**Лекція №7. Принципи структурного програмування**

Структурне програмування це методологія й технологія розробки програмних комплексів, заснована на наступних принципах:

*- програмування* повинне здійснюватися зверху-*униз;*

- увесь проект повинен бути розбитий на *модулі* з одним *входом* і одним *виходом* (оптимальний розмір модуля — кількість *рядків* на екрані *дисплея);*

- логіка *алгоритму* й *програми* повинна допускати тільки три основні структури: *послідовне виконання, розгалуження* й *повторення.* Неприпустимий *оператор* передачі керування в будь-яке місце *програми;*

- при розробці *документація* повинна створюватися одночасно із *програмуванням,* у вигляді коментарів до програми.

Ціль структурного програмування — підвищення надійності програм, забезпечення *супроводу* й модифікації, полегшення й прискорення розробки. У програмах з використанням структурного програмування добре простежується основний алгоритм, вони більш зручні в налагодженні і менш чутливі до помилок програмування.

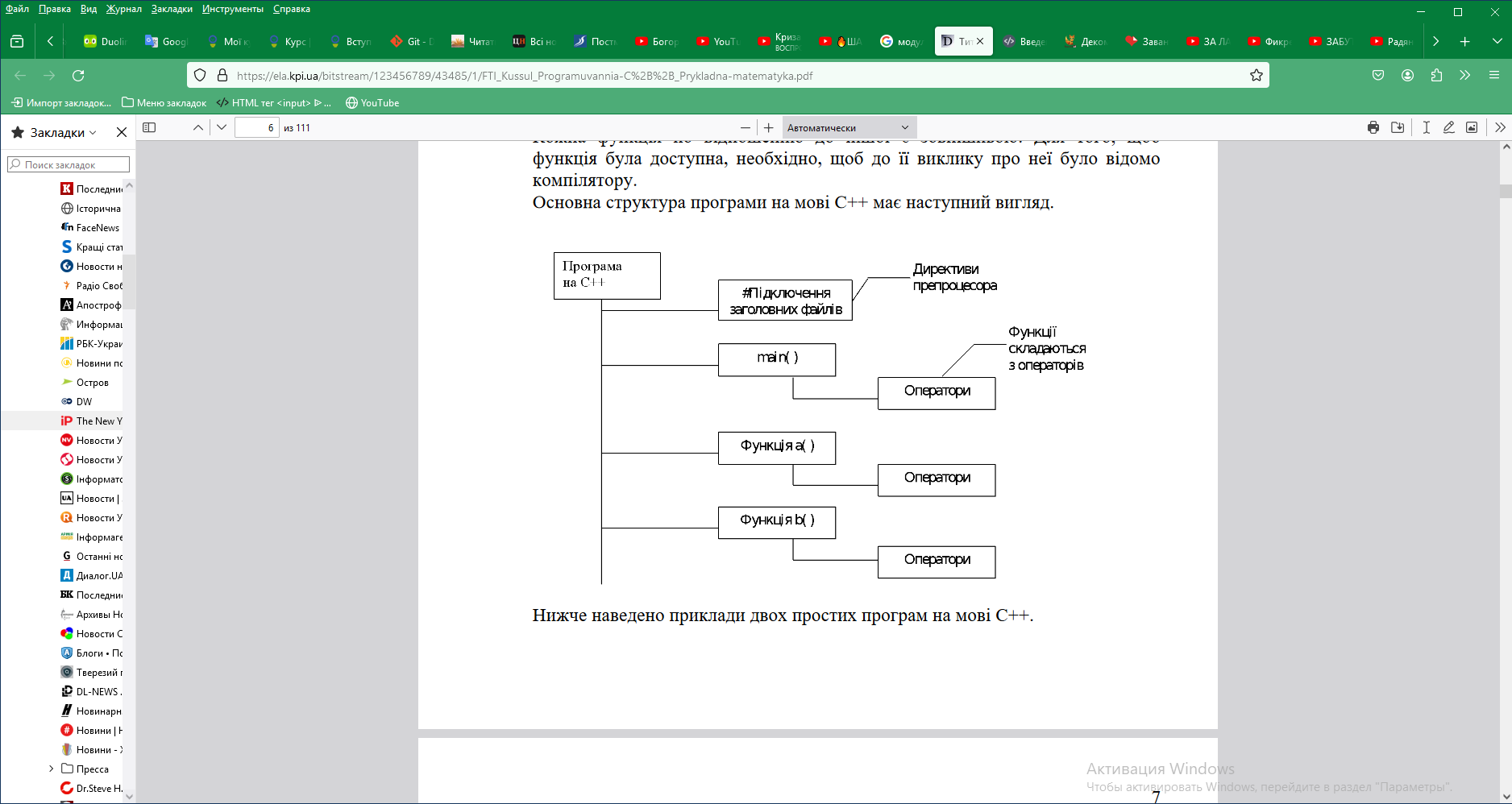
Витоки цієї методології сягають70-х років XX століття, коли швидкими темпами почала зростати складність програм, а отже, постала потреба у методах подолання такої складності. Вона ґрунтується на теоремі Бома-Якопіні (1966 р.), які довели, що будь-який виконуваний алгоритм може бути перетворений до структурованого вигляду, тобто такого виду, коли хід його виконання визначається тільки за допомогою трьох структур управління: послідовностей, розгалужень і циклів. Ідеї структурного програмування з'явилися на початку 700х рр. XX в. у компанії *IBM,* у їхній розробці брали участь відомі вчені Є. Дейкстра, X. Милі, Є. Батіг, С. Хоор.

