнювати довідник, вилучати компоненту власника та корегувати прізвище власника або назву установи, корегувати номер телефону. Програмний пошук номера телефону можна виконувати за прізвищем; за адресою власника; за назвами підрозділу та його установи, за адресою установи. Можна також виявити за номером телефону його власника.

23 Написати програму «Домашня бібліотека». Записи про книжки повинні мати жанри книг, прізвища авторів, назви книжок, рік видання, прізвище людини, яка позичила книгу, дату позичення, місце на полках. Програма повинна вміти поповнювати книжковий довідник, вилучати запис про загублену книжку, коригувати відомості про позичену книжку, шукати за різними ключами місце розташування книги.

24 Написати програму «Фільмотека». Програма повинна мати жанри фільмів, назви фільмів, режисерів, акторів. Програма повинна вміти поповнювати довідник з фільмами, вилучати загублений фільм та корегувати поля довідника. Програмний пошук фільмів можна виконувати за прізвищем режисера; за назвою фільму; за жанром; за прізвищем актора.

25 Написати програму «Реєстратура». Програма повинна реєструвати хворого, що прийшов до лікаря. Кожний запис містить: табельний № хворого прізвище та ім'я по батькові; дату відвідування лікаря; назву спеціаліста. Програма повинна вміти поповнювати файл реєстрації; вилучати запис або корегувати окремі поля запису; рахувати кількість відвідувань певного спеціаліста або кількість відвідувань за кожним спеціалістом за датою або діапазоном дат. Результати відвідувань спеціалістів видавати на екран.

26 Написати програму «Картка пацієнта». Програма повинна відшукувати у файлі реєстрації пацієнта за табельним № пацієнта всі його відвідування спеціалістів і складати картку пацієнта. Після чого лікар може поповнити картку за датою відвідування скаргами хворого, результатами аналізів та діагнозами.

## **2 ЛАБОРАТОРНА РОБОТА № 2**

### **МОДУЛЬНЕ ПРОГРАМУВАННЯ МОВОЮ VISUAL PROLOG**

#### **2.1 Мета роботи**

Мета лабораторної роботи:

- навчити створювати модульну структуру програми експертом застосувань;
- навчити створювати та налагоджувати програми, які мають модульну структуру, застосовуючи розроблену функціональну модель системи.

#### **2.2 Порядок виконання лабораторної роботи**

2.2.1 Вивчити матеріал «Модульне програмування мовою Visual Prolog» додаток Д.

2.2.2 Розробити та налагодити програму на Пролозі за вашим № завдання.

#### **2.3 Опис середовища функціонування DESIGN/IDEF**

Система DESIGN версія 3.1 функціонує в ОС WINDOWS 98/XP/NT на IBM та IBM сумісних комп'ютерах.

#### **2.4 Методичні вказівки до виконання лабораторної роботи**

2.4.1 Завдання 2.2 виконується програму за моделлю системи, яку ви створили у лабораторній роботі №1.

Після створення експертом коду структури програми подивіться на її структуру за опцією меню PROJECT / PROJECT TREE

На реалізацію програми відводиться 4 тижні. Якщо у групі працює 2 студента, то кожен студент розроблює свої модулі. Демонстрація роботи програми виконується на тестах.

#### **2.5 Зміст звіту**

Звіт містить наступні матеріали.

- 2.5.1 Мету роботи
- 2.5.2 Сторінки моделі
- 2.5.3 Працюючу програму.
- 2.5.4 Тести прогону.
- 2.5.5 Дерево проекту.

## **2.6 Контрольні запитання для самоперевірки**

- 2.6.1 Чому не можна реалізувати модульну програму без опції Project?
- 2.6.2 Для чого застосовуються глобальні предикати?
- 2.6.3 Чи бувають глобальні секції Clauses?
- 2.6.4 Для чого застосовуються глобальні домени?
- 2.6.5 Де розміщують глобальні динамічні факти з характеристикою single?
- 2.6.6 У якій формі об'являються глобальні предикати?
- 2.6.7 Чим визначається тип процедури з секції Clauses?
- 2.6.8 З чого складається секція інтерфейсу модуля?
- 2.6.9 Як об'явити програму написану іншою мовою програмування у програмі мовою Пролог?
- 2.6.10 Для чого потрібен експерт застосувань?

## **3 ЛАБОРАТОРНА РОБОТА № 3**

### **ОБ'ЄКТНА - ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ МОВОЮ VISUAL PROLOG**

#### **3.1 Мета роботи**

Мета лабораторної роботи:

- навчити писати та налагоджувати об'єктно-орієнтовані програми;
- навчити застосовувати у програмі різні методи об'єктно-орієнтованого програмування.

#### **3.2 Порядок виконання лабораторної роботи**

3.2.1 Розглянути матеріал «Об'єктна-орієнтоване програмування мовою Visual Prolog» додаток Е.

3.2.2 Розробити та налагодити об'єктна-орієнтовану програму мовою Пролог за вашим завданням із лабораторної роботи №1.

#### **3.3 Опис середовища функціонування DESIGN/IDEF**

Система DESIGN версія 3.1 функціонує в ОС WINDOWS 98/XP/NT на IBM та IBM сумісних комп'ютерах.

#### **3.4 Методичні вказівки до виконання лабораторної роботи**

3.4.1 Розробити тести для тестування програми.

3.4.2 Завдання налагоджувати на розроблених тестах.

#### **3.5 Зміст звіту**

Звіт містить наступні матеріали.

3.5.1 Мету роботи

3.5.2 Працюючу програму.

3.5.3 Опис класів, об'єктів, методів.

3.5.4 Тести прогону.

### **3.6 Контрольні запитання для самоперевірки**

3.6.1 Пояснити, що таке клас?

3.6.2 Які процедури називають методами?

3.6.3 Чим відрізняються об'єкти одного класу?

3.6.4 Як звертаються до об'єкту певного класу?

3.6.5 Як вилучити об'єкт певного класу?

3.6.6 Чи може об'єктна-орієнтована програма містити звичайні описи та твердження логічного програмування?

3.6.7 Як розташовують у програмі батьківські та похідні класи?

3.6.9 Поясніть роботу механізму успадкування?

3.6.10 Як реалізується в об'єктна - орієнтованому програмуванні поліморфізм?

3.6.10 Наведіть приклади статичних предикатів.

3.6.11 Як можна застосувати абстрактні класи?

3.6.12 Для чого застосовують стандартний предикат THIS?

## ЛІТЕРАТУРА

1 Анатолий Адаменко, Андрей Кучуков Логическое программирование и Visual Prolog.-/ Анатолий Адаменко, Андрей Кучуков Санкт-Петербург: "БХВ Петербург, 2003, -990с.

2 Д. Марселлус Программирование экспертных систем на Турбо-Прологе. / Д. Марселлус – М.: Финансы и статистика, 1994, 254с.

3 І.В. Левада, Л.Ю. Дейнега Конспект лекцій з дисципліни «Технології компонентного програмного забезпечення» для студентів професійного напрямку 6.150103 «Програмна інженерія». /Укл. І.В. Левада, Л.Ю. Дейнега – Запоріжжя: ЗНТУ, 2015, -114с.

4 Конспект лекцій з дисципліни «Методи та засоби комп'ютерних інформаційних технологій» для студентів спеціальності 8.080403 «Програмне забезпечення автоматизованих систем» напрямку «Програмна інженерія». /Укл. І.В. Левада, – Запоріжжя: ЗНТУ, 2008, 94с.

5 Методичні вказівки до лабораторних робіт з дисципліни «Методи та засоби комп'ютерних інформаційних технологій» спеціальності 8.080403 напрямку 6.050103 «Програмна інженерія» денної форми навчання. /Укл. І.В. Левада, – Запоріжжя: ЗНТУ, 2008, 62с.

6 Integrated computer – aided manufacturing (ICAM) architecture part II , volume IV, function modeling manual (IDEF0). Русская версия . Минск: МетаТехнология, 1981.

## ДОДАТОК А

### ОСНОВНІ ПОНЯТТЯ ФУНКЦІОНАЛЬНОЇ МОДЕЛІ IDEF0

#### А.1 Функціональний блок і дуги

Модель IDEF0 - це ієрархічна структура з функціональних діаграм, які визначають що робить система, разом з пояснюючим текстом, глосарієм і діаграмами FEO (For Expression Only).

Базовою одиницею функціональних діаграм моделі є *функція*. Функцією може бути усяка виробнича операція. Назву функції розташовують на діаграмі у середині блоку та записують дієсловом у інфінітиві або дієслівним зворотом.

Блок разом з функцією називають функціональним блоком. Він має ідентифікатор, який розташовано в правому нижньому куту блока. Ідентифікатор блока складається з ідентифікатора діаграми, на якій його розташовано, та номеру блока на діаграмі. Ідентифікатор завжди починається з букви А(activity - функція). Це вказує, що діаграма функціональна. Приклад функціонального блоку подано на рисунку А.1, де функціональний блок №1 розташовано на діаграмі А2.

На діаграмі об'єкти зображують дугами (стрілками). Кожна дуга має мітку, що визначає її зміст. Мітки записують іменником або оборотом з іменником.

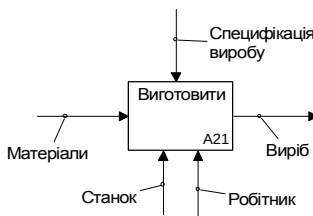


Рисунок А.1 - Приклад функціонального блоку

Функцію зображують разом з об'єктами: що використовуються нею у роботі (на рисунку А1 дуга «матеріали»); керують та ініціюють її (дуга «специфікація виробу»), виконують її (дуги «станок» і «робітник»), виробляються нею (дуга «виріб»). *Об'єкти називають обмеженнями виконання функції*. Функція виконується лише при

наявності необхідних об'єктів.

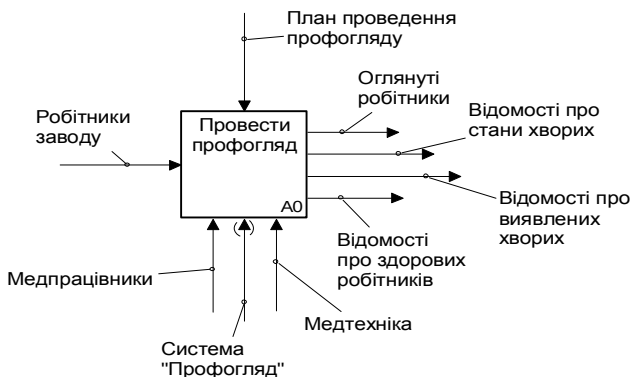
Обмеженнями можуть бути інформація, люди, механізми, інші матеріальні об'єкти.

Дуги проводяться до сторін блоку. Дугу проведену до лівої сторони блоку називають «вхід». Дугу проведену від правої сторони блоку називають «вихід». Дугу проведену до нижньої сторони блоку називають «механізм». Дугу проведену до верхньої сторони блоку називають «керуюча дуга».

Функція включається в роботу та контролюється керуючою дугою. Дуга керування повинна бути хоча б одна для кожної функції. Інших дуг може бути кілька, але не більше 4. Вхідні дуги перероблюються функцією у вихідні дуги.

## А.2 Ієрархічна структура з діаграм

Вершиною ієрархічної структури з діаграм є діаграма А-0. На діаграмі А-0 зображують одну функцію, яка визначає *призначення системи. Мету утворення системи, мету утворення функціональної моделі та точку зору на утворення моделі* розташовують під функціональним блоком. Приклад діаграми А-0 подано на рисунку А.2.



