МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ
до виконання самостійної роботи
з дисципліни
“Моделювання та аналіз програмного забезпечення”
для студентів
спеціальності 6.05010301
“Програмне забезпечення систем”

усіх форм навчання

2017

Методичні вказівки до виконання самостійної роботи з дисципліни “Моделювання та аналіз програмного забезпечення” для студентів спеціальності 6.05010301“Програмне забезпечення систем”, усіх форм навчання /Укл.: С.М. Сердюк, Ж.К. Камінська. – Запоріжжя: ЗНТУ, 2017. – 47с.

Укладачі: С. М. Сердюк, доцент, к.т.н.,
Ж.К. Камінська, асистент

Рецензент: С.К. Корнієнко, доцент, к.т.н.

Відповідальний
за випуск: С.М. Сердюк, доцент, к.т.н.,

Затверджено
на засіданні кафедри
“Програмних засобів”

Протокол № ____
від _____ 2017

ЗМІСТ

ВСТУП

Самостійна робота студентів – це спланована пізнавальна, організаційно і методично направлена на досягнення результату діяльність студента, яка здійснюється без прямої допомоги викладача. Вона є основним способом оволодіння студентами навчальним матеріалом у час, вільний від обов'язкових аудиторних занять [1].

Мета виконання самостійної роботи – поглиблення, узагальнення і закріплення теоретичних знань і практичних умінь студентів з дисципліни, що вивчається, шляхом вироблення вміння самостійної роботи з навчальною і фаховою науково-технічною літературою.

Завдання самостійної роботи студентів:

- розвиток творчих здібностей та активізація розумової діяльності студентів;
- формування у студентів потреби безперервного самостійного поповнення знань;
- розвиток морально-вольових зусиль студентів;
- самостійна робота студентів як результат їх морально-вольових зусиль;
- навчити студентів самостійно працювати з літературою;
- навчити студентів творчо сприймати навчальний матеріал і осмислювати його;
- •сформувати навички щоденної самостійної роботи з метою одержання та узагальнення знань, умінь і навичок.

Форми самостійної роботи студентів:

- підготовка до лекцій і лабораторних занять;
- опрацювання теоретичних основ прослуханого лекційного матеріалу;
- вивчення окремих тем або питань, що передбачені для самостійного опрацювання;
- опрацювання навчальної та методичної літератури;
- опрацювання наукової літератури та періодичних видань;
- виконання розрахунково-графічного завдання,
- написання рефератів,
- контрольної роботи – для студентів заочної форми навчання.

Зміст самостійної роботи студентів з дисципліни визначається навчальною програмою дисципліни та робочою навчальною програмою вивчення дисципліни. На самостійну роботу виноситься: частина теоретичного матеріалу, менш складного за змістом, окремі практичні завдання та роботи, що не потребують безпосереднього керівництва викладача.

Складовими самостійної роботи студента є обов'язкова та вибіркова частини. Обов'язкова складова передбачає опанування програмного матеріалу дисципліни. Вибіркова складова передбачає виконання завдань, які студент вибирає з метою підвищення свого професійного рівня, особистого рейтингу. Як правило, до таких завдань відноситься робота науково-дослідницького і творчого характеру.

Навчально-методичні засоби самостійної роботи студентів:

- основна література (підручник, конспект лекцій, навчальні та методичні посібники);
- додаткова література (наукова, фахова, періодична);
- методичні матеріали.

Місцями виконання самостійної роботи з дисципліни є бібліотека, навчальні кабінети, комп'ютерні класи та лабораторії університету, а також домівка студента.

При використанні студентами складного обладнання чи устаткування, складних систем доступу до інформації передбачаються можливості отримання необхідної консультації або допомоги з боку викладача.

Вимоги до самостійної роботи студента:

– робота має бути виконана особисто студентом або групою студентів, де кожен її член самостійно виконує свою частку колективної роботи;

–робота повинна являти собою закінчену розробку (чи її етап), де розкриваються й аналізуються актуальні проблеми з певної теми або її окремих аспектів;

– робота має демонструвати достатню компетентність автора (авторів) у розкритті питань, що досліджуються;

– робота повинна мати навчальну, наукову й (або) практичну спрямованість і значимість, містити певні елементи новизни (при виконанні науководослідної роботи).

Оформлення звітів зі самостійної роботи студентів здійснюється відповідно до вимог, розроблених кафедрою, та інших нормативних документів, що стосуються виконання та оформлення наукових, навчально-методичних та інших робіт.

Керівництво самостійною роботою студентів – це індивідуально-консультативна робота – це форма організації навчальної роботи викладача зі студентами, яка здійснюється шляхом створення необхідних умов для виявлення і розвитку індивідуальних здібностей студента на основі особистіснодіяльнісного підходу. Вона проводиться з метою посилення мотивації студентів до пізнавальної діяльності і спрямування її в необхідному напрямку.

Індивідуально-консультативна робота, як правило, проводиться у вигляді консультацій, інколи – індивідуальних занять у формі: діалогу з різних навчальних проблем; перевірки виконання завдань; виконання індивідуальних завдань (курсових та дипломних проектів (робіт), розрахункових, творчих робіт тощо); евристичної бесіди; наукової роботи та ін.

Консультація – одна з форм організації навчального процесу, що проводиться з метою отримання студентом відповіді на окремі теоретичні чи практичні питання, пояснення певних теоретичних положень та їх практичного застосування. При цьому виділяють такі види консультацій: тематичні – проводяться за певними темами дисципліни або найбільш складними питаннями програмного матеріалу; цільові – використовуються перед проведенням модульної контрольної роботи або іншого виду поточного чи підсумкового контролю; активні – консультації з використанням активних методів навчання (наприклад, у формі прес-конференції); з самостійної роботи – проводяться при підведенні підсумків самостійної роботи.

Формами контролю студентів за якістю оволодіння навчальним матеріалом є: самоконтроль за допомогою контрольно-тестових завдань та контроль з боку викладача, який здійснюється за допомогою методів поточного і підсумкового контролю.

Методами поточного контролю є:

- усне опитування студентів на лабораторних заняттях;
- перевірка практичних завдань, виконаних студентами індивідуально;
- перевірка рефератів та організація їх презентацій;
- співбесіда.

Формами підсумкового контролю є:

- проведення контрольних робіт в аудиторії (за тематичними модулями);
- екзамен (залік) за передбаченими програмою курсу питаннями та практичними завданнями.

Облік успішності студентів з виконання самостійної роботи здійснюють викладачі у журналах обліку успішності.

1 ЗАГАЛЬНІ ВІДОМОСТІ ПРО ДИСЦИПЛІНУ

Метою даної дисципліни є вивчення теоретичних та методологічних основ моделювання програмного забезпечення, набуття практичних навичок імітаційного моделювання складних систем, застосування мови моделювання UML та відповідного об'єктно-орієнтованого підходу до моделювання програмного забезпечення.

Внаслідок вивчення дисципліни студенти повинні:

• вивчити методи моделювання складних обчислювальних систем і приладів;

• придбати практичні навички та досвід побудови імітаційних моделей складних систем;

• вивчити елементи теорії і практики моделювання програмного забезпечення з використанням мови UML та відповідного об'єктно-орієнтованого підходу.

На основі вивчення дисципліни "Моделювання програмного забезпечення" студенти повинні знати:

• основні ідеї та методи аналізу та синтезу складних систем, у тому числі комп'ютерних інформаційних систем, сучасні системи імітаційного моделювання комп'ютерних систем;

• етапи життєвого циклу програмного забезпечення;

• методи моделювання складних інформаційних систем;

• засоби мови UML для моделювання програмного забезпечення;

• мову специфікації формальної семантики Object Constraint Language (OCL);

• найбільш відомі об'єктно-орієнтовані CASE-інструменти.

вміти:

• проводити системний аналіз предметної області;

• виконувати розробку математичних моделей складних систем;

• розробляти, відлагоджувати програми імітаційних моделей складних систем та кваліфіковано аналізувати результати моделювання для синтезу нових систем, чи покращення існуючих;

• виконувати об'єктно-орієнтовану декомпозицію та об'єктно-орієнтований аналіз предметної області, використовуючи ієрархію класів, діаграми процесів та об'єктів;

• розробляти проект програмної системи на базі технології об'єктно-орієнтованого програмування, використовуючи відповідні діаграми мови UML.

2 ТЕОРЕТИЧНІ ПИТАННЯ З ДИСЦИПЛІНИ

2.1 Методологія системного аналізу

Принципи системного підходу. Поняття елементів, зв'язків, системи. Структури, їх види та форми представлення. Протиріччя, що виникають при описанні систем. Стратифікований підхід до моделювання систем. Недоліки стратифікованого підходу до моделювання систем Месаровича. Узагальнений підхід до структуризації ерготехнічних систем Губінського. Множина структур необхідних для проведення системного аналізу систем.