Метою структурування є перетворення неструктурованої програми на еквівалентну їй структуровану, тобто таку, що складається з обмеженого набору керуючих алгоритмічних структур. Методи структурування ґрунтуються на поняттях функціонального вузла, а також на поняттях простої, елементарної і складеної програми.

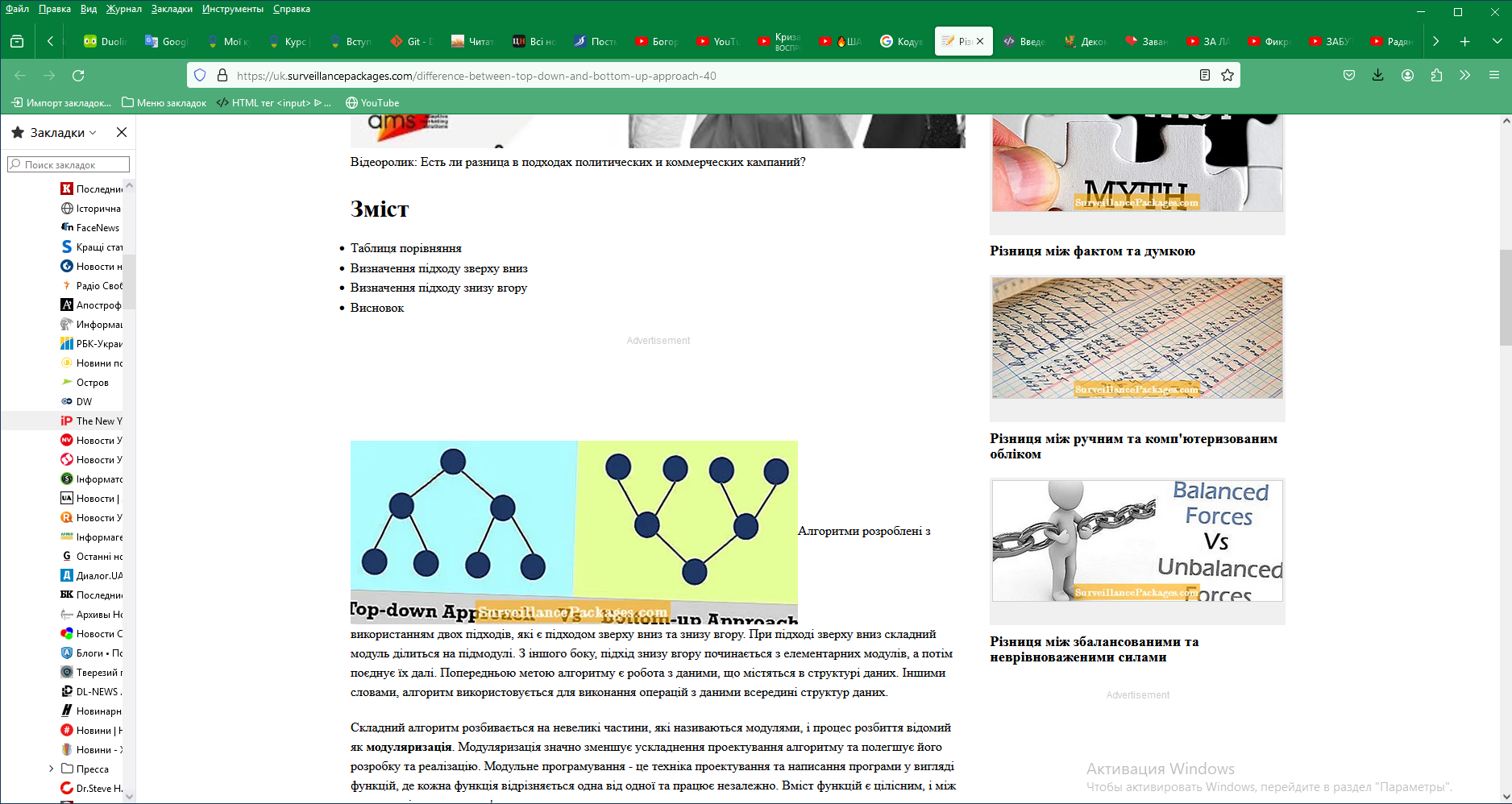
Існує декілька важливих моментів структурного програмування:

1. ***Вихідний код повинен мати модульну структуру***. Тобто, програма розділяється на дрібніші одиниці – процедури і функції. Ці частини або підпрограми, можуть викликатися з будь-якого місця у ній. Процедури – окремі ділянки коду, які виконують певні дії, задані алгоритмом та мають власну назву, в С/С++ термін "процедура" майже не використовується. Функції також можуть обчислювати деякі змінні,  мають значення, яке повертається, і можуть використовуватись в основній частині програми і в інших підпрограмах. Деякі підпрограми можуть мати рекурсивну структуру, тобто виклик з «самої себе». Це може допомогти вирішити задачу, але і призвести до зациклювання.  Таким чином, модульність визначає, що будь-яка частина програми може бути повторно використана або змінена пізніше без впливу на інші частини коду.

Кожна програма на мові С++ у своєму складі повинна мати головну функцію main(). Саме ця функція є початковою точкою входу в програму. Крім функції main() до програми може входити будь-яка кількість функцій. Кожна функція по відношенню до іншої є зовнішньою. Для того, щоб функція була доступна, необхідно, щоб до її виклику про неї було відомо компілятору. Основна структура програми на мові С++ має наступний вигляд.



1. ***Кодування програми повинно виконуватися зверху-вниз чи знизу вгору***. Схема «зверху-вниз» добре зрозуміла для дослідження написаної програми і пошуку помилок. Схема «знизу-вгору» використовується, коли алгоритм програми не розроблений, але вже написані деякі підпрограми, які реалізують певні дії.

Алгоритми розроблені з використанням двох підходів, які є підходом зверху вниз та знизу вгору. При підході зверху вниз складний модуль ділиться на підмодулі. З іншого боку, підхід знизу вгору починається з елементарних модулів, а потім поєднує їх далі. Попередньою метою алгоритму є робота з даними, що містяться в структурі даних. Іншими словами, алгоритм використовується для виконання операцій з даними всередині структур даних.

Складний алгоритм розбивається на невеликі частини, які називаються модулями, і процес розбиття відомий як **модуляризація**. Модуляризація значно зменшує ускладнення проектування алгоритму та полегшує його розробку та реалізацію. Модульне програмування - це техніка проектування та написання програми у вигляді функцій, де кожна функція відрізняється одна від одної та працює незалежно. Вміст функцій є цілісним, і між модулями існує низька зв'язок.

Таблиця порівняння

| **Основа для порівняння** | **Підхід зверху вниз** | **Підхід знизу вгору** |
| --- | --- | --- |
| Основні | Розбиває масштабну проблему на менші підзадачі | Вирішує фундаментальну проблему низького рівня та інтегрує їх у більшу |
| Процес | Підмодулі аналізуються поодиноко. | Вивчіть, які дані слід інкапсулювати, і передбачає концепцію приховування інформації. |
| Спілкування | Не потрібно в підході зверху вниз | Потрібен певний обсяг спілкування |
| Надмірність | Може містити зайву інформацію | Надлишок можна усунути |
| Мови програмування | Структурно / процедурно-орієнтовані мови програмування (тобто С) дотримуються підходу зверху вниз. | Об'єктно-орієнтовані мови програмування (такі як C ++, Java тощо) дотримуються підходу знизу вгору. |
| В основному використовується в | Документація модуля, створення тестового випадку, реалізація коду та налагодження | Тестування |

**З верху до низу** підхід в основному ділить складну задачу або алгоритм на кілька менших частин (модулів). Ці модулі піддаються подальшому розкладанню до тих пір, поки отриманий модуль не стане основною програмою, яку, по суті, можна зрозуміти і не можна розкласти далі. Після досягнення певного рівня модульності розкладання модулів припиняється. Підхід зверху вниз - це поетапний процес розбиття великого програмного модуля на простіші та менші модулі для ефективної організації та кодування програми. Потік управління при такому підході завжди йде вниз. Підхід зверху вниз реалізований на мові програмування "С" за допомогою функцій.