Мета утворення системи: Прискорити та поліпшити якість профогляду

Мета утворення моделі: Показати функціональну структуру автоматизованої системи профогляду

Точка зору: Медпрацівники підприємства

Рисунок А.2 - Приклад діаграми А-0: - «Профогляд»

Зміст моделі залежить від точки зору розроблювачів і від мети утворення моделі. При розробці моделі колективом певного підприємства модель буде мати функції, які виконують на цьому підприємстві. Якщо мета розробки моделі аналіз системних вимог, то в моделі знаходяться функціональні вимоги до системи. Якщо розроблюється програмний продукт, то в моделі є функції: загального призначення програми, функції модулів та функції процедур модулів.

Існує п'ять точок зору на зміст програмного продукту і, як наслідок, на модель системи: договірна (замовника та постачальника); експлуатаційна(оператора та користувача); інженерна(розроблювачів системи та спеціалістів по супроводженню); підтримки(виконавців допоміжних процесів); управлінська(менеджерів).

Функціональне моделювання різних точок зору дозволяє реалізувати різні моделі, що відповідають кожній точці зору. Після чого всі точки зору враховують у одній моделі. Такий підхід дозволить реалізувати якісний програмний продукт. Взагалі функціональне моделювання можна застосовувати для систем будь-якого типу. В розглянутому прикладі – це система профогляду підприємства.

На діаграмі А-0 функція має дуги тільки для відкритих систем. Дуги діаграми А-0 називають зовнішніми дугами, вони являються інтерфейсом між середовищем і системою. Замкнена система не має середовища і тому не має дуг для спілкування. Замкнені системи створюють, якщо не цікавляться середовищем системи.

На діаграмі рівня нижче ніж діаграма А-0 розташовують функції, що деталізують функцію - призначення системи з ідентифікатором А0. Функція, що деталізується, називається вузлом. Діаграма, що деталізує вузол, називається діаграмою декомпозиції. А діаграма, що містить вузол, називається батьківською діаграмою. Діаграма декомпозиції має ідентифікатором ідентифікатор свого вузла, а назва функції стає назвою діаграми декомпозиції. Таким чином діаграма декомпозиції призначення системи має теж ідентифікатор А0.

На діаграмі декомпозиції можна розмістити від 3 до 6 блоків. Кожен блок має свій ідентифікатор, який складається з ідентифікатору

діаграми та номера блоку на діаграмі. Виняток - ідентифікатори блоків на діаграмі A0, вони записуються без 0. Наприклад, A15 – блок номер 5 на діаграмі A1, але блок A2 на діаграмі A0. Приклад діаграми декомпозиції дивися на рисунку A.3.

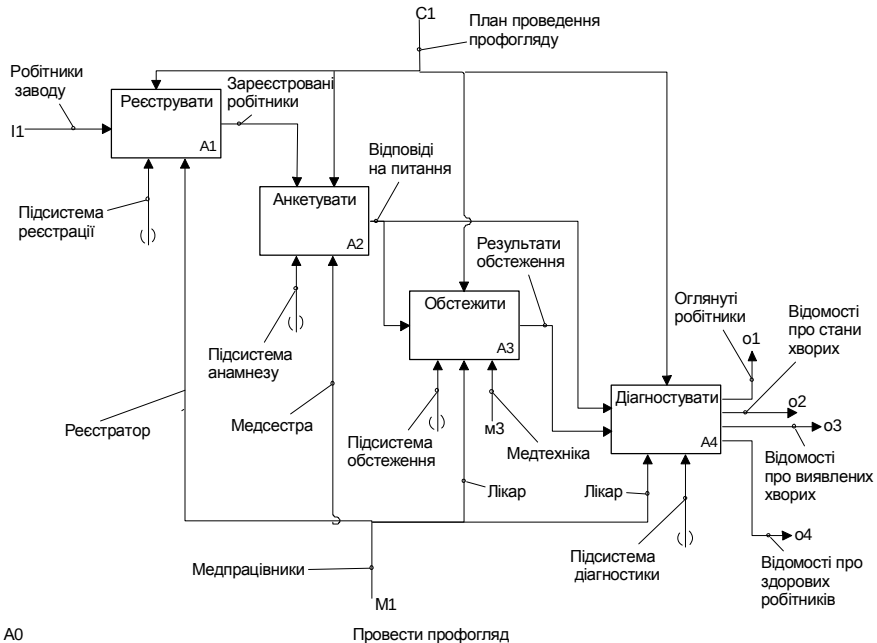


Рисунок А.3 - Приклад діаграми декомпозиції А0: - «Провести профогляд »»

Функції діаграми декомпозиції можуть деталізуватися на нижчому рівні. Утворення діаграм декомпозиції продовжується поки не буде подана суть моделі. Кажуть, що мета створення моделі, досягається. Зв'язок діаграм моделі за ідентифікаторами вузлів та іменами функцій дозволяє розглядати функціональну модель, як єдине ціле, не тільки за змістом, а й формально.

Ієрархічна структура з діаграм спрощує розуміння змісту моделі. Кожний рівень ієрархії призначено для свого читача. Верхні діаграми моделі містять функції загального рівня та призначені для керівництва. Діаграми нижчих рівнів призначені для спеціалістів, що працюють на робочих місцях і містять функції виробничих операцій. Діаграми моделі зображують на стандартній майстер-сторінці.

### А.3 Зв'язок функцій дугами на діаграмі декомпозиції

Функції діаграми декомпозиції можуть деталізувати свій вузол, розкриваючи його зміст. Функції, що розташовані на одній діаграмі декомпозиції, повинні пов'язуватись за змістом. Зв'язок між функціями встановлюється дугами.

Вихідна дуга однієї функції може стати керуючою або вхідною дугою іншої функції. Таким чином встановлюється порядок виконання функцій на діаграмі. Але взагалі, функції на діаграмі можуть виконуватися одночасно, або про порядок їх виконання навіть не можна нічого сказати. Тому кажуть, що *на діаграмі не зображується алгоритм*.

Дугу може використовувати кілька функцій діаграми. Така дуга має мітку на спільній частині, а мітка відноситься до всіх гілок. Дивися дугу «оператор» на рисунку А.4.

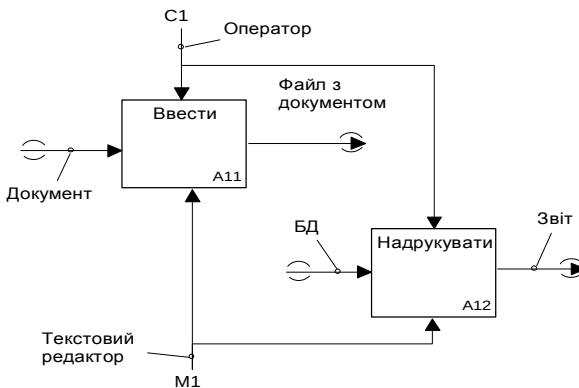


Рисунок А.4 - Приклад використання дуги різними функціями

Дуга загального характеру може деталізуватися при використанні різними функціями. Загальна мітка такої дуги розміщується на спільній частині, а мітки, що деталізують загальну мітку, розташовують на гілках. Спільну частину дуги називають трубопроводом. Дуга «пояснювальна записка» на рисунку А.5 є трубопроводом, а дуги «реферат» і «інструкція системному програмісту» деталізують її.

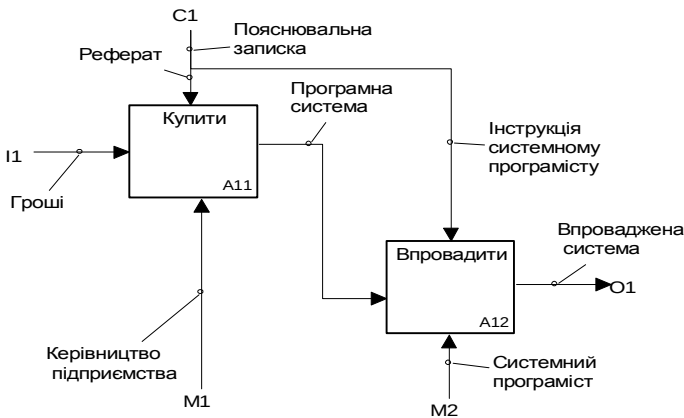


Рисунок А.5 - Приклад деталізації дуги

Якщо кілька функцій діаграми виробляють однотипні дуги, то їх можна злити в одну дугу.

Дуги, які породжуються на діаграмі декомпозиції та мають два кінця на сторонах блоків називають внутрішніми дугами. На рисунку А.5 дуга «Програмна система» внутрішня.

Дуги діаграми декомпозиції, що породжуються вузлом діаграми і мають один вільний кінець називають прикордонними дугами. Прикордонні дуги повинні відповідати дугам вузла, завдяки чому модель буде несуперечливою та діаграми пов'язуються в єдине ціле.

На діаграмі декомпозиції кожна прикордонна дуга на вільному кінці містить код, який показує тип дуги на вузлі і її номер на стороні блоку вузла: І – вхідна, С – керуюча, О – результат роботи функції, М – механізм. Номер дуги на стороні блока вузла рахується зліва направо або зверху вниз.

На рисунку А.5 зображено частину діаграми А1, що одержала прикордонні дуги від свого вузла з діаграми А0. Прикордонні дуги: «гроші» - перша вхідна дуга вузла, «керівництво підприємства» - перша дуга механізму вузла, «системний програміст» - друга дуга механізму вузла, «пояснювальна записка» - перша керуюча дуга вузла, «впроваджена система» - перша дуга результату.

#### А.4 Порядок виконання функцій на діаграмі декомпозиції

Графічна мова подавання моделей не має спеціальних засобів для вказівки порядку виконання функцій. Але дуги можуть диктувати порядок виконання функцій.

З рисунку А.5 видно, що функція «впровадити» не виконається раніше, ніж виконається функція «купити». Поки не з'явиться дуга «програмна система», функція «впровадити» не може працювати. Рисунок А.5 - приклад послідовного виконання функцій.

Про порядок виконання функцій «ввести» і «надрукувати» на рисунку А.4 не можна нічого сказати. Функції можуть виконуватися одночасно або послідовно, бо невідомо, коли поступить дуга «документ».

Функції, які зображено на рисунку А.6 виконуються паралельно. Для них дуги «директива» та «гроші» поступають одночасно.

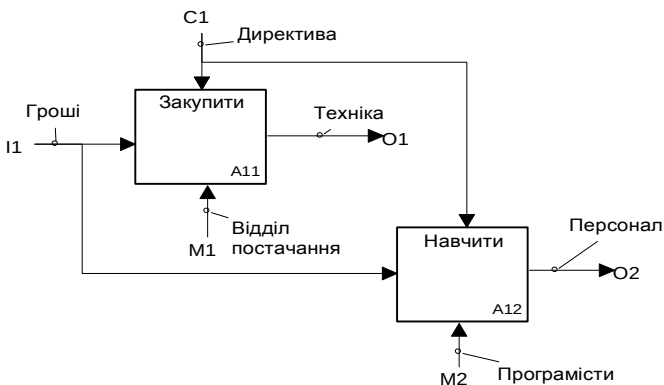


Рисунок А.6 - Приклад паралельного виконання функцій

На рисунку А.7 зображено конкуренцію функцій за матеріал, що користується попитом.