Література / 1, стор.11-16; 6, стор.14-35; 7, стор.37-41, 65-67/

2.2 Призначення та види мов моделювання

Необхідність класифікації моделей. Поняття детермінованих та стохастичних моделей. Статичні, динамічні, дискретні, дискретно-безперервні та безперервні види моделювання. Основні характеристики видів моделювання. Необхідність та переваги мов моделювання.

Література /1, стор.21-28; 2, стор. 76-89/

2.3 Дискретні системи моделювання

Узагальнена структура та функціонування систем моделювання: керуюча програма моделювання, підпрограма початку імітації, підпрограма збору статистики, підпрограма закінчення імітації. Організація квазіпаралелізму в моделях засобами розгляду активності. Особливості активностних, транзактних, агрегатних, процесно-орієнтованих способів моделювання: умови застосування, схеми моделей об'єктів. Порівняння способів організації квазіпаралелізму в моделях.

Література /3, стор.14-36/

2.4 Технологія проектування імітаційних моделей

Етапи створення імітаційних моделей: змістовне описання об'єкту моделювання; розробка концептуальної моделі; перехід до формального опису; перетворення формальної мови в описання імітаційної моделі; програмування та відлагодження імітаційної моделі. Засоби підвищення рівня технології побудови моделей.

Проведення імітаційного експерименту. Особливості аналізу результатів моделювання.

Література /3, стор.37-91; 6, стор. 23-56, 345-389; 7, стор.35-40, 57-64/

2.5 Уніфікований процес розробки програмного забезпечення

Основні поняття. Структура уніфікованого процесу розробки. Життєвий цикл програмного забезпечення та його критичні етапи. Управління процесом за допомогою "прецедентів використання" системи. Декомпозиція процесу на безліч робочих процесів. Фази процесу розробки системи. Артефакти. Учасники. Робочі процеси. Їх склад і призначення. Методологія структурного аналізу та проектування. Методологія функціонального моделювання. Методологія моделювання потоків даних.

Література / 4, стор. 10-18; 5, стор. 17-42, 202-218, 456-480; 6, стор. 63-66/

2.6 Методології структурного аналізу інформаційних систем

Сутність структурного підходу SADT: призначення, склад функціональної моделі, ієрархія діаграм. Група засобів структурного аналізу та їх взаємини: методологія структурного аналізу та проектування SADT, діаграми потоків даних DFD, діаграми "сутність зв'язок" ERD. Діаграми потоків даних DFD: призначення, основні компоненти діаграм, побудова ієрархії діаграм DFD. Моделювання даних: модель "сутність зв'язок" ERD, метод Баркера. Приклад моделювання предметної області методом Баркера. Методологія IDEF: аналіз стандартів IDEF, сутність та основні об’єкти методологій IDEF1 та IDEF1X.

Література /4, стор.20-38, 62-79, 112-115; 5, стор.409-434, 8, стор. 263-273/ 63-66/

2.7 CASE-технології

Аналіз ринку об'єктно-орієнтованих CASE-систем. Принципи побудови та основні компоненти CASE-систем, що підтримують мову UML і Уніфікований процес розробки програмного забезпечення. Засоби автоматизації тестування. Метрики якості програмного

забезпечення. Класифікація CASE-засобів. Перетворення (рефакторинг) програм з метою поліпшення їх якісних характеристик. Інструментальні засоби для рефакторинга програм. Приклади використання CASE-систем.

Література /4, стор.37-91; 8, стор.18-62, 252-262/

2.8 Мова UML

Призначення та основні поняття мови UML. Історія створення мови UML і процес його стандартизації. Графічна нотація мови UML. Діаграми статичної структури, прецедентів, кооперації, послідовності, станів, діяльності та їх використання при моделюванні поведінки системи. Моделювання реалізації системи за допомогою діаграм компонент і розгортання. Моделювання на мові UML структур бібліотек класів. Представлення елементів нотації мови UML засобами мов програмування. Семантика мови UML:

призначення і структура метамоделі мови UML, склад, призначення і функціональність пакетів базових класів мови UML. Метамодель мови UML. Засоби розширення мови UML. Призначення, синтаксис і семантика мови OCL. Формалізований опис метамоделі мови UML за допомогою мови OCL. Стандартизація мови OCL.

Література /5, стор.42-197; 8, стор.77-158, 203-251/

3 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

3.1 Лабораторна робота №1. Методи створення моделей з використанням системи імітаційного моделювання SIMC.

Мета роботи: ознайомлення з принципами побудови імітаційних моделей систем масового обслуговування (СМО) з використанням системи імітаційного моделювання SIMC.

3.2 Лабораторна робота № 2. Моделювання систем масового обслуговування з одним обслуговуючим приладом та чергою

Мета - вивчення методів моделювання різних дисциплін обслуговування в СМО з одним приладом та чергою, аналіз вихідних даних моделювання з метою вибору оптимального варіанту реалізації системи яка моделюється.

3.3 Лабораторна робота № 3. Моделювання системи масового обслуговування зі зворотнім зв'язком. Моделювання багатоканальної системи масового обслуговування.

Мета роботи: вивчення методів моделювання СМО із зворотнім зв'язком та багатоканальних СМО на основі використання систем імітаційного моделювання SIMC.

3.4 Лабораторна робота № 4. Використання розподілу імовірності в системі імітаційного моделювання SIMC. Генератори випадкових чисел.

Мета роботи: вивчення методів описання рівномірного та нерівномірного розподілу безперервних та дискретних випадкових величин в SIMC та способів їх практичного використання при моделюванні СМО.

3.5 Лабораторна робота № 5. Моделювання довільних дисциплін обслуговування з використанням ланцюгів користувача. Вивчення принципів побудови гістограм.

Мета роботи: освоєння процедур графічного виводу інформації про розподіл випадкових величин та синхронізації подій в моделі (сигнали) в системі імітаційного моделювання SIMC. Вивчити

принципи моделювання довільних дисциплін обслуговування на основі списків користувач.

3.6 Лабораторна робота № 6. Діаграми класів.

Мета роботи: навчитися проектувати діаграми класів з використанням програмного середовища Rational Rose, зокрема, створювати класи, атрибути та операції класів, накладати зв’язки між класами, генерувати коди класів на мові програмування C++.

3.7 Лабораторна робота № 7. Діаграми послідовностей.

Мета роботи: навчитися проектувати діаграми послідовностей з використанням програмного середовища Rational Rose, зокрема, створювати об’єкти, налаштовувати час життя об’єктів, створювати повідомлення між об’єктами та встановлювати властивості повідомлень.

3.8 Лабораторна робота № 8. Діаграми кооперації.

Мета роботи: навчитися проектувати діаграми кооперації (ДК) з використанням програмного середовища Rational Rose: створювати об’єкти, накладати зв’язки між об’єктами, створювати повідомлення між об’єктами та встановлювати властивості повідомлень.

4 ТЕХНОЛОГІЯ ВИКОРИСТАННЯ СИСТЕМИ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ SIMC

4.1 Загальна класифікація

Для побудови моделей потрібні об’єкти, що володіють всілякими властивостями. У різних прикладних областях властивості, властиві моделюємим системам, різні, різний спосіб формалізації моделей і склад об'єктів, які входять у них. У цьому змісті ступінь універсальності систем моделювання можна характеризувати доступними в них об'єктами, а також властивостями об'єктів, реалізованих через визначені набори операцій.

У системі моделювання SIMC використовуються абстрактні набори об'єктів, конкретне семантичне призначення яких установлюється програмістом, програміст же реалізує логіку їхньої взаємодії, використовуючи визначені в системі й (якщо це необхідно) складені самостійно об’єктно-орієнтовані функції. Перш ніж перейти до розгляду окремих об'єктів, дамо їхню загальну класифікацію.

Об'єкти системи моделювання SIMC за часом існування в моделі й способу створення діляться на статичні й динамічні.

Статичні об'єкти представляються системними змінними постійно розміщеними в оперативній пам'яті. Вони необхідні увесь час роботи моделі. Прикладом такого об'єкта є *systime* - змінна дійсного типу, значення якої визначає модельний час.

Динамічні об'єкти створюються в міру необхідності. На час їхнього існування для них виділяється місце у вільній області оперативної пам'яті. Знищення такого об'єкта приводить до звільнення займаної пам'яті. Отже, знищення й створення динамічних об'єктів дають принципову можливість моделювання систем із числом об'єктів більшим, ніж при статичному розміщенні. Прикладом динамічних об'єктів можуть служити гістограми, існування яких необхідно тільки на час виміру деяких випадкових значень.