Таким чином, метод зверху вниз починається з абстрактного дизайну, а потім послідовно цей дизайн вдосконалюється, щоб створити більш конкретні рівні, поки не буде необхідності додаткового вдосконалення.

**Знизу вгору** підхід працює прямо протилежно до підходу зверху вниз. Спочатку він включає проектування найбільш фундаментальних частин, які потім поєднуються, щоб зробити модуль вищого рівня. Ця інтеграція підмодулів та модулів у модуль вищого рівня багаторазово виконується до отримання необхідного повного алгоритму.

Функції підходу знизу вгору з шарами абстракції. Основним застосуванням підходу знизу вгору є тестування, оскільки кожен фундаментальний модуль спочатку тестується перед об’єднанням його з більшим. Тестування проводиться за допомогою певних функцій низького рівня.

Підхід зверху вниз та підхід знизу - це методи проектування алгоритмів, де зверху вниз - це звичайний підхід, який розкладає систему від специфікації високого рівня до специфікації низького рівня. З іншого боку, підхід знизу вгору є більш ефективним і працює у зворотному порядку, коли спочатку розробляються примітивні компоненти, а потім переходять на вищий рівень.

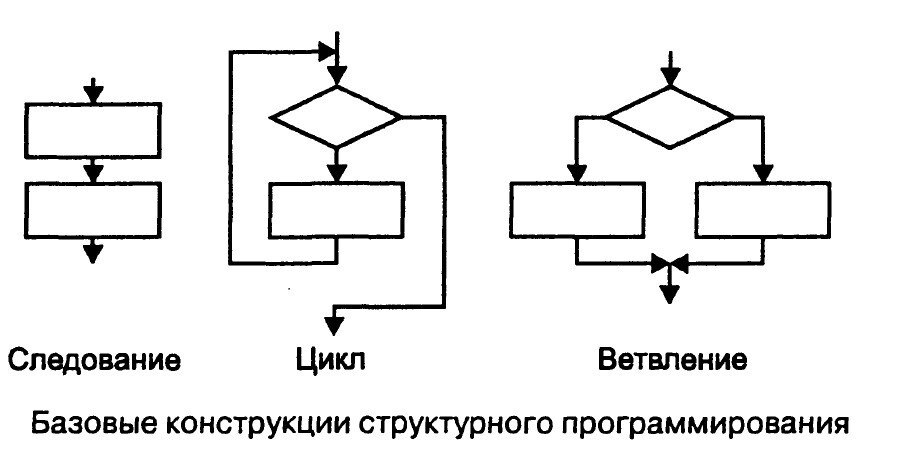
Підхід зверху вниз робить акцент на ізоляції підмодулів (означає низьку зв'язок між модулями), ігноруючи ідентифікацію концепції зв'язку та повторного використання. У підході знизу вгору, приховування інформації та її повторне використання є основними факторами.

1. Наявність керуючих елементів. ***У структурному підході використовуються цикли, умови і послідовності.***

Структурне програмування поліпшило загальне сприйняття коду та сприяло легшому написанню програм. За методами структурного програмування, алгоритм програми стає універсальним і за рахунок цього будь-який інший розробник зможе його змінити або використати у своїй програмі.

Метою структурування є перетворення неструктурованої програми на еквівалентну їй структуровану, тобто таку, що складається з обмеженого набору керуючих алгоритмічних структур. Методи структурування ґрунтуються на поняттях функціонального вузла, а також на поняттях простої, елементарної і складеної програми.

Елементарна програма — це проста програма, що не містить блоків, які складаються більше ніж з одного вузла. Потрібно зазначити, що існує обмежена кількість елементарних програм, з яких основними є три програми, які зображені на рис. 1.



**а) Слідування б) Цикл в) Розгалуження**

Рисунок 1 - Базові конструкції структурного програмування

***Структурне програмування допомагає створювати ефективний і надійний код.***

**Базові конструкції структурного програмування**

Структурне програмування ґрунтується на теоремі, яку спрощено можна переказати так:

* Дві програми вважаються еквівалентними, якщо за будь-яких однакових вхідних даних вони завжди видають однакові вихідні дані або однаково закриваються через помилку.
* Структурна програма використовує тільки три конструкції:
  + послідовне виконання – “виконати дію 1, виконати дію 2”;
  + розгалуження – “якщо умова дотримана, виконати дію 1, інакше дію 2”;
  + цикл – “поки умова дотримується, виконувати дію 1”.
* Оператор GoTo використовується для неструктурних програм.
* Для кожної неструктурної програми існує еквівалентна структурна програма.