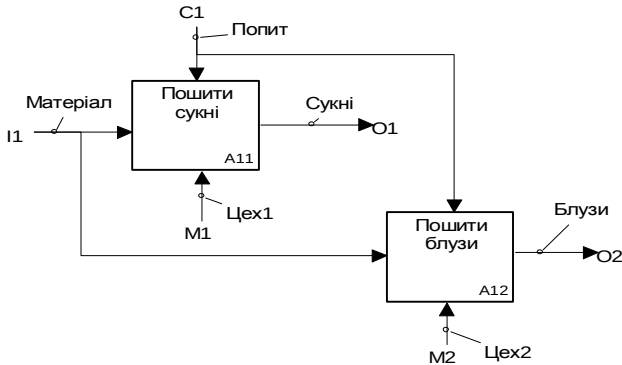


Рисунок А.7 - Приклад конкуренції функцій за матеріальні об'єкти

Діаграми можуть містити зворотні зв'язки при помилках у роботі функцій та для зображення ітерацій. На рисунку А.8 показано приклад зворотного керуючого зв'язку при помилці компіляції програми. Зворотній керуючий зв'язок подається керуючою дугою зверху.

Якщо зворотній зв'язок за ітерацією застосовують для накопичення даних, то він подається вихідною дугою низом.

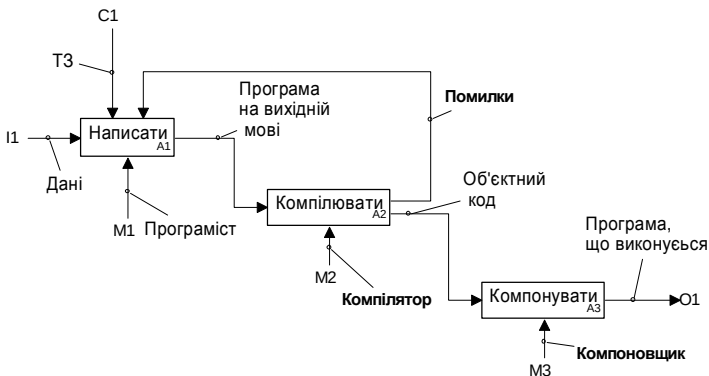


Рисунок А.8 - Приклад діаграми із зворотним зв'язком

## А.5 ICOM-коди прикордонних дуг та тунелі

Зв'язок між вузлом батьківської діаграми та діаграмою декомпозиції встановлюється через дуги вузла. Дуги вузла переносяться на діаграму декомпозиції, якщо це не заперечується. Дуги вузла, що переносяться на діаграму декомпозиції, називають прикордонними дугами. Перенесення дуги заперечується тунелем.

Прикордонні дуги на діаграмі декомпозиції позначають мітками, які називають ICOM-кодами. ICOM-код вказує, де у вузла розташована дуга, що помічена ICOM-кодом на діаграмі декомпозиції.

Вхідні прикордонні дуги позначають буквою І (input), вихідні прикордонні дуги позначають буквою О(output), керуючі прикордонні дуги позначають буквою С(control), прикордонні дуги механізмів позначають буквою М(mechanism). Керуючі дуги та дуги механізмів вузла нумерують зліва направо. Вхідні дуги та вихідні дуги вузла нумерують зверху вниз. ICOM-коди дозволяють швидко знайти на вузлі прикордонні дуги діаграми декомпозиції.

Приклади прикордонних дуг з ICOM-кодами дивися на рисунку А.8. Вузол для діаграми декомпозиції, яка показана на рисунку А.8, подано на рисунку А.9. Якщо на діаграмі декомпозиції деталізується функція, то для неї ICOM-коди призначають заново. ICOM-коди не проставляють на зовнішніх дугах діаграми А-0.

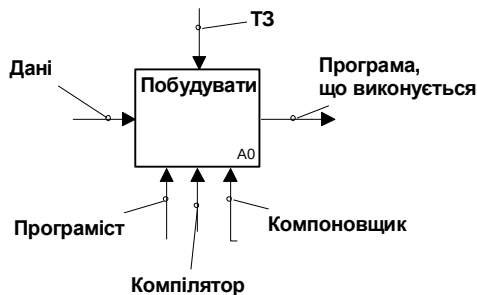


Рисунок А.9 - Приклад вузла для діаграми на рисунку 3.8

Дуга вузла може не використовуватися функціями на діаграмі декомпозиції. Тоді на кінці дуги, розташованому на грані блоку вузла, ставлять тунель( ).

Дуги, які породжуються на діаграмі декомпозиції не від її вузла, відмічають на вільному кінці тунелем ( ). Такі дуги не мають ICOM-коду.

Щоб показати на батьківській діаграмі дугу «клас», а на діаграмі декомпозиції дуги «елементи класу», застосовують тунелі. Дуга «клас» повинна мати тунель на кінці, що розташований на стороні блоку вузла. Дуги «елементи класу» повинні мати тунелі на вільних кінцях. Такий зв'язок між дугами «клас-елементи» називається абстрактним. Дуга «клас» не переходить на діаграму декомпозиції, а дуги «елементи» приходять на діаграму декомпозиції не від вузла. Абстрактний зв'язок – змістовий зв'язок.

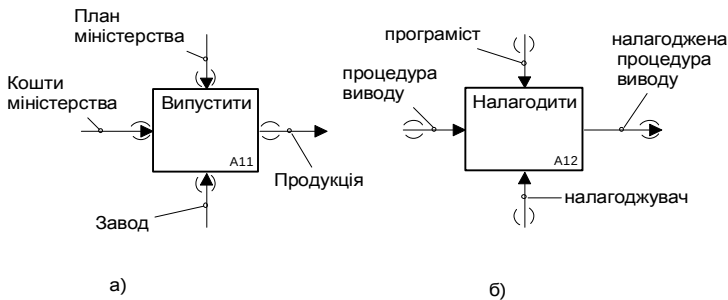
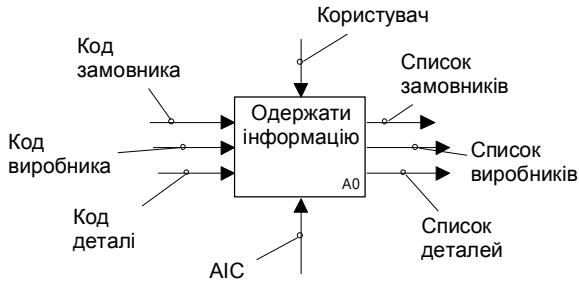


Рисунок А.10 - Приклади використання тунелів

На рисунку А.10 а) подані дуги загального характеру, які не будуть використовуватись на діаграмі нижнього рівня. На рисунку А.10 б) показані дуги, що не прийшли від вузла.

## А.6 Активації функцій

Функція може працювати в різних режимах роботи. Кожен режим вимагає різних комбінацій вхідних, керуючих дуг і механізмів. Кожній такій комбінації відповідають різні вихідні дуги. Комбінації дуг, що необхідні для роботи функції у певному режимі, називають активаціями. Кожному режиму роботи функції-вузла на діаграмі декомпозиції відповідає певна функція та її активація. Якщо функція не вузол, то не завжди можна визначити активації для її режимів роботи.



A-0

Рисунок А.11 - Приклад вузла з активаціями

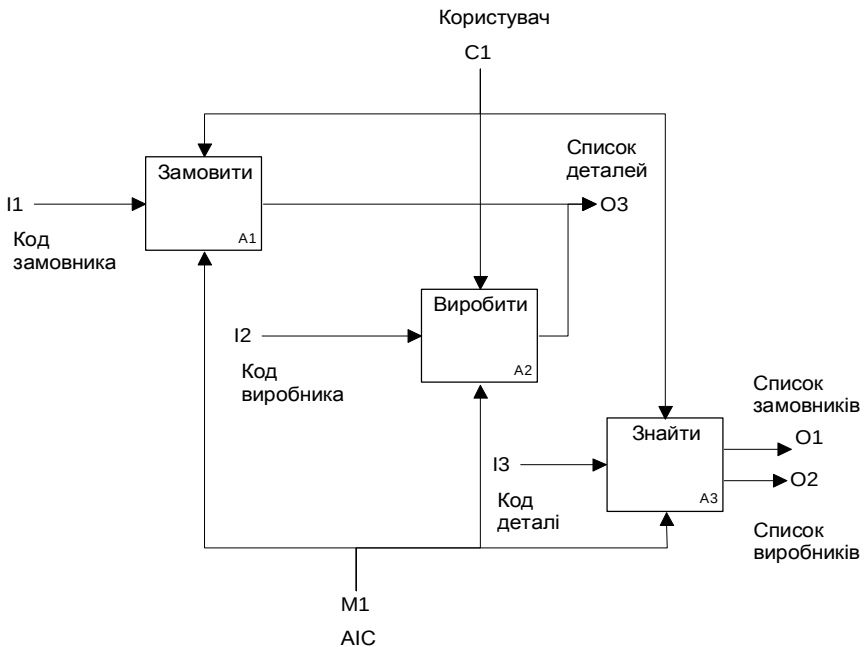


Рисунок А.12 - Приклад діаграми із активаціями вузла A0

На рисунку А.11 подана функція-вузол, що працює у кількох режимах, а на рисунку А.12 її діаграма декомпозиції.

### А.7 Контекст моделі та контекст діаграми декомпозиції

Зовнішні дуги на діаграмі А-0 визначають контекст моделі. Вони пов'язують модель з її середовищем. Діаграму А-0 із зовнішніми дугами називають контекстною діаграмою моделі. Замкнені системи не мають контексту.

Контекстом діаграми декомпозиції являються функції та дуги, які розташовані разом з вузлом на батьківській діаграмі. Для функції «виробити» на рисунку А.12, контекстом її діаграми декомпозиції «виробити» будуть функції «замовити» і «знайти» разом з їх дугами. Контекст діаграми декомпозиції «виробити» зображується у верхньому її полі „CONTEXT”. Дивися рисунок А.13.

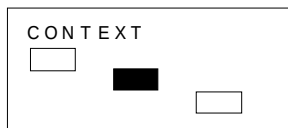


Рисунок А.13 - Контекст діаграми декомпозиції «виробити»

На рисунку А.13 блоки показують функціональні блоки батьківської діаграми та порядок розташування вузла і функцій з контексту.

### А.8 Правила графічного подавання компонент діаграми

Правила графічного подавання компонент діаграми показані нижче.

- 1 Назви функцій розміщують у середині блоків.
- 2 Ідентифікатори блоків розміщують у правому нижньому куту блока.
- 3 На діаграмі блоки розташовують по діагоналі. Проте, це необов'язково. Розміщення блоків диктується простотою читання діаграми.
- 4 Дуги розміщують на вертикальних та горизонтальних лініях. Мінімізуйте перетини дуг. Не доводьте дуги до межі діаграми. Чим менше перетинів дуг, тим зрозуміліша діаграма.
- 5 Мітки, перетини дуг, кути розміщують на деякій відстані від

блоків.

6 Не проводьте до сторони блока більше 4 дуг. Якщо потрібно розмістити більше дуг, об'єднуйте їх за змістом.

7 Розміщуйте ICOM – коди на вільних кінцях прикордонних дуг.

8 Для розгалужених дуг, що входять в декілька блоків, зображуйте одну й ту ж гілку на відповідному місці.

9 Дуги, що не прийшли від вузла, містять тунелі на вільних кінцях.

10 Показуйте накопичення даних на діаграмі дугою, що йде низом, а керуючий зворотний зв'язок дугою, що йде верхом.

### **А.9 Дерево вузлів моделі. Читання моделі**

Структуру функціональної моделі можна зобразити деревом вузлів, або у вигляді змісту книги, де сторінки - діаграми. Структуру моделі називають індексом моделі або змістом моделі. Індекс моделі використовують для читання моделі. Його зручно використовувати для першого знайомства з моделлю, для глибокого вивчення частини моделі, всієї моделі.