Об'єкти бувають переміщуваними й нерухомими. Переміщення об'єктів викликає здійснення подій у системі та зміну системного часу. Найважливішим типом переміщуваних об'єктів є транзакти, всі інші об'єкти можуть переміщуватися тільки будучи закріпленими за транзактами. Транзакт, як абстрактний об'єкт системи, може являти

собою деталь, яка оброблюється на потокової лінії, літак що прибуває до аеропорту або електричний імпульс, що включає виконавчий пристрій.

Іншими словами транзакт може собою представляти будь-який елемент однорідного потоку об'єктів у реальній системі. Як переміщувані об'єкти транзакти можуть затримуватися, групуватися, змінювати свої властивості або властивості їхнього навколишнього модельного середовища.

У системі моделювання об'єкти можна розглядати як скалярні й множинні.

Скалярні об'єкти характеризуються одним єдиним значенням, приміром, системний час (*systime*) і системна подія (*sysevent*), відповідно, дійсне й позитивне ціле число.

Множинні об'єкти характеризуються набором значень, приміром, транзакт має цілі дійсні й бульові параметри,крім того він несе в собі інформацію про чергову (для даного транзакта) подію що виконується й час його виконання, а також іншу інформацію. Гістограма також являє приклад множинного об'єкта, вона містить інформацію про число спостережень випробовуваної величини, про кількість показань спостережуваного значення і різні інтервали, а також іншу інформацію, яка використовується для підрахунку математичного очікування й дисперсії, а також для виводу гістограми в графічному виді.

Об'єкти бувають одиночні й групові.

Групові об'єкти поєднуються в сукупність по деякій ознаці. Основна форма існування групових об'єктів - списки.

Приміром, транзакти зв'язуються в списки залежно від того, якими груповими властивостями вони володіють.

Транзакти можуть заноситися у список:

delist - список незатребуваних моделлю транзактів;

current - список транзактів, готових до просування в теперішній момент часу;

userlist - списки, створювані користувачем для організації різних дисциплін обробки.

Крім зазначених є й інші списки, обговорення яких буде проводитися нижче. Помітимо, що в списки зв'язуються й інші об'єкти. Найчастіше дії над списками здійснюються автоматично.

Іншим типом групових об'єктів у системі SIMC є ансамблі. Ансамблі як сукупності транзактів можуть мати цілий набір групових властивостей, що допускають одночасну зміну для всіх членів

ансамблю. Приміром, група транзактів може рухатися по декількох можливих маршрутах у моделі. Окремі транзакти можуть перебувати в різних крапках моделі й бути приписаними до різних списків. Зміна номера маршруту у всіх членів ансамблю приведе до зміни шляху проходження у весь, утворюючий ансамбль транзактів.

Точне визначення об'єктів як типів даних у системі моделювання SIMC приводиться нижче. Дії над об'єктами розглядаються під час обговорення функцій.

4.2 Системні константи, типи й змінні

Безліч об'єктів визначена в системі SIMC шляхом введення абстрактних типів даних. Створення або визначення необхідної кількості об'єктів покладено на програміста, він також зобов'язаний передбачити дії над об'єктами при виникненні в системі певних умов.

Системні константи використовуються для настроювання системи моделювання на доступну при реалізації на конкретній ЕОМ пам'ять. Число системних констант дуже мале, але вони впливають на можливості системи.

Нижче визначаються наступні константи:

evemax - обмежує в системі максимальну кількість подій; кожна подія, як уже вказувалося, визначає послідовність дій (фактично алгоритм поводження транзакта при його переміщенні у відповідну події точку моделі). Чим більше число подій визначене в моделі, тим більш детальним є опис реальної системи. У цьому змісті evemax обмежує можливості SIMC по поданню реальних систем.

signmax - константа обмежує кількість сигналів у системі. Сигнали використаються для взаємодії між транзактами, що перебувають у різних частинах моделі. Сигнали можна передавати й приймати (якщо буде потреба очікуючи їхньої появи).

hint - константа, що задає максимальне число інтервалів у гістограмі. Властивість hint задає число точок на числовій осі, що задає інтервали для табулювання значень.

Наступні константи мають відношення до транзактів:

prtymax - максимальна абсолютна величина пріоритету транзакта. Пріоритети транзакта використовуються в багатьох випадках. Зокрема, якщо в деякий фіксований момент часу на переміщення по моделі претендують одночасно декілька транзактів,

то, як правило, рухатися буде той, котрий має вищий пріоритет, інші будуть чекати його зупинки.

Константи, що встановлюють число параметрів транзакта:

mptb число бульових параметрів:
mptr число дійсних параметрів:
mpti число цілочисельних параметрів:
mptf число покажчиків на прилади:
mptq число покажчиків на черги:
mpts число покажчиків на накопичувачі.

Докладне призначення констант дається у визначенні типів.

4.2.1 Системні типи

Введення абстрактних типів даних, що представлені нижче, покладено в основу формалізації процесу моделювання.

Пропоновані типи служать для створення системного середовища й використовуються користувачем для визначення необхідних йому об'єктів.

Помітимо, що в системі моделювання визначені як типи даних:

1) структури абстрактних об'єктів;

2) покажчики на ці структури:

Покажчики використовуються для визначення положення об'єкта, а сам об'єкт містить необхідні поля даних, значення яких використаються при моделюванні.

4.2.2 Скалярні типи даних

Для визначення змінних, що мають зміст часу, використовується тип ***double***.

При моделюванні час змінюється від нуля до деякої позитивної величини. Одиниця виміру часу в системі не визначається: отже інтерпретація часу в моделі задається тими одиницями часу, які визначив користувач.

Тип *prtyrange=-prtymax..prtymax* визначає діапазон зміни пріоритетів транзактів, надалі використовується для визначення нових типів даних.

Простор зміни системних подій визначено типом *event=1..evemax*.

Нумерація подій у моделі не повинна виходити за межі, задані типом *event*. Моделі, процеси й підмоделі займають зв'язну область простору подій.

Тип *enum parmtype {parmb,parmi,parmr,parmf,parmq,parms,last_parmtype}* специфікую типи параметрів транзакта:

parmb задає параметр бульового типу;
parmi задає параметр цілого типу;
parmr задає параметр дійсного типу;
parmf задає параметр типу покажчик на прилад;
parmq задає параметр типу покажчик на чергу;
parms задає параметр типу покажчик на накопичувач.

Тип *parmtype* використовується допоміжними функціями.

До числа скалярних типів також відносяться покажчики на деякі типи системних змінних. Їхній опис будемо приводити при визначенні відповідного типу.

4.2.3 Множинні типи даних

Для завдання імен у системі визначений тип даних *typedef array<1,8,**char**> alfa*.

Відзначимо, що в деяких функціях у якості параметрів можлива підстановка констант типу *alfa*. **Текстовий рядок-параметр типу *alfa* повинен містити число символів у точності рівне восьми!**

Перераховуємі нижче множинні типи даних будемо визначати за наступною схемою:

- призначення типу;
- специфікація типу в системі моделювання;
- покажчик на розглянутий тип (якщо такий є).

При описі полів об'єктів у розділі "специфікація" призначення поля дається коментарем. У коментарях символом *"U"* відзначені ті з них, які можуть змінюватися користувачем за його розсудом; символом *"S"* відзначені поля, використовуємі для системних цілей. Їхній уміст можна тільки читати. Зміна значень цих полів може привести до непередбачених наслідків.

ТРАНЗАКТ - абстрактний динамічний об'єкт, що характеризується рядом властивостей, що задаються значеннями

параметрів. У ході моделювання транзакти можуть створюватися, переміщатися по моделі й знищуватися.

struct *transact* { // транзакт

Параметри:

// Установлюється користувачем при створенні власних функцій для роботи зі списками користувача. Див. пр.

array<1,mptb,***bool***> pb; // бульові U

array<1,mpti,***int***> pi; // цілі U

array<1,mptr,***double***> pr; // дійсні U

array<1,mptf,*pfacility*> pf; // посилання на прилади U

array<1,mptq,*pqueue*> pq; // посилання на черзі U

array<1,mpts,*pstorage*> ps; // посилання на накопичувачі U

prtyrange prty; // пріоритет U

bool testprty; // ключ пріоритету U

// **Всі інші поля використовуються тільки системою !**

int ans, // номер ансамблю

nans, // кількість членів ансамблю

nom; // номер транзакта

ptransact predans, // посилання на попер., наст.

sledans, // транзакт в ансамблі

pred, // посилання на попер., наст.

sled; // транзакт у списку

event eve; // подія ініціалізації

double nexttime; // час активізації

plistt translist; // посилання на займаємий список };

typedef struct *transact* ptransact*; // посилання на транзакт

ПРИЛАД - динамічний об'єкт, призначений для моделювання обслуговуючого апарата. Транзакт може очікувати входу в прилад, займати або захоплювати прилад, звільняти прилад. По приладу збирається відповідна статистика.