Ґрунтуючись на цій теоремі, іноді структурне програмування називають “програмуванням без Go-to”. Але це не зовсім правильно. У кожному правилі є винятки, і в структурному програмуванні оператор go-to використовувати допустимо. Але все ж, перш ніж його використовувати, варто переглянути, чи не можна написати код без цього оператора.

При написанні програм використовують такі базові алгоритмічні конструкції :

1. Слідування – блоки алгоритму виконуються послідовно.

2. Розгалуження – в залежності від умови виконується одна або інша гілка алгоритму.

3. Цикл – група блоків алгоритму виконуються декілька разів. Обов’язковою вимогою є наявність умови виходу з циклу.

На попередніх заняттях ми розглядали програми лінійної структури, реалізацію розгалуження та циклу.

Якщо функціональний вузол елементарної програми замінити елементарною програмою, утвориться складена програма. Якщо вузол складеної програми замінити елементарною програмою, знов утвориться складена програма. В такий спосіб із будь-якої множини елементарних програм утворюється певний клас складених програм. Складена програма, побудована з певної множини елементарних програм, називається структурованою.

Позначимо літерою S клас складених програм, базисна множина якого містить такі елементарні програми, як послідовність (рис. 1, а), вибір (рис. 1, в) і цикл із передумовою ( рис. 1, б). Теорема про структурування стверджує, що будь-яку просту програму шляхом покрокового перетворення можна замінити функціонально еквівалентною структурованою програмою, що належить означеному вище класу S. Згадане покрокове перетворення здійснюється за переліченими нижче правилами:

1. *правило простоти* - створення програми слід починати із простої програми;
2. *правило пакетування* - кожну дію можна замінити двома послідовними діями (вихід одного функціонального вузла з'єднується із входом наступного);
3. *правило вкладання*–кожну дію можна замінити будь-якою структурою керування (будь-який блок може бути замінений на структуру керування вибору або повторення);
4. правила 2 і 3 можна застосовувати у будь-якій послідовності необмежену кількість разів.

Принцип застосування правил пакетування та вкладання до простої програми продемонстровано на рис. 2.