Для першого знайомства з моделлю читають діаграми верхнього рівня А-0 і А0, які дають загальну уяву про модель. Для детальнішого знайомства читають діаграми нижчого рівня А1, А2, і т. д. . Для глибокого вивчення частини моделі читання моделі виконуйте зверху вниз за обраною гілкою дерева. Тобто, для детального вивчення функції обирають з індексу всі діаграми, що деталізують дану функцію. Для читання діаграми декомпозиції треба спочатку розглянути діаграму, потім розглянути вузол діаграми і визначити головний вхід, керування та вихід. Використовуючи головні дуги, знайти на діаграмі декомпозиції головний шлях – від головного входу до головного виходу. Визначити, чи є на діаграмі побічні шляхи. Ознайомитися з текстом, глосарієм, діаграмою FEO.

### **А.10 Текст**

Кожна діаграма моделі може доповнюватися однією текстовою сторінкою. У тексті розповідають про те, що не можливо відобразити графічно. Текстова сторінка не повторює змісту діаграми, а доповнює

її. Якщо на одній сторінці не можна розмістити весь текст, то його можна також структурувати, деталізуючи виділені на сторінці фрагменти тексту.

Текст до діаграми А-0 відносять до всієї моделі, він дає загальну характеристику всієї моделі. Текст записується у формі змістовного оповідання. Всі текстові сторінки, структуровані так, як структурована модель. Текст роз'яснює, як досягається мета утворення моделі та точку зору розроблювача. Текст утворювати необов'язково. Але, якщо його утворюють, то після закінчення розробки діаграми(моделі).

### **А.11 Застосування глосарію та діаграм FEO**

Глосарій – словник, в якому визначають спеціальні терміни тексту, аббревіатури, поняття графічної моделі (назви функцій, дуг). Визначення необхідні для однозначного розуміння змісту моделі. Поняття визначають через атрибути або текстом. У середовищі DESIGN/IDEF структура глосарію утворюється автоматично для назв функцій та дуг. Означення назв вводить користувач. Наприклад медпрацівники: лікарі, медсестри, лаборанти. Якщо користувач ввів означення для мітки дуги чи назви функції, то означення автоматично з'явиться у глосарію для такої ж мітки чи назви. Декілька назв, що мають однакові означення, вилучаються програмно, як надлишок.

DESIGN/IDEF дозволяє фіксувати ціну виконання функцій. Визначаючи ціну виконання для функцій, що не є вузлами, можна автоматично підрахувати ціну виконання вузлів або системи. Функція може виконуватися під час роботи її вузла декілька разів. Частоту виконання функції вказують у глосарію. У глосарію вказують для функцій тривалість виконання щоб підрахувати тривалість виконання всієї системи. Діаграми FEO(For Exposition Only) використовують для ілюстрації рисунками, графіками, діаграмами різних типів деяких моментів зображених на діаграмах IDEF0.

## ДОДАТОК Б

### РЕАЛІЗАЦІЯ ФУНКЦІОНАЛЬНИХ МОДЕЛЕЙ ЗАСОБАМИ DESIGN/IDEF

*Загальні відомості:* Для виконання дій над об'єктом діаграми моделі треба зробити об'єкт активним – обрати його лівою кнопкою миші. Активний об'єкт виділяється чорними квадратами.

Команди оболонки визначають дію над об'єктом діаграми. Для зручності роботи кожна команда має свій маркер, який показує яка команда активна в певний момент. По замовчанню маркер '+’.

Майже всі команди виконуються за однією схемою.

1 Обрати об'єкт лівою кнопкою миші.

2 Обрати команду лівою кнопкою миші або гарячою клавішею. Маркер прийме форму відповідну команді.

3 Виконати дію вказану у команді. Дію можна повторювати.

4 Щоб завершити команду, треба натиснути ESC або клацнути правою кнопкою миші. Показчик змінить свою форму на символ '+’.

Дії над об'єктами описано у командах оболонки.

Функціональна модель подається ієрархічною структурою з діаграм. Рухатися від одної діаграми моделі до іншої треба тільки за спеціальними командами руху. Забороняється рухатися просто за сторінками. Такий рух приводить до збійної ситуації.

Нижче подано команди оболонки, необхідні для побудови функціональної моделі.

#### **Б.1 Створення функціональної модель IDEF0**

Створення функціональної моделі необхідно виконувати у наступному порядку.

1 Відкрити файл з назвою STARTUP.IDD з каталогу STARTUP.

На робочому столі відкриється вікно, що містить майстер - сторінку, пронумеровану 10001. Майстер сторінка має стандартну форму, яка буде автоматично переноситься на нові сторінки моделі створені користувачем.

2 Обрати команду **New\_Page** з меню **Page**. На екрані з'являться опції, що вказують на тип нової сторінки. Обрати опцію **New A-0 Page** і натиснути **OK**. Утворюється сторінка **A-0**, яка буде вершиною ієрархічної моделі з діаграм. Сторінка буде містити стандартну форму з майстер сторінки і автоматично створений один активний блок. По замовчанню ідентифікатор функціонального блоку на діаграмі **A-0** позначається «**A0**».

3 Обрати нове ім'я вашої моделі, щоб замінити ім'я **STARTUP**. Обрати каталог, що буде містити нову модель.

4 Зберегти файл **STARTUP.IDD** під обраним ім'ям. Для цього вибрати команду **SAVE AS** з меню **File**.

## **Б.2 Додавання назви функції до блоку**

### **Команда **TURN OFF** з меню **TEXT (F2)****

1 Установіть кирилицю за командою **ATTRIBUTES** з меню **TEXT**.

Для роботи з кирилицею оберіть шрифт **ARIAL CYR**. Шрифти розташовані в алфавітному порядку. Закріпити шрифт для майбутнього використання можна вказавши опцію **Set for Future Objects**.

З цією ж метою можна використовувати команду **ATTRIBUTES** з меню **IDEF0**. В цьому випадку переконайтеся, що ви закріпили вибір атрибутів опцією **APPLY**.

1 Оберіть команду **TURN OFF** з меню **TEXT**. Показчик розміщується усередині блоку. Уведіть текст.

2 Завершення уводу тексту виконується за командою **TURN ON** з меню **TEXT**.

## **Б.3 Створення дуги блоку**

### **Команда **—ARROW** з меню **CREAT(F5)****

Використовуйте цю команду, щоб малювати дугу між двома **DESIGN/IDEF** об'єктами.

Натисніть кнопку миші усередині об'єкта-джерела і перетягніть показчик дуги усередину об'єкта адресата. Звільніть кнопку миші,

коли дуга торкнеться бажаної сторони об'єкта. Об'єктом може бути блок, ICOM – код дуги.

Намалювати дугу, у якої тільки один кінець знаходиться на об'єкті, можна встановивши покажчик на певне місце сторінки, натиснути ліву кнопку і перетягнути покажчик до сторони блоку.

Щоб намалювати вихідну дугу потрібно спочатку намалювати мітку, де буде закінчуватися голова дуги або малювати дугу в зворотному напрямку.

#### **Б.4 Створення мітки дуги**

##### **Команда LABEL з меню CREAT(F3)**

Мітку використовують, щоб визначити дугу функції. Коли інструмент мітки активний, текст включається автоматично. Для текстових міток використовуйте шрифт ARIAL CYR.

- 1 Оберіть команду CREAT/LABEL.
- 2 Установіть покажчик на обране місце та клацніть кнопкою миші. Установка покажчика вказує точку введення мітки на сторінці.
- 3 Щоб розірвати рядок натисніть клавішу ENTER.
- 4 Однією командою можна створити кілька міток.
- 5 Закінчити команду можна клацнувши правою кнопкою миші.
- 6 Щоб перемістити мітку по сторінці, оберіть мітку і рухайте.
- 7 Корегувати мітку треба за командою F2.
- 8 Командою можна розмістити на діаграмі ціль та точку зору на модель.

#### **Б.5 Прикріплення мітки до дуги**

##### **Команда ATTACH LABEL з меню IDEF0(Shift-F3)**

- 1 Оберіть мітку.
- 2 Позиціонуйте мітку туди, де Ви бажаєте, щоб вона була коло дуги.
- 3 Оберіть команду Attach Label.
- 4 Позиціонуйте покажчик на дузі в місці прив'язки.
- 5 Коли дуга замигає, натисніть кнопку миші. Текст мітки прив'язаний до дуги. У місці з'єднання з'являється маленьке коло.

6 Щоб видалити прикладену мітку, виберіть прикладену мітку і виберіть команду Detach Label.

### **Б.6 Створення діаграми декомпозиції для деталізації блоку**

#### **Команда Decompose з меню Page(Pg Down)**

Наступні сторінки моделі утворюються тільки за командою Decompose.

Команда створює нову сторінку, на якій буде розміщена діаграма, що деталізує функціональний блок(вузол). На діаграму декомпозиції автоматично переносяться мітки прикордонних дуг, ICOM-коди дуг. Через прикордонні дуги діаграми пов'язуються між собою. Так будується ієрархічна структура моделі.

Кожна сторінка, що створюється в моделі, буде містити стандартну форму з майстер-сторінки.

### **Б.7 Розміщення на діаграмі декомпозиції кількох блоків**

**Команда PLACE BOXES з меню IDEF0 (SHIFT-F11) створює кілька блоків**

Використовуйте команду, щоб автоматично розмістити кілька блоків на поточній сторінці (кількість блоків за замовчуванням три).

Можна вводити самому будь-яку кількість, що не перевищує максимальне число блоків, дозволених для сторінки. Ви можете змінювати максимальну кількість блоків за допомогою команди ATTRIBUTES у меню IDEF0.

Блоки рівномірно розташовуються по діагоналі від верхнього лівого кута до правого нижнього кута сторінки.

**Команда CREATE IDEF BOX з меню IDEF0 (F11) створює один блок**

За вибором команди, покажчик приймає форму об'єкта, який буде зображено. Установіть покажчик в те місце, де Ви хочете розмістити вершину блоку і клацніть кнопкою миші. Блоку буде призначено номер наступний за максимально існуючим на IDEF0-діаграмі.

## **Б.8 Створення відгалуження від дуги**

### **Команда BRANCH з меню IDEF0 (F12)**

1 Команда малює нову дугу, що відгалужується, від обраної дуги. Оберіть дугу установивши на неї покажчик і клацнувши кнопкою миші.

2 Оберіть команду. Покажчик прийме форму дуги.

3 Помістіть покажчик на сторону блоку IDEF0 чи мітку, де повинна закінчитися нова дуга. Границя об'єкта замигає.

4 Натисніть кнопку миші, коли покажчик установиться на стороні об'єкта адресата, у місці де повинна закінчуватися дуга. Дуга намальована.

Граматичний контроль стрілок IDEF0 GRAMMAR повинен бути включений.

## **Б.9 Злиття дуги в одну**

### **Команда JOIN з меню IDEF0 (Shift-F12)**

Команда малює нову дугу, що зливається з обраною старою дугою.

1 Оберіть дугу.

2 Оберіть команду, покажчик прийме форму дуги.

3 Установіть покажчик на блок чи мітку звідкіля буде малюватися нова дуга для об'єднання з обраною. Границі покажчика замигають.

4 Щиглик кнопки миші завершує створення нової дуги. Граматичний контроль стрілок треба включити.

## **Б.10 Створення тунелю**

Клацнути на кінці дуги, де бажано розмістити тунель, двічі.