struct *facility* { // прилад

bool test; // ознака включення U

alfa name // символьне ім'я U

// **Всі інші поля використовуються тільки системою!**

enum {free,seized,preempted}

status; // стан приладу

pfacility sled, // посилання на наст. і попер.

pred; // прилади в списку

ptransact transpoint; // посилання на обр. транзакт
int p, // число захватів
ci; // число входів
double timef, // часи: зайнятості,
pretime, // попереднього звернення
mtime, // службове поле
pro; // завантаження приладу
plistt fl, // посилання на списки очікуючих
inter; // та перерваних транзактів };
typedef struct *facility* pfacility*; // посилання на прилад

ЧЕРГА - динамічний об'єкт, призначений для реєстрації просування транзактів на певних ділянках моделі, збору й обробки відповідної статистики.

struct *queue* { // черга
bool test; // ознака включення U
alfa name; // символьне ім'я U
// **Всі інші поля використовуються тільки системою!**
enum {empty,full} status;// статус(порожня/непуста)
int lq, // довжина черги: поточна,
mq, // максимальна,
size; // гранична
pqueue sled, // посилання на наст., попередню
pred; // черги в списку черг
int ci, // число входів: загальне,
co; // у чергу нульової довжини
double timeq, // загальний час зайнятості
pretime, // час попер. звернення
lm, // середня довжина
mtime; // службове поле };
typedef struct *queue* pqueue*; // посилання на чергу

НАКОПИЧУВАЧ (багатоканальний пристрій) - динамічний об'єкт, призначений для моделювання одного або декількох обслуговуючих пристроїв. Транзакт може очікувати входу в накопичувач, займати в накопичувачі одну або кілька осередків, залишати накопичувач.

struct storage { // накопичувач
bool test; // ознака включення U

alfa name; // символьне ім'я U

// **Всі інші поля використовуються тільки системою!**

int s, // ємність накопичувача

ss, // поточний уміст

sf, // ємність, що залишається

sm, // макс. уміст

ci; // число входів

double ut, // завантаження накопичувача

smean, // середній уміст

mtime, // службове поле

times, // час: зайнятості,

pretime; // попер. звернення

pstorage pred, // посилання на попередній,

sled; // наст. накопичувачі

plistt slt; //посилання на список очікуючих транзактів };

typedef struct *storage* pstorage*; // посилання на накопичувач

Гістограми в системі імітаційного моделювання використовуються для збору й нагромадження статистичної інформації про будь-які скалярні, дійсні або цілі змінні. Всі необхідні гістограми задаються користувачем у частині визначень, створюються в розділі створення модельного середовища. Вимірювані значення табулюються в гістограмі в подійній частини моделі.

struct *histogram*{ // гістограма

// **Значення полів задаються користувачем при створенні гістограми**

bool graf; // ключ друку графіка

alfa name; // символьне ім'я

hint2 ihint; // число точок табулювання

double maxx, minx;

// **Всі інші поля використовуються тільки системою!**

unsigned total, // загальна кількість входів

sum,

sumsqr,

phistogram sled,

pred;

harr x;};

typedef struct *histogram* phistogram*; // посилання на гістограму

Таблиці в системі моделювання є зручним засобом для визначення табличних розподілів імовірностей.

```
struct table { // таблиця
array<1,hint,double> x;   //      масив значень
array<1,hint,double> p;   //      масив імовірностей
hint2 ihint;                      //      число інтервалів};
```

Спискові структури або списки використовуються для зв'язування об'єктів системи моделювання й створення динамічної (змінюваної в часі) модельного середовища. Нижче визначаються списки різних об'єктів. Загальними для них є наступні параметри:

```
struct parml { //ПАРАМЕТРИ СПИСКУ
double timel,             //      часи зайнятості,
       pretime;           //      попереднього звернення
int ci,                   //      число входів: загальне,
       co,                //      у порожній список
       ll,                //      довжина поточна,
       lm;                //      максимальна
alfa name;                //      ім'я списку};
```

Найважливішим списком є список транзактів - об'єктів, що переміщаються в моделі. У системі моделювання є кілька списків транзактів. Визначення списку наступне:

```
struct listt {        // СПИСОК ТРАНЗАКТІВ
ptransact first;  //      посилання на перший транзакт
parml p;          //      поле параметрів списку
plistt sled,      //      посилання на наступний,
       pred;      //      попередній список};
typedef struct listt* plistt;     //
```

посилання на список транзактів

Деякі списки транзактів поєднуються в загальні. Для цього використовуються наступні типи даних:

```
struct listl { // СПИСОК СПИСКІВ ТРАНЗАКТІВ
plistt first;     //      посилання на перший список тр.
int ll;           //      кількість списків у списку
};
```

Обумовлені користувачем об'єкти: прилади, черги, накопичувачі й гістограми зв'язуються в списки, типи яких визначені нижче.

```
struct listf {      //      СПИСОК ПРИЛАДІВ
```

pfacility first; // посилання на перший прилад
parml p; // поле параметрів списку
plistf sled, // посилання на наступний,
pred; // попередній список };
typedef struct *listf* plistf*; // посилання на список приладів
struct *listq* { // СПИСОК ЧЕРГ
pqueue first; // посилання на першу чергу
parml p; // поле параметрів списку
plistq sled, // посилання на наступний,
pred; // попередній список};
typedef struct *listq* plistq*; посилання на список черг
struct *lists* { // СПИСОК НАКОПИЧУВАЧІВ
pstorage first; // посилання на перший накопичувач
parml p; // поле параметрів списку
plists sled, // посилання на наступний,
pred; // попередній список };
typedef struct *lists* plists*; // посилання на список накопичувачів
struct *listh* { // СПИСОК ГИСТОГРАММ
phistogram first; посилання на першу гістограму
parml p; // поле параметрів списку
plists sled, // посилання на наступний,
pred; // попередній список };
typedef struct *listh* plisth*; // посилання на список гістограм

Списки приладів, черг, накопичувачів і гістограм необхідні для організації автоматичної видачі результатів моделювання. Покажчики (посилання) на пойменовані списки використовуються системою моделювання.

4.2.4 Системні змінні

У системі моделювання визначена невелика кількість загальносистемних змінних, за допомогою яких здійснюється керування моделюванням і відслідковується стан моделі.

Більшість системних змінних змінюється автоматично, довільне присвоювання їм значень може непередбаченим чином вплинути на хід моделювання.

Виключення становлять наступні змінні, керуючі вихідним друком по ходу моделювання:

bool *trace*, ключ включення трасування

errtest; ключ розширеної (аварійної) видачі

Далі будемо давати визначення змінної й приводити її опис.

Для відліку відносного часу на черговому кроці модельного експерименту використовується змінна, яка встановлюється в момент скидання статистики:

double *resettime*; час останнього скидання статистики

Змінна **"СИСТЕМНИЙ ЧАС"** - чисельна величина, щодо якої синхронізуються всі події в моделі. При здійсненні подій системний час може або залишатися незмінним, або збільшуватися.

double *systime*; системний час

Службова змінна, що використовується в підсумковій видачі:

event ievemax; остання виконувана подія

"**ПОТОЧНА ПОДІЯ**" - подія, пов'язана з переміщенням активного транзакта.

event sysevent; поточна подія

Службова змінна, що встановлює число транзактів доступних у моделі:

int *itransmax*; кількість транзактів в *delist*

"**АКТИВНИЙ ТРАНЗАКТ**" - транзакт, що просувається по моделі в поточний момент виконання програми. Він визначається змінної:

ptransact trans; посилання на транзакт, що просувається Наступні змінні задають основні системні списки транзактів:

plistt delist, посилання на пасивний буфер

current, посилання на ланцюг поточних подій

future; посилання на ланцюг майбутніх подій

Об'єкти системи моделювання заносяться в списки, що задаються змінними:

plistq quelist; посилання на сист. список черг

plistf faclist; посилання на сист. список приладів

plists stlist; посилання на сист. список накопичувачів

plisth histlist; посилання на сист. список гістограм

Транзакти, що очікують появи деякого сигналу, події, що збираються в ансамблях, заносяться в списки, визначені змінними:

array<min_signal,max_signal,plistt> signlist: посилання на списки транзактів, очікуючих сигналів

array<1,evemax,plistt> ass; посилання на списки ансамблів, що збираються

array<1,evemax,plistt> waitl; посилання на списки транзактів, очікуючих подій

Списки, що задаються користувачем, заносяться в загальний список або список списків:

listl userlist; список списків користувача

Наступні змінні є службовими:

*array<1,evemax,**bool**> waitevent*; ознаки здійснення подій

*array<1,evemax,**int**> maxevent*; лічильники виконання подій

***long** V0*, джерела генераторів випадкових чисел

V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12,V13,V14,V15;

***double** eventall*; загальне число подій, що відбулися

ofstream outfile; основний вихідний файл

ofstream f; допоміжний вихідний файл

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке транзакт, прилад, черга, нагромаджувач?
2. За якими критеріями можуть класифікуватися об’єкти в SIMC?
3. Які ви знаєте системні константи SIMC?
4. Назвіть основні системні типи SIMC?
5. Які ви знаєте системні змінні SIMC?