## **Б.11 Рух за моделлю IDEF0**

### **Команда Page Structure з меню Page**

Переміщує у вікно Page Structure, яке відображає в ієрархічному порядку всі сторінки вашої моделі. Від цього вікна Ви можете перейти до будь-якої сторінки в моделі.

Поточна сторінка позначена чорною точкою. Для переходу до іншої сторінки, виберіть ім'я в Структурі Сторінок. Обрана сторінка відкриється або якщо ця сторінка знаходиться на Робочому столі (створена окремим файлом) сторінка робиться активною.

Натисніть ESC для повернення до поточного сторінки.

Перехід на діаграму декомпозиції вузла можливо виконати обравши вузол та двічі клацнувши усередині блоку.

Перехід на діаграму декомпозиції вузла можна також виконати виконав команду Parent з меню Page.

Повернутися з діаграми декомпозиції на батьківську діаграму можна обрав команду Child з меню Page.

## **ДОДАТОК В**

### **ЗАСТОСУВАННЯ МОДЕЛІ IDEFO ДО СТВОРЕННЯ ФУНКЦІОНАЛЬНОЇ СТРУКТУРИ МОДУЛЬНОЇ ПРОГРАМНОЇ СИСТЕМИ**

#### **В.1 Роль моделювання при розробці функціональної структури модульної програмної системи**

Подання структури програмної системи функціональною моделлю, дозволяє розроблювати модульну структуру системи, розробляти пов'язані за змістом процедури модулів. Розробка функціональної структури системи починається зі створення діаграми А-0.

На діаграмі А-0 функціональний блок А0 зображує функцію всієї системи, тобто зображує загальне призначення системи, а дуги показують інтерфейс програми з навколишнім середовищем.

До середовища програмної системи відносять її користувачів; інші програмні системи, техніку, організаційні структури підприємства, що взаємодіють з нею. Перелічені компоненти середовища виконують кожен свою функцію, але мають разом з системою спільну мету.

Функція призначення системи деталізується на діаграмі нижчого рівня А0 загальними функціями, які реалізують призначення системи. Кожна загальна функція діаграми подає окремий модуль програмної системи. Діаграма А0 показує зв'язок між модулями.

Кожна функція, відповідна модулю програми, може в свою чергу деталізуватися його процедурами на діаграмі нижчого рівня. Наочність графічного подання процедур дозволяє аналізувати вміст модулів та одержувати якісні модулі.

Подання вмісту модулів у вигляді діаграм декомпозиції, дозволяє зробити функціональну структуру системи простою, зрозумілою і дає можливість модифікувати її.

Модель можна застосовувати для подання програмного комплексу. Тоді на діаграмі А0 буде показано функції кожної програми комплексу і зв'язок між ними. Така діаграма дозволяє оглянути весь програмний комплекс разом.

## **В.2 Побудова функціональної моделі програмної системи**

Розглянемо процес створення функціональної моделі програмної системи на прикладі створення функціональної моделі системи «Облік наукової роботи».

Для розробки моделі треба:

1 Обрати точку зору на вміст моделі, мету створення моделі та контекст моделі.

2 Розробити контекстну діаграму А-0.

3 Розробити діаграму декомпозиції верхнього рівня А0, яка відображає функції модулів та зв'язок між модулями.

4 Розробити діаграми декомпозиції для вмісту кожного модуля.

### **В.2.1 Вибір точки зору на вміст моделі, мету створення та контекст моделі**

Метою створення моделі може бути функціональна модель меню, функціональна структура програми, розробка функціональних вимог до системи, тощо.

Від мети створення моделі залежать функції моделі. Наприклад, якщо метою є розробка меню системи, то модель подає структуру опцій меню. Хороше розташування опцій у структурі меню дозволяє швидко знаходити необхідну опцію. Повнота меню дозволяє користувачеві гнучко застосовувати можливості системи.

Мета створення моделі «Розробка функціональних вимог до системи» дозволяє замовнику достатньо повно розробити функціональні вимоги до системи.

Ми будемо вважати метою створення моделі системи «Облік наукової роботи» розробку функціональної структури програми. Тобто, застосуємо модель, як засіб розробки модулів, процедур системи, їх взаємозв'язків.

Від обраних точки зору і контексту також залежить вибір функції моделі. Будемо застосовувати договірну точку зору на створення моделі - точку зору замовника та постачальника.

Під контекстом моделі будемо вважати середовище, в якому застосовуватиметься система. До середовища віднесемо викладачів і студентів кафедри, які займаються наукою; існуючі види наукової діяльності; відділ науки університету і його вимоги до звіту про науку.

Точка зору на модель, контекст і мета створення моделі зображуються на діаграмі А-0.

## **В.2.2 Розробка контекстної діаграми А-0**

Функція контекстної діаграми – це призначення системи.

Придумайте назву функції, яка б указувала на призначення системи. Діаграма А-0 - основа всієї моделі.

Якщо система призначена для виконання кількох основних функцій, і важко знайти загальну для них назву, то спочатку краще розробити діаграму А0, на якій подаються кілька основних функцій системи, а потім узагальнити назви цих функцій і створити діаграму А-0.

Для нашого прикладу призначенням системи є ведення обліку наукової діяльності студентів.

Подайте дугами цієї функції інтерфейс системи з середовищем. Керує роботою системи користувач. За допомогою меню система опитує у користувача інформацію для роботи системи. Вказані вхідні дані функція перероблює в результати роботи. Механізм функції сама система.

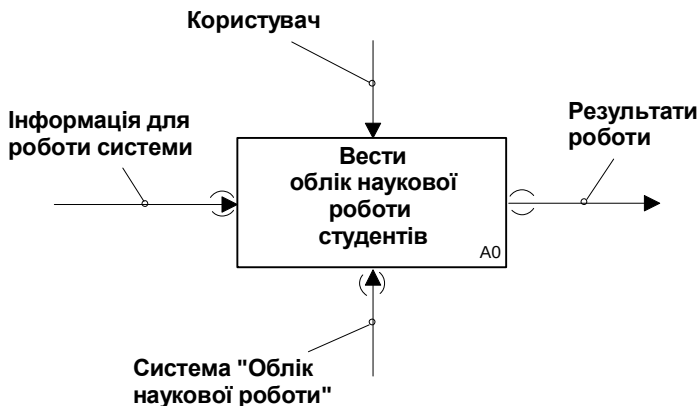
Тому вхідною дугою функції буде дуга «Інформація для виконання роботи». Вихідною дугою є дуга «Результат роботи». Дуга «Система «Облік наукової роботи»» є механізмом. Дуга «Користувач» керуюча.

Вказані дуги складають інтерфейс між системою і середовищем. Оскільки функція діаграми А-0 загальна, то і дуги сформульовані у загальній формі. На рисунку В.1 подано приклад діаграми А-0 для системи «Облік наукової роботи».

Рівень формулювання дуг повинен відповідати рівню формулювання їх функцій. Функції на діаграмі декомпозиції деталізують призначення і значить їх дуги повинні деталізуватися або розподілятися між функціями.

Тому дуги «Інформація для роботи системи», «Результати роботи» та «Система «Облік наукової роботи»» мають на кінцях, що лежать на сторонах блоків тунелі. Тунелі заперечують дугам з'явитися на діаграмі декомпозицій функціонального блоку.

Дуга «Користувач» не деталізується, бо користувач працює з системою на всіх рівнях. Приклад діаграми А-0 для системи «Облік наукової діяльності студентів» подано на рисунку В1.



**Мета роботи системи:** Поліпшити облік наукової роботи кафедри  
**Мета створення моделі:** Розробка функціональної структури системи  
**Точка зору:** Замовника системи і розроблювача

Рисунок В.1 - Приклад діаграми А-0: Система «Облік наукової роботи студентів»

### В.2.3 Розробка діаграми декомпозиції А0

Функції діаграми А0 являються загальними функціями, що реалізують призначення системи.

Кожна функція діаграми подає окремий модуль програмної системи. Діаграма А0 показує зв'язок між модулями. На діаграмі подаються функції модулів системи. Тому механізмами всіх функцій на діаграмі є самі модулі системи.

Діаграма А0 вважається справжньою вершиною моделі. За нею можна точно визначити, що робить система.

Утворення діаграми декомпозиції – індивідуальна, творча робота. Проте існує примірний план робіт по її створенню.

1 Утворюють список даних, які використовуються функцією вузла, керують нею, виконують її або виробляються нею. Близькі за змістом та рівнем дані об'єднують.

Функція вузла «Вести облік наукової роботи» застосовує наступні дуги: користувач; пункт меню; вид роботи; робота з даними; вид звіту; БД; відкоригована БД; надрукований звіт; оперативна інформація на екрані.

Групуючи дані за їх застосуванням можна одержати такі узагальнені функції системи: «обрати вид роботи», «вести БД», «зробити звіт». Кожну функцію буде виконувати певний модуль: «СУБД»; «Генератор звітів»; «Меню» - це дуги механізми.

У мові VISUAL PROLOG 5.2 кількість об'явлених предикатів(глобальних і локальних) у кожному модулю обмежена. В той же час головний модуль програми повинен містити всі глобальні об'яви.

Щоб розмістити у головному модулю більшу кількість глобальних предикатів, створюють додатково модуль RUN. Модуль RUN містить тільки глобальні об'яви доменів та предикатів з усіх модулів і ціль з глобальним предикатом RUN, що ініціює модуль роботи з меню системи. Тому до дуг можна додати ще дві дуги «RUN» і «Ініціювання Меню».

2 Списки даних групують за принципом «із – в», що вказує на переробку певних даних у інші дані. Поповніть, якщо треба список дуг. За кожною групою даних імена функцій модулів розміщують у блоках на діаграмі декомпозиції A0.

*Правило:* Для вірного формулювання назви функції контекстної діаграми A-0 і функцій діаграми A0 перевірте:

- а) чи охоплює назва функції діаграми A-0 зміст функцій на діаграмі A0;
- б) чи не ширше зміст функцій діаграми A0, ніж назва функції на діаграмі A-0. Це правило застосовують при подальшій декомпозиції вузлів.
- в) чи повний список дуг на діаграмах A-0 і A0.

Діаграма A0 докладніше показує, яким чином вхід головної функції на діаграмі A-0 перероблюється у вихід. На діаграмі A0 обов'язково присутні вхідні та вихідні дуги діаграми A-0, якщо це не заборонено тунелями. Групування дуг і відповідні їм назви функцій подано в таблиці В.1.

Таблиця В.1 - Групування дуг та формування імен функцій

| Вхідні дуги   | Вихідні дуги                                       | Керуючі дуги                            | Дуги механізми          | Назви функцій      |
|---------------|----------------------------------------------------|-----------------------------------------|-------------------------|--------------------|
|               | Ініціювання меню                                   | Користувач                              | Модуль RUN              | Ініціювати систему |
| Пункт меню    | Вид роботи; вихід                                  | Ініціювання меню; користувач; керування | Модуль меню             | Вибрати опцію меню |
| Робота з БД;  | Керування; Відкоригована БД                        | Користувач                              | Модуль СУБД             | Вести БД           |
| Вид звіту; БД | Оперативна інформація на екрані; надрукований звіт | Користувач                              | Модуль Генератор звітів | Зробити звіт       |

Блоки з функціями розташовують діагонально від лівого верхнього кута до нижнього правого кута, якщо необхідно для простоти читання діаграми, то правило порушують. Креслять дуги біля кожного функціонального блоку і надають дугам імена - мітки.

3 З'єднують функції діаграми дугами природними для них зв'язками.

4 Модифікують діаграму, об'єднуючи споріднені функції в одну більш загальну функцію, або розщеплюють функції для більшої деталізації. Дії виконують з метою одержати всі функції одного рівня деталізації.