5 ПРИНЦИПИ Й МЕТОДИ ПОБУДОВИ МОДЕЛЕЙ

5.1 Створення транзактів

Здійснення подій у моделі логічно пов'язане з переміщенням окремих транзактів. У момент здійснення події готовими до переміщення можуть виявитися один або декілька транзактів. Вони переміщуються по черзі. Порядок пересування готових транзактів визначається їхнім пріоритетом. Модельний час змінюється в тому випадку, коли список готових до переміщення в поточний момент часу транзактів вичерпаний. Переміщуємий транзакт назвемо активним.

Перш ніж транзакти стануть доступними до переміщення в моделі, вони повинні бути створені.

У моделі з єдиним просуваємим (активним) транзактом зв'язується посилання *trans*. Якщо її значення дорівнює *nil*, то в моделі транзактів немає взагалі, або вони перебувають в одному зі списків (по посиланнях: *current, future, waitl* (масив списків), списки приладу, накопичувача).

Створювані транзакти надходять у модель зі списку *delist*, а після завершення свого шляху в моделі повертаються в нього назад.

Уведення транзакта в модель здійснюється функцією ***void*** *create(**double** r)*, її єдиний параметр задає інтервал часу, через який у систему ввійде наступний транзакт. Величина "*r*" може бути константою або змінної, задаватися функцією. Залежно від способу визначення "*r*", функція *create* може виконувати роль генератора транзактів з різними (довільними) законами розподілу інтервалів між моментами їхнього надходження в модель. Всі події, що містять функцію *create*, повинні бути ініціалізовані при створенні модельного середовища. Для цього використовується функція ***void*** *initcreate(event e,**double** r)*, де *e* - номер події, що містить функцію *create(r)*, *r* - час надходження першого транзакта, що створюється функцією *create(r)*. Якщо ж необхідно забезпечити одночасний вихід транзактів у подію *e*, то функція *initcreate(e,r)* виконується відповідну кількість разів.

Приклад 1. Уведення транзактів у модель здійснюється кожні 5 одиниць модельного часу

```
#include "simc.h"
void main() {
...;
```

```
initcreate(4,0);      Ініціалізація події 4
...;
while(systime<1000)      {
        plan();
        switch(sysevent) {          Вибір системної події
                case 1: ...; break;
                ...: ...;
                case 4: create(5); break;   Створення транзактів
                ...: ...
        }}printall();  }
```

Приклад 2. Транзакти надходять у модель за законом нормального розподілу із середнім 4.0 і дисперсією 1.5.

```
#include "simc.h"
void main() {
...;
initcreate(3,0);      Ініціалізація події 3
...;
while(systime<1000) {
        plan();
        switch(sysevent){
                ...: ...;
                case 3: create(randnorm(4.0,1.5,v1)); break;
                ...: ...    v1 - джерело генератора випадкових чисел
        }}printall(); }
```

5.2 Знищення транзактів

Транзакти, що завершили своє просування по моделі, залишають її, проходячи через функцію ***void*** *destroy()*. Фактично ця функція поміщає транзакт, на якій указує посилання *trans*, у список *delist*.

Приклад 3. Знищення транзакта в події номер 12.

```
#include "simc.h"
void main() {
...;
while(systime<1000) {
        plan();
        switch(sysevent) {
```

```
...: ...;
case 12: destroy(); break; Знищення транзактів
...: ...
}}}printall(); }
```

5.3 Просування транзактів

Транзакти можуть просуватися в моделі тільки в моменти здійснення подій. Кожен транзакт у моделі рухається від функції *create* до функції *destroy*. Звернень і до тих, і до інших функцій у системі може бути кілька. Будь-які паралельні процеси в моделі здійснюються послідовно (їх можна синхронізувати), тому в будь-який момент виконання програми може просуватися тільки один транзакт.

Якщо в деякий момент модельного часу почалося просування транзакта, то він переміщається по моделі (без збільшення модельного часу) від події до події доти, поки не зустріне одну з функцій: *destroy, delayt, wait, seize, infac, enter* або функції, що поміщають транзакт у список користувача.

Перші дві безумовно припиняють просування транзакта, інші - залежно від настання зазначеної у функції події (*wait*) або залежно від стану зайнятості зазначених у функціях пристроїв (*seize, infac, enter*).

Послідовність дій при виконанні події довільна, однак варто пам'ятати, що зазначені функції можуть установлювати *trans=nil* (*delayt, destroy* - завжди встановлюють) і тимчасово виводити транзакт із числа активних (здатних просуватися в моделі в цей момент модельного часу при даному стані системи). Єдина функція - *delayt* - точно встановлює значення модельного часу нової активації транзакта. Момент активації транзактів, що пройшли через функції *destroy, wait, seize, infac, enter* залежить від наступного поводження моделі.

5.4 Затримка транзактів. Функція *delayt*

Якщо значення модельного часу (*systime==c)*, а час затримки у функції ***void*** *delayt(**double**)* дорівнює *r*, то транзакт заноситься в список *future*, а час виходу транзакта зі списку дорівнює *r+c*. Розміщення транзактів у списку *future* здійснюється в порядку зростання часів виходу. Якщо в списку декілька транзактів мають однаковий час

виходу, то ці транзакти розміщуються в порядку убування їхніх пріоритетів. Транзакти з однаковими пріоритетами й часом виходу розміщуються в порядку їхнього надходження в список.

Приклад 4. Модель із найпростішим випадком затримки.

```
#include "simc.h"
void main() {
...;
initcreate(1,0);
...;
while(systime<1000) {
        plan();
        switch(sysevent) {
          case 1: create(40); break;
                ...: ...;
          case 10: delayt(50); break;  // затримати транзакт на 50 одиниць модельного часу
                ...: ...;
                case 12: destroy(); break;
                }}printall(); }
```

Приклад 5. Затримка транзакта на час, експоненціально розподілене з μ=4.0

```
#include "simc.h"
void main() {
...;
while(systime<2500) {
        plan();
        switch(sysevent) {
                ...: ...;
                  case 42: delayt(randexp(4.0,v1)); break;
                  ...: ...
                } }printall();}
```

5.5 Блокування транзактів

5.5.1 Функція *wait*

Просування транзакта в моделі може бути припинене до настання заданої події. Для цієї мети призначена функція ***void***

wait(event e), де ***e*** - номер очікуваної події. Якщо подія ***e*** уже відбулася, то транзакт продовжує просуватися по моделі, інакше транзакт, на який указувало посилання *trans*, заноситься в список по посиланню *waitl*[*e*] останнім у своєму класі пріоритетів. При цьому значення посилання *trans* установлюється рівним *nil*. Після виконання події *e* транзакт, перший у списку *waitl*[e], переводиться в список *current*.

Приклад 6. Використання функції *wait* для організації черги перед вузлом затримки. Надходження транзактов експонентне, затримка - постійна, дисципліна обслуговування - FIFO. У цьому прикладі транзакти, що надходять на обробку, накопичуються в списку транзактів, що очікують подію 4, яке пов'язане зі знищенням транзакта, що переходить після затримки до події 4. Помітимо, що всі транзакти, що перебувають у списку *waitl[4]* у момент виникнення четвертої події будуть переправлені до події 3. Моделювання буде завершено, якщо число очікуючих транзактів буде більше або дорівнює 10.

```
void main(){
//variables
double j,lambda;
...;
while(waitl[4]->p->ll<10) {
        plan();
        switch(sysevent) {
                case 1: create(randexp(lambda,v1)); break;
                case 2: if(future->p->ll>1) wait(4); break;
                case 3: delayt(j); break;
                case 4: destroy(); break; }}}
...}
```

5.5.2 Функції *accept* й *send*

У системі СИМ-СИ визначені:

- тип "сигнал": *signal=1..signmax*;
- масив списків сигналів: *array<min_signal,max_signal,plistt> signlist*;
- дві функції *accept(signal sg), send(signal sg)*;

Функція void *accept(signal sg)* переводить активний транзакт у список сигналу *sg* зі списку *current*. Транзакт перестає бути активним і

залишається в цьому списку, очікуючи виконання функції ***void*** *send(signal sg)*. При виконанні функції **accept,** сигнали, які раніше надходили від функції *send,* не враховуються.

У відмінності від блокування просування транзактів чекаючих події, коли при здійсненні події активізується тільки один транзакт із очікуючих його, при формуванні сигналу функцією *send* всі транзакти зі списку сигналу sg переводяться в список *current*.

Фактично, *accept* - очікування сигналу *sg*, а *send* - посилка цього сигналу. *signmax* - системна константа, визначається при генерації SIMC.

5.6 Зміна порядку здійснення подій. Функція *next*

Якщо при обробці події *n* зустрілася функція ***void*** *next(event e),* де *e* - номер події, що повинне здійснитися після *n*, то значення *trans->eve* (а отже й *current->first->eve*) установлюється рівним *e*, а *trans=nil*. Далі автоматично вибирається активний транзакт: *trans=current->first*. Системна змінна *sysevent* одержить значення *sysevent=trans->eve*, тобто стане дорівнювати *e*.

5.7 Використання приладів

Використання приладів дозволяє зробити процес обробки більш наочним; крім того, по приладах збирається статистика, що може бути використана для оцінки їхнього функціонування. Прилади відносяться до об'єктів, названими у системі пристроями, тому функції звертання до них діляться на дві групи: функції уведення й функції виводу транзактів. Порядок запису функцій у моделі довільний, важливо, щоб логіка моделювання не була порушена спробою виводу транзакта з вільного приладу.

5.7.1 Функції уведення транзактів у прилад

5.7.1.1 Захват приладу. Функція *infac*

Функцією ***void*** *infac(pfacility&)* здійснюється захват приладу. Якщо прилад був вільний, то її дія аналогічна функції *seize*. Якщо ж у приладі перебував транзакт, то його обробка переривається й він надходить у список перерваних транзактів даного приладу (*f->inter*), а полю *status* приладу привласнюється значення *preempted*. Після звільнення приладу перерваний транзакт повертається в нього на

дообробку. Список перерваних транзактів будується в порядку, зворотньому надходженню. Якщо мають місце кілька рівнів захвату, то перервані транзакти обробляються з дисципліною LIFO. Число рівнів захвату не обмежено.

Якщо транзакт, що займає прилад перебуває в списку *current,* тобто захват повинен відбутися в той же момент модельного часу, що й звільнення приладу, то активний транзакт не захоплює прилад, а заноситься у відповідний список *waitl* і чекає звільнення приладу.

Приклад 7. Обробка транзактів з абсолютними пріоритетами

```
...;
if(f->transpoint!=nil)
if(trans->prty>f->transpoint->prty)
        infac(f);
else seize(f);
else seize(f);
```

5.7.1.2 Заняття приладу. Функція *seize*

Для реалізації дисципліни з відносними пріоритетами використовується функція ***void*** *seize(pfacility&)*. Якщо статус приладу дорівнює *free*, то його займає транзакт вищого пріоритету із числа, що претендують на його використання. Якщо прилад зайнятий (його статус дорівнює *seized*), то транзакт заноситься в список приладу останнім у своєму класі пріоритетів.

5.7.2 Функція *outfac*. Вивід транзакта з приладу

Всі події, здійснювані при занятті приладу, розміщуються після події, що містить функцію ***void*** *infac(pfacility&)*. Після того, як вони виконані, прилад необхідно звільнити, щоб забезпечити подальше просування транзакта, що займає прилад, і зробити прилад доступним для інших транзактів. Для цього використовується функція ***void*** *outfac(pfacility&)*.

Прилад звільнюється поза залежністю від яких-небудь умов, однак функція перевіряє, який транзакт звільняє прилад. Якщо прилад звільняється не тим транзактом, що його займав, то друкується повідомлення про помилку.

Якщо прилад був захоплений, то повертається на дообробку транзакт, що раніше його займав.

Якщо список приладу, що звільняє, містить транзакти, то **функція** outfac робить заняття приладу першим транзактом із цього списку.

Функція *outfac* звільняє прилад, зайнятий як функцією *seize*, так і функцією *infac*.

5.8 Реєстрація черг. Функції *inqueue, outqueue*

Черги можуть утворюватися в будь-яких частинах моделі, де виникає необхідність затримки або блокування транзактів. Спеціального обліку їхньої динаміки в загальному випадку не ведеться. Засобами для збору статистики й оцінки динаміки черг служать об'єкти типу *queue*.

Для входу в чергу використовують функцію ***void*** *inqueue(pqueue&),* для виходу з неї - ***void*** *outqueue(pqueue&).*

Функції дозволяють безумовно вставати в чергу й виходити з неї.

5.9 Накопичувачі (багатоканальні пристрої). Функції *enter* й *leave*

Для заняття транзактом багатоканального пристрою використовується функція ***void*** *enter(pstorage s,**int** c),* де c – кількість займаних осередків

Для звільнення багатоканального пристрою використовується функція ***void*** *leave(pstorage s,**int** c),* де c – кількість осередків, що звільняють

В відмінність від приладу накопичувач може бути звільнений не тим транзактом, яким був зайнятий.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке прилад?
2. Яка різниця між заняттям та захопленням приладу?
3. Які особливості виводу транзакту з приладу?
4. Яка різниця між приладом та нагромаджувачем?

5.10 Побудова гістограм. Функції *tabulate, newhist, prnhist*

Побудова гістограм у системі можлива для будь-якого типу скалярних даних, які можливо привести до типу ***double***.

З поняттям "гістограма" зв'язується об'єкт типу *histogram*. У моделі він визначається посиланням на нього: *phistogram h*;

Перед використанням змінної *h* для табулювання значень деякої випадкової величини необхідно створити гістограму. Для цього служить функція ***void*** *newhist(phistogram&,**double**,**double**,hint2,**bool**,alfa)*:

newhist(<посилання на гістограму>,
<нижня межа зміни табулюємої величини>,
<верхня межа зміни табулюємої величини>,
<число інтервалів табулювання>,
<"ключ" печатки графіка>,
<символьне ім'я гістограми>);

Максимальне число точок табулювання задається системною константою *hint=33*.

Функція **tabulate** має формат запису:

void *tabulate(phistogram hist,**double** r)*, де *hist* - посилання на гістограму, *r* - табулюєма величина. Функція записується в тій частині моделі, де вимірюється величина, що *цікавить* програміста, r.

Функція ***void*** *prnhist(phistogram)* служить для друку гістограми.

КОНТРОЛЬНІ ПИТАННЯ

1. В якому випадку необхідно використовувати об'єкт «ЧЕРГА»?
2. Що таке гістограма?
3. Що необхідно знати для створення гістограми?
4. Яка різниця між процедурами Resetall та Clear?

5.11. Створення, використання й обробка списків

У системі моделювання визначені поняття списків транзактів, приладів, черг, накопичувачів, гістограм і списку списків транзактів.

Для роботи з цими типами списків (крім списку списків транзактів) визначені функції створення списків, включення в них елементів й їхнє виключення. На списках задані функції проходження: вибір наступного й попереднього елементів зі списку. Всі списки в системі моделювання мають кільцеву структуру: перший елемент списку вказує на другий й останній, другий - на перший і третій... останній - на передостанній і перший. Елементи списку не нумеруються. Щоб виділити в списку перший елемент, на нього

вказує посилання *first* змінної типу список: *[посилання на список]->first*. Кільцева структура забезпечує пошук потрібного елемента в кожному із двох обраних програмістом напрямків.

Поняття списків визначається в СІ, тому нижче дається лише короткий огляд функцій, список їхніх параметрів і спосіб звертання до них.

5.11.1 Створення списків

Для створення списку необхідно визначити відповідну змінну типу посилання на нього:

plistt lt	посилання на список	транзактів
plistq lq	посилання на список	черг
plistf lf	посилання на список	приладів
plists ls	посилання на список	накопичувачів
plisth lh	посилання на список	гістограм

Списки створюються за допомогою наступних функцій:

***void** newtlist(plistt&)*	створення списку транзактів;
***void** newqlist(plistq&)*	створення списку черг;
***void** newflist(plistf&)*	створення списку приладів;
***void** newslist(plists&)*	створення списку накопичувачів;
***void** newhlist(plisth&)*	створення списку гістограм.

Списки створюються порожніми без яких-небудь об'єктів, що їх наповнюють. Для того, щоб помістити об'єкт у список, необхідно попередньо його згенерувати.

5.11.2 Включення об'єктів у списки

Для включення об'єкта в список використовуються наступні функції:

***void** inlt(plistt&,ptransact)*	для транзактів
***void** inlf(plistf&,pfacility)*	для приладів
***void** inlq(plistq&,pqueue)*	для черг
***void** inls(plists&,pstorage)*	для накопичувачів
***void** inlh(plisth&,phistogram)*	для гістограм

5.11.3 Видалення об'єктів зі списків

На объект, що видаляється, завжди вказує голова списку. наприклад: *l->first* - посилання на перший елемент списку l.

Таке визначення дозволяє до видалення елемента зі списку запам'ятати його присвоюванням *pl=l->first;* де *pl* - змінна посилального типу (*ptransact, pfacility, pqueue, pstorage, phistogram*).