5 Перевіряють рівень деталізації дуг. Рівень деталізації дуг повинен відповідати рівню деталізації функцій. Об'єднуйте споріднені дуги, якщо потрібно.

6 Аналізують імена функцій та дуг, щоб зробити їх конкретнішими та зрозумілішими. Спеціальні терміни заносять в глосарій.

7 Перевіряють, чи є на утвореній діаграмі всі важливі випадки роботи функції вузла; побічні шляхи.

8 Порівнюють всі дуги зі списком даних, щоб не загубити дані.

9 Малюють остаточний варіант діаграми за правилами синтаксису.

Остаточний варіант діаграми A0 подано на рисунку В.2.

На діаграмі A0 модуль «RUN», ініціює модуль роботи з меню через його глобальний предикат.

Модуль «Меню» опитує у користувача вид роботи і ініціює глобальний предикат модуля відповідного обраному виду роботи. Після чого предикати модуля виконують вказану роботу. Після закінчення роботи модуль повертає керування в модуль «Меню». Вибір користувачем пункту меню «Вихід» завершує роботу системи.

Модуль «Вести БД» виконує створення БД, її поповнення та корегування. Після виконання роботи дуга «БД» змінює свою якість, перетворюється у дугу «Відкоригована БД».

Модуль «Зробити звіт» виводить обраний користувачем звіт на екран або на друк залежно від обраного пункту меню.

Розроблена модель дозволяє не тільки створювати нову програмну систему, а і розвивати її, модифікувати при виправленні помилок.

Цьому сприяють засоби програми DESIGN/IDEF, які дозволяють просто поповнювати діаграми моделі, переносити вміст діаграми або її частини на інший рівень. Важливо, що при цьому автоматично створюються ICOM – коди та тунелі.

Моделі зроблені програмою DESIGN/IDEF компілюються і повідомляють про помилки.

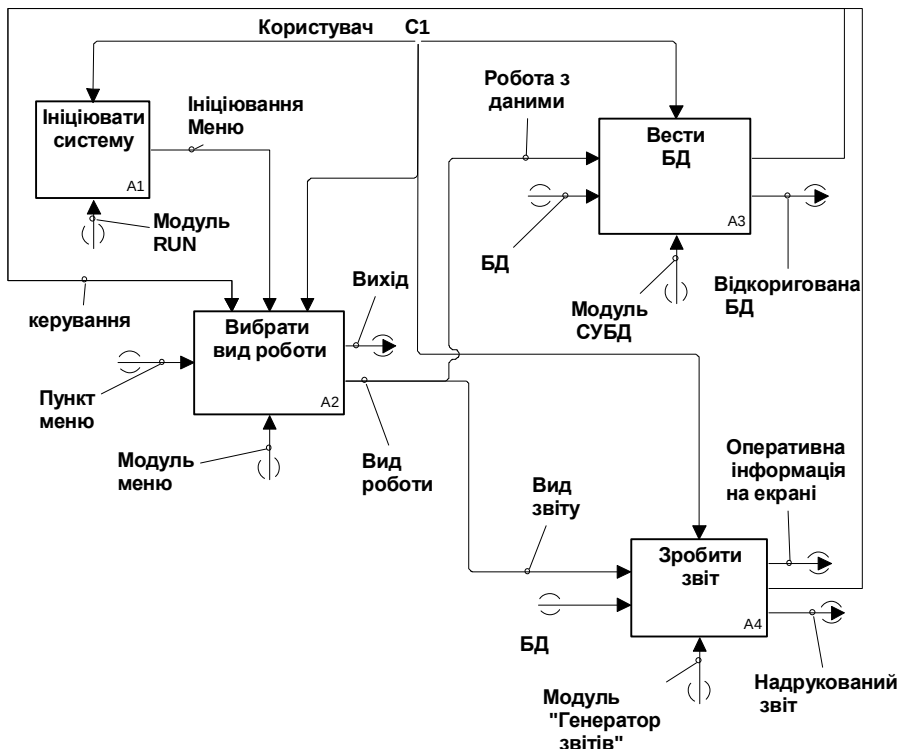


Рисунок В.2 - Приклад діаграми декомпозиції A0: Вести облік наукової роботи

Функції діаграми декомпозиції A0 (модулі) можуть в свою чергу деталізуватися на нижчому рівні на функції процедур модуля. Такі діаграми можна застосовувати для розробки функціональної структури модулів.

Зв'язок діаграм моделі за ідентифікаторами вузлів та іменами функцій діаграми дозволяє розглядати функціональну модель, як єдине ціле, не тільки за змістом, а й формально.

Ієрархічна структура з діаграм спрощує розуміння змісту моделі. В основі побудови ієрархічної структури діаграм функціональної моделі - IDEF0 лежать принципи абстракції та модульності.

### **В.3 Семантичні критерії якості функціональних моделей**

Для аналізу моделі структури програми використовують семантичні критерії якості функціональних моделей: вірність, повнота, стислість, узгодженість, складність/зрозумілість. Такий аналіз забезпечує зрозумілість, надійність, модифікування, ефективність роботи програмної системи.

#### **В.3.1 Типи семантичних критеріїв**

Оцінка якості змісту моделі виконується за 5 критеріями: повнота; стислість; узгодженість; вірність; складність/зрозумілість.

*Повнота* – достатність інформації в моделі для досягнення мети створення моделі. Повнота діаграми – достатність інформації діаграми для контексту діаграми.

*Вірність* – ступінь, з яким інформація про вміст діаграми або моделі точно описує систему, що моделюється. Вірність найбільше суб'єктивний критерій.

*Стислість* – назви функцій і мітки дуг стислі, якщо точно передають зміст, і при цьому не мають другорядної інформації. Терміни з тексту повинні бути зрозумілими програмісту.

*Узгодженість* – ступінь, з яким зображення всієї системи узгоджується із зображенням певного модуля. Також узгодженість – одноманітність символіки та термінології у модулі або в системі.

*Складність/зрозумілість* – між складністю та зрозумілістю існує зворотний зв'язок. Чим більш складна модель, тим менше її розуміють. Зрозумілість - ступінь, з яким ясно, чи досягається мета діаграми(моделі). Або ступінь, з яким можна дати відповідь на питання "про що розповідає діаграма(модель) ?".

Ясно, що відповідність моделі реальній системі (вірність та повнота, зрозумілість) залежать від складу функцій діаграм, стислості, вірності та узгодженості між собою їх назв. Але не меншу роль відіграють і їх дуги, як вони пов'язують між собою функції та як вони розташовані на діаграмах.

### **В.3.2 Зв'язок функцій діаграми дугами**

На діаграмах декомпозиції не зображуються алгоритми, але можливий порядок виконання функцій можуть показувати дуги, що з'єднують функції. Одна функція може працювати у різних режимах. Кожен режим вимагає наявності своїх дуг. Тому функція може виконуватися тільки при наявності всіх дуг необхідних для її роботи в певному режимі.

Таким чином порядок виконання функцій діаграми диктується наявністю дуг в певний момент роботи системи, тобто вірність, повнота та зрозумілість моделі залежать від наявності вірних дуг та узгоджених та стислих їх міток.

З рисунку В.2 (діаграма декомпозиції) системи «Вести облік наукової роботи» видно, що предикати модуля «Меню» не виконуються раніше, ніж виконається предикат модуля «RUN». Але виконання предикатів модуля «Вести БД» чи модуля «Зробити звіт» залежить від вибору користувачем пункту меню.

Іноді про порядок виконання функцій взагалі нічого не можна сказати. Якщо система програмний комплекс, то функції системи діаграми А0 подають програми системи, кожна з яких працює на своїй виробничій ділянці. Такі функції можуть працювати паралельно, послідовно або зовсім не можна сказати нічого про порядок їх виконання.

Діаграми можуть містити зворотні зв'язки при помилках у роботі функцій та для зображення ітерацій. При ітераціях дії виконуються повторно. На рисунку В.2 показано приклад зворотного керуючого зв'язку «Керування». Зворотній керуючий зв'язок подається керуючою дугою зверху. Зворотній зв'язок при накопиченні даних подається знизу.

### **В.3.3 Зчеплення модулів програми**

Зрозумілість системи залежить від характеристики зчеплення модулів. Зчеплення – ступінь залежності вмісту модулів між собою або ступінь пов'язування модулів даними. Зчеплення між модулями бувають двох типів: за природою зв'язку, за структурою зв'язку. Чим менше зчеплення модулів тим краще система.

### **В.3.3.1 Зчеплення модулів за природою зв'язку**

Зчеплення модулів за природою зв'язку.

*Зчеплення по кодам* – ступінь залежності вмісту модулів між собою. Зміст процедур модуля 2 тісно пов'язаний із змістом процедур Модуля 1. Зчеплення можливо, коли Модуль 1 «добре знає» структуру Модуля 2. Це означає, що у Модулі 1 запрограмована частково структура Модуля 2. Якщо у модулі 1 потрібно виконувати певні дії кілька раз, створюйте цю процедуру у модулі 1. У випадку необхідності створити універсальну процедуру, яка буде застосована у кількох модулях, створюйте її не орієнтовану на зміст модулів, а загального характеру.

*Керування.* Модуль 1 керує функціонуванням Модуля 2, надсилаючи керуючі дані. Керування може відбуватися через прапорці, перемикачі. Для цього Модуль 1 повинен «добре знати» зміст процедур Модуля 2. Написання таких глобальних процедур хибне. У Пролозі режим роботи процедури виявляється за конкретизацією даних.

*Нормальне сполучання.* Зчеплення модулів тільки через інтерфейси модулів. Всі вхідні та вихідні параметри модуля дані предметної області, що не містять керування.

### **В.3.3.2 Зчеплення за структурою зв'язку**

Зчеплення за структурою зв'язку – це ступінь пов'язування модулів даними.

*Зчеплення за середовищем,* де середовище спільна область даних. Модулі корегують одну глобальну структуру даних (наприклад файл або глобальні факти) або один модуль має доступ до даних іншого модуля. Працюючи з однією спільною областю, модуль може псувати дані іншого модуля.

*Зчеплення за записом.* Зв'язок між модулями встановлюється через передачу параметрів складного типу. Обидва модуля знають структуру складного даного. При корегуванні запису корегуються обидва модулі.

*Абстрактне зчеплення.* Зв'язок між модулями встановлюється через передачу параметрів простого типу. Реалізація модулів програми за нормальним та абстрактними зчепленнями робить модулі слабо зчепленими. Незалежні за змістом модулі можна перероблювати не торкаючи інших модулів. Повністю незалежні модулі не існують.

Треба намагатися утворювати *слабо зчеплені* модулі. Чим більше зчеплені модулі, тим менше зрозуміла система, і як наслідок, тим більше помилок буде зроблено при її розробці, модифікуванні.

### **В.3.4 Змістова зв'язність процедур модулів програми**

*Зв'язність* – ступень об'єднаності процедур модулів за змістом.

Чим більше пов'язані за змістом процедури модуля, тим більш зрозумілий вміст модуля. Чим більше взаємозалежність частин модуля, тим міцніше модуль і краще якість ПЗ.

Міцність модуля впливає на його розуміння, надійність, на можливість модифікації системи.

До хороших типів зв'язності відносять:

#### *1 Випадкова зв'язність*

У модулі між процедурами та функціями немає явних змістових зв'язків, тому інтерфейс модуля містить майже всі параметри процедур. Порушено принцип «чорного ящика». При роботі такого модуля буде багато помилок, модифікування його важке.