Виключити елемент зі списку можна за допомогою наступних функцій:

void *outtlist(plistt&)*
void *outflist(plistf&)*
void *outqlist(plistq&)*
void *outslist(plists)*
void *outhlist(plisth)*

Покажчик списку *first* після видалення вказує на наступний один по одному об'єкт у списку. Якщо объект, що видаляє, єдиний у списку *l*, то встановлюється *l->first=nil*.

5.11.4 Перегляд елементів списку. Сканування

Доступ до елемента списку визначається посиланням на нього й записом після точки поля, значення якого необхідно програмістові, наприклад: *quelist->first->lq* - поточна довжина першої черги в списку черг *quelist*, *current->first->pr[1]* - перший дійсний (double) параметр транзакта, що стоїть першим у списку *current*.

Посилання first указує на голову списку. Для її просування вправо необхідно привласнити: *l->first=l->first->sled*

Для просування посилання first уліво привласнюється: *l->first=f->first->pred*

5.11.5 Списки користувача. Організація різних дисциплін обслуговування за допомогою списків користувача

Список користувача створюється функцією ***void*** *newuserlt(plistt& list,alfa name).*

При створенні список користувача заноситься в системний список списків транзактів *userlist*.

Функції ***void*** *inlfifo(plistt)* і ***void*** *inllifo(plistt)* виводять активний транзакт зі списку *current* і поміщають його в список користувача відповідно останнім або першим у списку.

Користувач може написати свої функції, що впорядковують список по якій-небудь ознаці, наприклад, у порядку зростання першого цілочисельного параметра:

void inlpi1(plistt lt) {

```
ptransact t;
outtlist(current);
trans->testprty=false;        // !!!
if(lt->first==nil)
        inlt(lt,trans);
else if(lt->first->pi[1] > trans->pi[1])
                inlt(lt,trans);
        else { t=lt->first; scanlt(lt);
                while(lt->first->pi[1] >= trans->pi[1])
                        scanlt(lt);
                inlt(lt,trans);
                lt->first=t;   }trans=nil;}
```

Функція ***void*** *outuserlt(plistt)* поміщає транзакт, що стоїть першим у списку користувача в список *current*.

Транзакт має поле *testprty*, що може приймати значення *true* або *false*. При виконанні функції *priority* (див. п. 2.13. 3) транзакти, значення поля *testprty* у яких *true*, переміщуються в списках відповідно до нового значення пріоритету. Якщо список упорядковується не по пріоритетах, а по якій-небудь іншій ознаці, то перед включенням транзакта в список його полю *testprty* необхідно привласнити значення *false*.

КОНТРОЛЬНІ ПИТАННЯ

1. Назвіть основні типи списків, які використовуються в Simpas-і?
2. Як створити список?
3. За допомогою яких процедур об’єкти включаються до списків та знищуються із списків?
4. Що таке сканування списку?
5. Навіщо створюються списки користувача?

5.12 Модельне середовище

5.12.1 Створення модельного середовища

Сукупність приладів, черг, накопичувачів й інших об'єктів моделі являє собою модельне середовище. Для створення модельного середовища написані спеціальні функції.

Для створення черги використовується функція *newqueue(<посилання на чергу>,<ім'я>)*. Її параметрами є змінна типу посилання на чергу й ім'я черги з восьми символів. Посилальна змінна визначається в такий спосіб:

```
pqueue q1;
...
newqueue(q1,'q1  ');
```

Значення q1 установлюється на створену чергу.

Створення приладів і накопичувачів здійснюється аналогічним чином:

```
pfacility f1;
pstorage st1;
...
newfac(f1,'f1      ');
newstorage(sm1,'sm1      ');
```

Створені об'єкти включаються в системні списки *quelist, faclist* й *stlist*. Якщо користувачеві треба помістити об'єкт у який-небудь інший список черг, приладів або накопичувачів, необхідно попередньо видалити його з відповідного системного списку.

Створення гістограм було розглянуто вище.

5.12.2 Знищення черг, приладів, накопичувачів і гістограм

Об'єкти, виключені зі списків можуть бути знищені за допомогою функцій

```
void destrs(pstorage&)
void destrf(pfacility&)
void destrq(pqueue&)
void destrh(phistogram&)
```

при цьому виділена під об'єкт пам'ять звільняється.

Користуватися цими функціями треба обережно. При знищенні приладу або накопичувача можуть втратитися транзакти, що перебувають у їхніх списках (буде видане відповідне повідомлення про помилку). Якщо в ході моделювання буде звертання до знищенного об'єкту, то виконання програми буде перервано.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке модельне середовище?

2. Що таке системне середовище?

3. Яке призначення процедури InitList?

4. В яких випадках необхідно застосовувати процедури знищення черг, приладів, нагромаджувачів та гістограм?

5.13 Ансамблі

5.13.1 Створення ансамблів. Функція *split*

Транзакти можуть уводитися в модель як функціями *create* й *initcreate*, так функцією ***void*** *split(**int** n,event e)*, де *n* - число створюємих транзактів, *e* - номер події, у яке направляються знову створені транзакти. Активний транзакт, на який указує посилання *trans*, називається транзактом-батьком, а створені тразакти - нащадками.

Всі нащадки мають такі ж значення параметрів і пріоритет як у батька. Транзакти, уведені у модель функцією *split* разом із транзактом-батьком є членами одного ансамблю. Якщо який-небудь із нащадків буде оброблений функцією *split* і сам стане батьком, то всі його нащадки будуть належати до того ж ансамблю.

5.13.2 Збір членів ансамблю. Функція *assemble*

Функцією ***void*** *assemble(**int** n)* здійснюється збір *n* членів ансамблю. При вході першого члена ансамблю в подію, що містить функцію *assemble*, він заноситься в системний список *ass[sysevent]*.

Наступні члени цього ансамблю виводяться функцією *assemble* у список *delist*.

При вході n-го члена ансамблю в подію, що містить функцію *assemble* перший член ансамблю, що збирається, видаляється зі *списку ass[sysevent*] і заноситься в *список* delist, а n-ий член ансамблю продовжує просуватися по моделі далі.

5.13.3 Зміна пріоритету всіх членів ансамблю. Функція ***priority***

Для зміни пріоритету активного транзакта й всіх транзактів - членів того ж ансамблю служить функція ***void*** *priority(prtyrange p)*, де p - нове значення пріоритету. При цьому інші члени ансамблю переміщуються в списках відповідно до нового значення пріоритету.

Призначення пріоритету окремому транзакту виробляється простим присвоюванням: *trans->prty=<новий пріоритет>;*

5.13.4 Зміна значення параметра всіх членів ансамблю. Функція *parmans*

Для зміни значення параметра всіх членів ансамблю служить функція ***void*** *parmans(parmtype,**int**)*. Попередньо необхідно привласнити нове значення даному параметру активного транзакта. Після виконання функції *parmans* призначений параметр всіх членів ансамблю приймає те ж значення, що й відповідний параметр активного транзакта.

5.14 Генератори випадкових чисел

У процесі моделювання майже завжди необхідна генерація випадкових величин. Для цього в системі існують функції:

double *rand01(**long**& v)* - генератор випадкових чисел, рівномірно розподілених в інтервалі [0;1). v - число-"джерело".

double *randab(**double** a, **double** b, **long**& v)* - генератор випадкових чисел, рівномірно розподілених в інтервалі [a;b). v - число-"джерело".

double *randexp(**double** lambda, **long**& v)* - генератор експоненціально розподілених випадкових чисел з інтенсивністю lambda. v - число-"джерело".

double *randnorm(**double** xmean, **double** disp, **long**& v)* - генератор випадкових чисел, розподілених за нормальним законом із середнім xmean і дисперсією disp. v - число-"джерело".

Приклад 8. Присвоєння змінної *r* значення випадкової величини, рівномірно розподіленої в інтервалі [4;7)

```
r=randab(4.0,7.0,v)
```

Закон розподілу випадкових чисел може бути заданий таблицею. Для цього в системі визначений тип "таблиця" у такий спосіб:

```
struct table {
array<1,hint,double> x,p;
hint2 ihint;};
```

Для роботи з генераторами випадкових чисел, розподілених відповідно до таблиці, необхідно задати таблицю.

Для дискретного розподілу *ihint* відповідає кількості значень, які може приймати випадкова величина. *h* - масив значень, *p* - масив імовірностей, причому *p[1]* - імовірність присвоєння випадковій величині значення *x[1]*, *p[2]* - імовірність присвоєння значення *x[2]* за умови, що їй не привласнене значення *x[1]*, і т.д., так що *p[ihint]=1.0* (імовірність присвоєння випадковій величині значення *x[ihint]* за умови, що вона вона не прийняла жодне з попередніх значень.