#### *2 Логічна зв'язність*

Модуль складається з процедур, які відносяться до одного класу або за класом вхідних або вихідних даних, а не за змістом. Майже завжди з такого модуля кожен інший модуль використовує тільки одну процедуру.

Наприклад: модуль «обрати тест за характеристикою людини» може мати різні процедури тестування, що характеризують людину за різними властивостями. Різні модулі програми використовують один інтерфейс модуля тестування з великою кількістю параметрів. Логічна зв'язність природна тільки для великих модулів. У цьому випадку

кілька модулів програми може використовувати з модуля багато процедур.

### *3 Часова зв'язність*

Процедури модуля виконуються в один конкретний період роботи системи. Між процедурами ніякого зв'язку за змістом немає. Але, процедури тісно пов'язані з іншими модулями системи. Процедури з часовою зв'язністю мають патологічний зв'язок з іншими модулями.

Наприклад. Система «Експерт» має наступні модулі: ініціалізації системи; інтерфейсу користувача; розуміння тексту природною мовою; адміністратора; експертизи. Процедури модуля ініціалізації наступні: зробити початкові установки для модуля інтерфейсу; ініціювати модулі адміністратора та розуміння тексту значеннями по замовчанню. Ясно, що всі початкові установки для модулів свої. Процедури модуля ініціалізації пов'язані тільки часом виконання системи – входом у систему. Такі процедури можуть мати дуже різні параметри, що порушує принцип захованості інформації.

### *4 Процедурна зв'язність*

Функції модуля групуються в модулі тому, що формально беруть участь в одному процесі або циклі.

Нехай є два файли: файл середніх зарплат службовців за кожний місяць і файл з кількістю виробів, що випускають за зміну різні цехи. Модуль знаходження максимумів повинен містити процедуру знаходження максимальної середньої зарплати службовця за рік та процедуру знаходження максимальної кількості виробів за зміну цехами.

Залежності за даними між процедурами немає. Якщо зробити процедури окремо, то зв'язність модуля буде логічною. Програміст вирішив оптимізувати код, включив знаходження максимумів для кожного списку в один цикл тому, що кількість цехів і кількість місяців у році однакові. Кількість цехів стабільна. Операції циклу виконуються однакові, але для різних даних. Якщо побудується новий цех, то програма буде видавати невірні дані. Таку помилку важко знайти.

Програміст, що бажає використати спільний код для кількох процедур, робить структуру модуля заплутаною і створює джерело помилок.

*Подані вище типи зв'язності не відносяться до змістових типів зв'язностей. Наступні типи зв'язностей змістові.*

#### *5 Комунікативна зв'язність*

Процедури модуля працюють з одним даним. Прикладом комунікативної зв'язності може бути модуль обробки рядків. Порядок виконання процедур обробки рядка залежить від виклику. Модулі з комунікативною зв'язністю використовуються в усіх мовах програмування.

#### *6 Послідовна зв'язність*

Процедури модуля утворюють конвеєр, на якому послідовно оброблюються певні дані. Результати першої процедури стають вхідними даними другої процедури. Далі дії повторюються. Кажуть, що такий тип зв'язності причинно-наслідковий.

#### *7 Функціональна зв'язність*

Всі процедури модуля готують дані для функціонування однієї функції модуля, що повертає розв'язок задачі. Модулі з функціональною залежністю легше всього використовувати.

## ДОДАТОК Д

### МОДУЛЬНЕ ПРОГРАМУВАННЯ МОВОЮ VISUAL PROLOG

#### Д.1 Структура модульної програми мовою Visual Prolog

Програма з кількох модулів має один головний модуль і кілька допоміжних модулів. Модулі програми взаємодіють через виклики глобальних процедур і фактів.

Всі модулі програми оформлюються подібно програмі з одного модуля, тобто мають секції опису предикатів і доменів(типів даних) і секції тверджень.

Головний модуль має секцію Goal, а інші модулі не мають.

Крім того, головний модуль повинен мати глобальні об'яви всіх модулів програми, а інші модулі можуть мати тільки ті описи, що вони заносять в інтерфейс або застосовують.

В будь-якій програмі мовою Visual Prolog може бути лише одна неіменована секція Facts. Тому, якщо є одна глобальна неіменована секція Facts, то в будь-якому модулі можуть бути лише іменовані локальні секції Facts. Глобальна неіменована секція розміщується в головному модулі програми.

Порядок об'явлення глобальних доменів і глобальних предикатів у різних модулях може бути різним. Такий підхід дозволяє створювати бібліотеки в об'єктних кодах з процедурами, що застосовує вся програма.

Кожен модуль має область інтерфейсу і область реалізації.

#### *Область реалізації*

Область реалізації містить локальні описи доменів, предикатів і локальні твердження, що записуються на базі локальних описів. Локальні описи і твердження можуть застосовуватися тільки в тому модулі, в якому вони знаходяться. З локальних тверджень складаються локальні процедури модуля.

### *Область інтерфейсу*

Область інтерфейсу містить глобальні описи доменів і описи глобальних предикатів. Кожна глобальна секція починається словом Global.

Глобальні секції Predicates і Facts містять опис предикатів, якими записані твердження глобальних процедур і глобальні динамічні факти. Такі процедури і факти, можна викликати з інших модулів.

Глобальні факти з характеристикою single можуть розміщуватися тільки в головному модулі. Причому секція Goal повинна розміщуватися перед секцією Clauses.

Намагайтеся не застосовувати глобальні факти, краще застосовувати глобальні предикати, що працюють з локальними фактами. Застосування глобальних фактів є варіантом зчеплення модулів за середовищем.

Порядок розміщення локальних і глобальних секцій у модулях програми не важливий.

### **Д.2 Описи глобальних предикатів**

При об'яві глобального предикату для кожного, крім характеристики предикату та типів аргументів, вказують конкретизацію його аргументів. Наприклад, якщо користувач напише свій предикат str\_len, його об'ява буде мати вид:

Global Predicates

Nondeterm str\_len (string, integer) - (i, o) (i, i) (o, i)

Якщо конкретизація параметрів для режимів роботи предикату не вказана, то по замовчанню всі параметри процедури вважаються вхідними.

Глобальною може бути предикат - функція. При об'яві предикату - функції перед функтором предикату вказують його тип. Тип функтору – це тип результату роботи функції.

Наприклад.

Global Predicates

Nondeterm integer kvadrat (integer) - (i)

З Прологу можна викликати програму написану на іншій мові програмування. Таку програму об'являють, як зовнішній предикат. Об'ява зовнішнього предикату повинна містити мову. Наприклад, для використання програми на мові С, треба об'явити:

```
Global Predicates
pro_c (integer, integer) - (i, o) language c
```

### Д.3 Приклад модульної програми

Розглянемо приклад модульної програми. Програма обчислює вік людини. Вона містить головний модуль і два допоміжних. Модулі програми зовсім малі. Це допоможе краще зрозуміти її структуру.

Головний модуль «P1» видає повідомлення, що буде обчислено вік людини. Модуль «P11» пропонує ввести поточний рік і вводить його. Модуль «P12» пропонує ввести рік народження і вводить його.

Результати роботи допоміжних модулів повертаються в головний модуль після чого головний модуль обчислює вік людини і виводить його.

Для створення модульної програми ми відокремимо інтерфейси (глобальні секції) модулів від локальних секцій модулів. Для глобальних предикатів вкажемо можливу конкретизацію аргументів. Нижче подано текст програми.

```
/*Головний модуль P*/
Include "P.inc"
Global domains
рік_п = integer
рік_н = integer
Global predicates
рік_поточний (рік_п) - (o)
рік_народження (рік_н) - (o)

Predicates
Do

Goal
Do.
```

Clauses

Do:-write ("Підрахуємо ваш вік "), nl, рік\_народження (Рік\_н),  
рік\_поточний(Рік\_п), Вік= Рік\_п - Рік\_н,  
Write ("Ваш вік ", Вік).

/\*Модуль P1\*/

Include "P.inc"

Global domains

рік\_п = integer

Global Predicates

рік\_поточний (рік\_п) - (o)

Clauses

рік\_поточний (Рік\_п):- write ("Введіть поточний рік"), nl,  
readint(Рік\_п).

/\*Модуль P2\*/

Include "P.inc"

Global domains

рік\_н = integer

Global predicates

рік\_народження (рік\_н) - (o)

Clauses

рік\_народження (Рік\_н):- write ("Введіть рік народження "),nl,  
readint(Рік\_н).

Модульну програму виконують за опцією RUN.

#### **Д.4 Створення модульної програми експертом застосування**

Для реалізації модульної програми треба створити проект. Для чого застосовують опцію оболонки Visual Prolog - Project. Проект створюється під керівництвом *експерта застосувань*.

Експерт застосувань реалізує автоматично ряд дій, які вимагають глибинних знань від програміста при створенні програми. Для реалізації модульної програми треба знати *засоби опису*

*структури програми з модулів.* Експерт застосувань дозволить легко, наглядно створити модульну програму і переглянути її структуру.

Після створення проекту і головного модуля за опцією NEW створюються допоміжні модулі і додаються до структури програми.

### *Опція new*

Опція NEW відкриває каталог проекту. Пропонується ввести ім'я модуля для створення файлу у каталозі проекту. В файлі програміст буде зберігати допоміжний модуль програми на вихідній мові. У вікні експерту застосувань до структури проекту додається модуль з вказаним ім'ям.

По замовчанню кожен модуль містить директиву Include «<ім'я головного модуля>.inc». Експерт застосувань опитує у діалоговому вікні File Inclusions for Module, які заголовочні файли з глобальними об'явами необхідно створити для поточного модуля і чи включати ці файли у файл «<ім'я головного модуля>.inc».

Якщо користувач ще не знає, які глобальні предикати і домени буде містити модуль, то можна відмовитися від створення файлів. До діалогового вікна можна повернутися після написання модуля за опцією Attributes у вікні експерта застосування. Опції Edit і Delete застосовуються для корегування або видалення обраного файлу з глобальними об'явами.

Діалогове вікно містить три групи опцій:

- «Create Heard Files». Група створення файлів для глобальних об'яв поточного модуля;

- «Include Heard Files in». Група включення директив «Include» у файл «<ім'я головного модуля>.inc» - За директивами компілятор включить вміст заголовочних файлів;

- «Include Heard Files in *ім'я головного модуля.pro*» група включення довідки, констант та предикатів графіки у поточний модуль програми. Остання група застосовується для візуального інтерфейсу.

На рисунку Д.1 показано діалогове вікно для додаткового модуля P1 програми, яка обчислює вік людини.

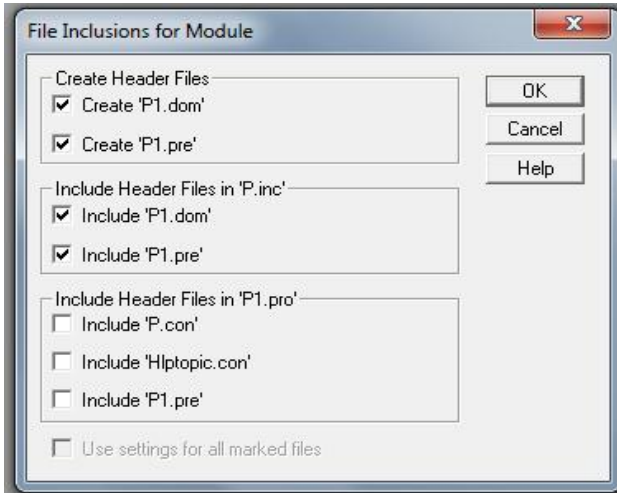


Рисунок Д.1- File Inclusions for Module

За першою групою опцій Експерт застосувань створює для глобальних доменів і предикатів кожного модуля порожні файли, а програміст заповнює їх або корегує заповнені.

Файли модулів P1 і P2 будуть містити:

```
/*Project: P Filename: p1.DOM*/
```

```
Global domains
```

```
рік_пот = integer
```

```
/* Project: P Filename: p1.PRE */
```

```
Global Predicates
```

```
рік_поточний (рік_пот)-(o)
```

```
/* Project: P Filename: p2.DOM */
```

```
Global domains
```

```
рік_нар = integer
```

```
/*Project: P Filename: p2.PRE*/
```

```
Global Predicates
```

```
рік_народження (рік_нар)-(o)
```

За другою групою опцій експерт включає в файл P.inc директиви:

```
Include "p1.dom"
Include "p1.pre"
Include "p2.dom"
Include "p2.pre"
```

Компілятор перед процесорній обробці включає вміст файлів з глобальними об'явами у файл P.inc, а за директивою «Include P.inc» включає вміст файлу P.inc у всі модулі.

```
Include /* Project: P Filename: P.INC*/
Global domains
DB_SELECTOR = browselist_db % For treebrowser tool
FILE = fileselector1; fileselector2 % To be edited
Include "p11.dom"
Include "p11.pre"
Include "p12.dom"
Include "p12.pre"
```

В результаті всі глобальні домени та глобальні предикати програми включаються у кожний модуль програми. Такий спосіб зручний для створення не дуже великих програм.

## **Д.5 Команди компіляції та побудови програми**

*Команда* PROJECT/COMPILE MODULE (Ctrl+F9) компілює обраний модуль.

*Команда* PROJECT/BUILD (Alt+F9) компілює тільки модулі, що були змінені та будує програму.

*Команда* PROJECT/REBUILD ALL (Ctrl+Alt+F9) компілює всі модулі і будує програму.

*Команда* PROJECT/LINK ONLY (Shift+F9) команда не компілює модулі. Вона будує програму для виконання.

*Команда* PROJECT/RUN (F9)

Команда компілює модулі, що були змінені, будує програму і запускає програму на виконання.

*Команда* PROJECT/ PROJECT TREE (Ctrl+T)

Після компіляції проекту можна подивитися дерево проекту. Щоб одержати текст модуля у вікні достатньо натиснути двічі лівою кнопкою миші на файлі.

## ДОДАТОК Е

### ОБ'ЄКТНА – ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ МООВОЮ VISUAL PROLOG

#### Е.1 Класи та об'єкти

Засоби об'єктна-орієнтованої мови дозволяють будувати програму, що складається з об'єктів певного типу і зв'язків між об'єктами. Така програма моделює об'єкти предметної області та статичні зв'язки між ними.

Домен – це множина об'єктів певного типу. Опис об'єктів одного домену називають класом, а ідентифікатор класу є типом об'єкта.

Клас описує об'єкт, як дані і процедури для роботи з даними. Глобальні процедури дозволяють об'єктам інших класів звертатися до об'єкту. Глобальні процедури називають методами. Глобальні дані можуть бути джерелом помилок, але дані можна також об'являти глобальними. В програмі до класу може відноситися будь-яка кількість об'єктів. Клас містить секцію інтерфейсу і секцію реалізації.

#### *Секція інтерфейсу класу*

У інтерфейсі класу визначають глобальні домени і глобальні предикати, глобальні факти. Інтерфейс класу обмежується ключовими словами Class і Endclass. Інтерфейс подається у формі:

```
Class ім'я_класу : [батьківський клас1, батьківський клас2, ...]
Domains /*секція глобальних доменів*/
Predicates /*секція предикатів для глобальних процедур*/
Facts /*секція предикатів для глобальних фактів*/
Endclass [ім'я класу]
```

Під батьківськими класами розуміють класи, від яких клас успадковує домени, предикати, факти. Клас, що успадковує предикати і домени, називають похідним класом.

В Domains об'являють глобальні домени. В Predicates об'являють глобальні предикати для статичних або нестатичних тверджень. Глобальні предикати об'являються як при модульному

програмуванні з можливими конкретизаціями. Характеристика `static` перед назвами секцій `predicates` і `facts` вказує, що твердження цієї секції існують тільки в одному екземплярі і належать класу, а не об'єктам. Статичні твердження можна застосовувати там, де бачиться його клас.

По замовчанню доступ до глобальних процедур, фактів і до об'єктів, які визначаються глобальними доменами, вільний для всього проекту (`Public`). Глобальні процедури, факти і домени називають відкритими. Глобальні предикати в секції `Facts` застосовують для запису динамічних фактів, доступних об'єктам інших класів.

### *Секція реалізації класу*

Секція реалізації класу обмежується ключовими словами `Implement` і `Endclass`. Термін `Implement` означає «реалізувати».

`Implement ім'я_класу:[список батьківських класів]`

`Domains /*секція локальних доменів*/`

`Predicates /*секція предикатів для локальних процедур*/`

`Facts /*секція предикатів для локальних динамічних фактів*/`

`Clauses`

Процедури і факти, предикати і домени яких об'явлені в секціях інтерфейсу або реалізації.

`Endclass [ім'я класу]`

У секціях `Class` і `Implement` можна розташувати кілька локальних секцій `Domains`, `Predicates`, `Facts` і одну секцію `Clauses`.

Можуть бути секції `static predicates` і `predicates`. Також може бути об'явлено кілька секцій `Facts` з різними іменами.

В секції `Clauses` розташовані глобальні та локальні процедури і динамічні факти необхідні для роботи об'єкту. Область їх видимості визначається секціями інтерфейсу і реалізації. Секції інтерфейсів різних класів можна групувати, тобто можна спочатку розташувати всі інтерфейси класів, а потім всі секції реалізації. Важливо, щоб те що застосовується було описано раніше, тобто клас описаний раніше, ніж секція реалізації.

## Е.2 Створення об'єкту та робота з ним

Розглянемо приклад програми, що містить один клас, який породжує один об'єкт. Програма одержує з фактів відомості про студентів і виводить інформацію про них.

```

Class s
Predicates
Nondeterm do()
Endclass s

Implement s
Predicates
Nondeterm stud (string, integer)

Clauses
Stud("Петренко",76).
Stud ("Кисіль",75).
Do ():-stud (F,S), write("Прізвище",F,"\n","Середній бал",S), nl,
fail.

Do ().
Endclass s

Goal
R1=s::New, R1:Do(),R1:Delete().

```

Для створення об'єкту застосовується конструктор *New()*. Виклик конструктору *New* подається:

Клас:: *New()*

Конструктор *New* повертає посилку на створений об'єкт вказаного класу у динамічній оперативній пам'яті. Посилку зберігають у змінній для подальшої роботи з об'єктом.

Об'єкту можна надсилати повідомлення. Повідомлення - це виклик методу цього об'єкту (виклик процедури або функції). Аргументами процедури або функції можуть бути значення змінних, константи, посилки на певний об'єкт. Аргументи можуть бути відсутні.

У прикладі показано, як у цілі створюється об'єкт та викликається метод «Do» класу «a» для роботи із створеним об'єктом. Факти об'єкту локальні і належать тільки йому.

Деструктор Delete() вилучає об'єкт. В прикладі після роботи з об'єктом він вилучається.

### Е.3 Успадкування класів

Для похідного класу працює *механізм успадкування*. Тобто, методи (процедури) та дані батьківського класу можна застосовувати у об'єктах похідного класу.

Для успадкування треба виконання певної умови: «Методи та факти батьківського класу, що успадковуються, повинні знаходитися в інтерфейсі батьківського класу або в секції реалізації мати характеристику Public.».

Треба пам'ятати, що при успадкуванні створюється об'єкт тільки похідного класу, який успадковує предикати та домени батьківського класу та свого класу. Важливість успадкування в тому, що в ієрархії можна застосовувати багато разів один код, тобто створити *множину процедур*. Розглянемо приклад завдання, в якому застосуємо успадкування.

Нехай є відомості про результати сесії студента. Треба порахувати його середній бал.

```

Class середній_бал
Facts-s
Nondeterm student (string, string, integer)
Single sred (integer, integer)
Predicates
Nondeterm count(real)-(o)
Endclass середній_бал

Implement середній_бал
Clauses
sred (0, 0).
student ("Коваль", "ЛП", 82).
student ("Коваль", "ФП", 78).
Count(_):- student(_,_,S), sred (S1,N), N1=N+1, Sr = S+S1,
            assert (sred (Sr, N1)), fail.
Count (Sr):-sred (S, N1), Sr=S/N1.
```

```

Endclass середній_бал
Class вивід: середній_бал
Predicates
Nondeterm print_s()
Endclass вивід

Implement вивід
Clauses
print_s ():- count(Sr), student(F,_,_),
               Write (F," Середній бал ", Sr).
Endclass вивід

Goal
R1= вивід::new, R1: print_s (), R1: delete.

```

В програмі нам потрібно мати тільки об'єкт похідного класу, який виводить середній бал студента. Тому створюється тільки об'єкт класу «вивід». Об'єкт викликає предикат count з батьківського класу і підраховує середній бал. Після чого результат виводиться об'єктом на екран. Батьківський клас повинен розміщуватися у програмі вище ніж похідний.

#### Е.4 Абстрактні класи

Класи називають абстрактними, якщо вони не мають секції реалізації. Абстрактні класи визначаються як звичайні класи, але на початку визначення розміщується слово Abstract.

Наприклад: Abstract class.

Абстрактні класи застосовують для роботи в одному місці програми з об'єктами різних класів. Аналогом абстрактного класу є об'єкт узагальненого типу даних в логічній програмі:

```

Domains
tip = stud (string);
      grup (integer)
Goal
Readterm (tip, D), write (D).

```

Розглянемо приклад. Нехай треба одним предикатом вводить прізвище студента і кількість студентів в групі.

Створимо два класи «студент» і «група», що успадковують абстрактний клас «людина». Обидва класи можна віднести до одного домену з іменем абстрактного класу. Об'ява у програмі предикату з аргументом типу абстрактний клас дозволить передавати у предикат посилку на об'єкти будь-якого класу, що належить до типу абстрактного класу.

```
Abstract class a
```

```
Predicates
```

```
Read_pr ()
```

```
Endclass a
```

```
Class студент : a
```

```
Predicates
```

```
Read_pr ()
```

```
Endclass студент
```

```
Class група : a
```

```
Predicates
```

```
Read_pr ()
```

```
Endclass група
```

```
Implement студент
```

```
Facts-s
```

```
Nondeterm студент (string)
```

```
Clauses
```

```
Read_pr ():- write ("Введіть прізвище студента \n"), readln (Pr),  
assert (студент (Pr), s).
```

```
Endclass студент
```

```
Implement група
```

```
Facts-v
```

```
Nondeterm група (integer)
```

```
Clauses
```

```
Read_pr ():-write ("Введіть кількість студентів у групі \n"),  
readint (Pr), assert (група (Pr), v).
```

```
Endclass група
```

Predicates

Rd (a)

Clauses

Rd (L):- L: Read\_pr.

Goal

S = студент::new (),

V = група::new (),

Rd (S),

Rd (V),

S: delete (), V: delete ().

Програма містить фрагмент написаний чисто логічним стилем, він не відноситься до жодного класу. Предикат RD має своїм аргументом посилку не на об'єкт певного класу, а на узагальнений об'єкт абстрактного класу. Це дозволяє викликати метод RD з посилками різних класів Rd (S), Rd (V).