Реалізує дискретний табличний розподіл функція ***double*** *randdtable(table t,* ***long****& v)*, де *t* - таблиця, *v* - число-"джерело".

Для безперервного табличного розподілу генератор - функція ***double*** *randtable(table t,* ***long****& v)* (*t* - таблиця, *v* - "джерело") - дає випадкові числа, з імовірностями *p[i]* які потрапляють в інтервали від *x[i-1]* до *x[i]*. При цьому в таблиці повинне бути *p[1]=0.0.*

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке ансамбль?
2. За допомогою яких процедур можно створити транзакти?
3. Який принцип роботи процедури Assemble?
4. Яким чином можна змінити приорітет і параметри членів ансамблю?

5.15 Процес моделювання. Функція *plan*

Процес моделювання полягає в багаторазовому повторенні функції ***void*** *plan****()*** і блоку вибору події.

Усередині блоку вибору події оператор ***switch****(sysevent)* вибирає подію, номер якого збігається зі значенням *sysevent.*

Функція *plan*:

а) переводить перший транзакт зі списку *waitl[sysevent]*, якщо цей список непустий, у список *current*; при цьому встановлюється *waitevent[sysevent]=****false****;* якщо список *waitl[sysevent]* був порожній, установлюється *waitevent[sysevent]=****true****;*

б) перевіряє *trans==nil*:

якщо *trans==nil*, то функція *plan* виконує (у порядку запису) одне з наступних дій:

- намагається вибрати активний транзакт зі списку *current*;
- якщо список *current* порожній, то вибирає активний транзакт зі списку *future* і встановлює нове значення модельного часу, потім

вибирає зі списку *future* всі транзакти, значення поля *nexttime* яких збігається з новим значенням модельного часу;

- призначає *sysevent=trans->eve*.

Якщо *trans!=nil*, те призначається *sysevent=succ(sysevent)*.

5.16 Створення системного середовища. Функція initlist

Перед початком моделювання необхідно створити системні списки, створити необхідну кількість транзактів, помістивши їх у пасивний буфер і привласнити початкові значення системним змінним.

Це здійснюється функцією ***void*** *initlist(**int** n)*. Параметр цієї функції *n* - число транзактів, необхідне для моделювання.

5.17 Структура моделі

Модель являє собою функцію на C++ й оформлюється відповідним чином. Всі моделі мають загальні риси побудови. Типова структура моделі приводиться нижче:

```
void model() {
//variables
pfacility  <посилання на прилади>;
pqueue     <посилання на черзі>;
pstorage   <посилання на накопичувачі>;
plistt     <посилання на списки транзактів>;
phistogram <посилання на гістограми>;
...
initlist(<у транзактів>);
newfac(...);                          ...;
newqueue(...);                        ...;
newstorage(..., ...);          ...;
newuserlt(...);                ...;
newhist(..., ...);             ...;
initcreate(<номер події>,<час>); ...;
  обмеження числа повторень, наприклад, за часом:
while(systime<...) {
        plan();
        switch(sysevent){
                case 1: <дія для події 1>; break;
```

```
                    case 2: <дія для події 2>; break;
                    case 3: <дія для події 3>; break;
                    ...
                    case n: <дія для події N>; break;} }
    printall(); }
```

5.18 Скидання статистики й очищення системного й модельного середовища. Функції resetall й clear

Функція ***void*** *resetall()* служить для скидання статистики, накопиченої в процесі моделювання. Для всіх списків, черг, приладів і накопичувачів обнуляются число входів, час зайнятості; час попереднього обігу стає рівним *systime*, і т.д.

Функція ***void*** *clear()* повертає усі транзакті в список *delist,* скидає всю накопичену статистику, тобто повертає систему в стан, що передує початку моделювання.

КОНТРОЛЬНІ ПИТАННЯ

1. Який алгоритм роботи процедури Plan?
2. В яких випадках відбувається зміна модельного часу?
3. Із яких основних елементів складається структура моделі?
4. Що таке ядро генератора випадкових чисел?
5. Яки способи задання законів розподілення випадкових чисел існують в Simpas?

5.19 Вивід результатів моделювання

Для друку статистики, зібраної в результаті моделювання, служить функція ***void*** *printall*(). Статистика, що роздруковується цією функцією при *errtest==false*, містить:

- значення абсолютного системного часу у вигляді
systime=<абсолютний системний час>;
- номер поточної події у вигляді
sysevent=<номер поточної події>;
- для всіх виконуваних подій: кількість виконань кожної події у вигляді
event <номер події>
total <число виконань>
- для всіх черг у моделі: ім'я черги, число входів, число входів з нульовим часом очікування, максимальна довжина, середній час

очікування, відсоток входів у порожню чергу, поточна довжина, середній час очікування без обліку нульових входів, середня довжина черги;

- для всіх приладів у моделі: ім'я приладу, число входів, середній час обробки транзакта в приладі, завантаження, число захватів;

- для всіх накопичувачів у моделі: ім'я накопичувача, ємність, завантаження, середній час перебування транзакта в накопичувачі, поточний уміст, максимальний уміст, середній уміст, число входів;

- для всіх списків користувача: ім'я списку, поточна довжина, максимальна довжина, транзакти, що перебувають у списку з полями: номер, *prty* (пріоритет), *eve* (подія ініціалізації), *nexttime* (час потрапляння в список), *ans* (номер ансамблю), параметри транзакта.

- для видачі на друк розширеної (аварійної) статистики необхідно виконати присвоювання *errtest=true*; при цьому додатково друкуються системні списки по посиланнях *current, future, waitl[i]*, де *i=1..iEveMax* (номер останньої виконуваної події) і інші з розміщеними в них транзактами. Друк системних списків виконується по аналогії з друком списків користувача, який описано вище.

У системі існують функції для виводу на друк деякої частини статистики:

void *prnt1(ptransact t)* - друк полів транзакта. t - посилання на транзакт

void *prnlt(plistt& lt)*; - друк полів списку транзактів і транзактів, що

перебувають у ньому . *lt* - посилання на список

void *prwaitl()* - друк списків, що очікують транзактів (списків *waitl[i]*)

void *prnq(pqueue q)* - друк полів черги. *q* - посилання на чергу

void *prnlq(plistq lq)* - друк списку черг. *lq* - посилання на список

void *prns(pstorage st)* - друк полів накопичувача. *st* - посилання на накопичувач

void *prnls(plists ls)* - друк списку накопичувачів. *ls* - посилання на список

void *prnf(pfacility f)* - друк полів приладу. *f* - посилання на прилад

void *prnlf(plistf ls)* - друк списку приладів. *lf* - посилання на список

6 ЗАСОБИ ВІДЛАГОДЖЕННЯ

6.1 Діагностика помилок

Під час виконання моделі можуть виникати помилки, які реєструються C++ (приводять до аварійної зупинки), або не реєструються C++, але порушують логіку моделювання. Помилки першого виду можуть бути наслідком помилок другого виду, тому частина дій, які можуть викликати виникнення помилки, реєструється функціями моделювання. Якщо ці помилки серйозні, вони можуть завершитися функцією *exit()*. При цьому роздруковується розширена (аварійна) статистика. Коди всіх помилок з поясненнями виводяться на печатку за допомогою функції ***void*** *error(**int** errcode)*, де *errcode* - код помилки. Повний перелік кодів помилок і повідомлень, які регеструються, приводиться в додатку.

Появлення регеструємої помилки, у результаті якої продовження моделювання стає неможливим, приводить до запису аварійної статистики у файл *error.html*.

6.2 Трасування

У системі існує можливість "покрокового трасування". Для цього необхідно виконати функцію ***void*** *starttrace(),* інформація, *що* друкує при цьому, наведена в додатку.

ЛІТЕРАТУРА

Основна література

1. Системный анализ в экономике и организации производства. / С.А. Валуев и др. -Л.: Политехника, 1991.-398с.

2. Советов Б.Я., Яковлев С.А. Моделирование систем. / Б.Я. Советов, С.А.Яковлев. -М.: ВШ, 1985. - 271с.

3. Максимей И.В. Имитационное моделирование на ЭВМ. / И.В. Максимей.-М.: Радио и связь, 1988. -232с.

4. Калянов Г.Н. CASE структурный системный анализ. / Г.Н. Калянов. - М.: ЛОРИ, 1996. - 243 с.

5. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. / Дж. Рамбо, М. Блаха – СПб: Питер, 2007. -544с.

Додаткова література

6. Шрайбер Т.Дж. Моделирование на GPSS. - М.: Машиностроение 1980. -592с.

7. Попович П.Р., Губинский А.И., Колесников А.М. Эргономическое обеспечение деятельности космонавтов. - М.: Машиностроение. 1985. -272с.

8. Трофимов С.А. CASE-технологии: практическая работа в Rational Rose. / С.А. Трофимов. - М.: Бином-Прес, 2012. - 288 с.