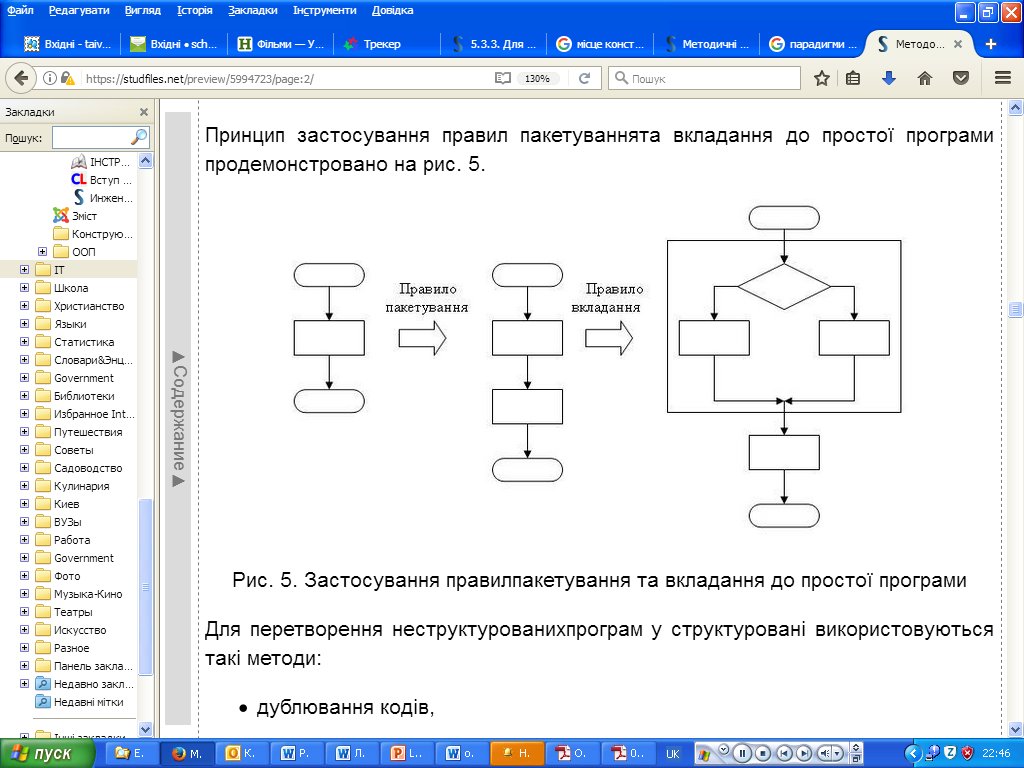


Рисунок 2 – Застосування правил пакетування та вкладання до простої програми.

Для перетворення неструктурованих програм у структуровані використовуються такі методи:

* дублювання кодів,
* введення змінної стану,
* метод булевих ознак.

*Метод дублювання кодів* застосовується для перетворення неструктурованих програм, що не містять циклів. Для отримання структурованої програми дублюють ті блоки (модулі), у які можна увійти з декількох точок.

*Метод булевої ознаки* використовується для перетворення неструктурованих програм, що містять цикли. В програму вводиться додаткова змінна, яка набуває значення true чи false. Доки змінна ознаки зберігає це значення, виконання циклу триває. Значення змінної ознаки всередині циклу модифікується лише за певних умов.

*Блочне програмування* — це організація програми у вигляді сукупності відносно незалежних складових частин - блоків, якими фактично є підпрограми. Як показує практика, великі задачі простіше вирішуються, якщо розглядати їх як сукупність менших задач.

Як правило, програмний блок розв’язує порівняно нескладну задачу, логічно незалежну від інших задач. Його властивості:

* відповідає лише одній задачі;
* має один вхід і один вихід;
* має порівняно невеликий розмір;
* доступний за своїм ідентифікатором;
* може викликати інший блок(не в усіх мовах програмування);
* підтримує незалежність функціонування (заміна підпрограми на аналогічну не впливає на всю програму).

Тобто, кожна програма представляється лінійною послідовністю блоків. Кожен блок складається з одного або декількох інших блоків, кожен із яких має також рівно один вхід і рівно один вихід і сам може розглядатися як блок. Сенс блоків, що мають один вхід і один вихід, полягає в тому, що можна вийняти їх, змінити їх начинку і вставити назад без зміни інших сполучень в програмі.

Блоки повинні бути незалежними в межах інтерфейсу програми і структури даних. Практика показала, що чим вищий ступінь незалежності підпрограм, тим простіше розібратись в окремих програмних блоках і в програмі в цілому; тим менша ймовірність так званого *хвильового ефекту* – появи нових помилок при виправленні старих або при внесенні змін у програму. Тому не варто без крайньої необхідності використовувати в підпрограмах глобальні змінні. Всі зв'язки між підпрограмами мають підтримуватися через списки параметрів.

Подивимось на низхідне програмування з розглянутих позицій.

В основу *методу низхідного проектування* покладено структурну декомпозиціюзадачі із застосуванням принципу покрокової (поетапної) деталізації. Спадне проектування передбачає, що розробка програм ведеться методом «зверху вниз», від загального до деталей. Спочатку задача розглядається як єдиний блок, що виражає загальне призначення програми. Далі ця задача поділяється на декілька дрібніших підзадач в тому порядку, в якому вони повинні виконуватися. Потім кожна з підзадач розбивається на свої підзадачі, що належать другому рівню деталізації і т.д. Процес покрокової деталізації підзадач здійснюється до тих пір, поки підзадачі чергового рівня не стануть досить простими для незалежного розв’язання (наприклад, кожній із них буде відповідати окрема команда мови програмування).

*Процедурне структурне програмува́ння* передбачає використання принципу *процедурної декомпозиції* на всіх рівнях проектування програмної системи. У відповідності із даним принципом кожен логічно цілісний програмний блок, отриманий на певному рівні деталізації, оформлюється як *підпрограма*. Наприклад, підпрограма обчислення визначника матриці, підпрограма з находження суми елементів ряду тощо.

Розробка процедурної структурованої програми ведеться покроково, методом "зверху вниз" з використанням механізму так званих «заглушок» - підпрограм, які нічого не роблять. При цьому спочатку пишеться основна програма, у якій замість кожного логічного зв'язного фрагмента тексту вставляється виклик підпрограми, яка буде виконувати цей фрагмент. Замість справжніх, працюючих підпрограм, в програму вставляються "заглушки". Отримана програма перевіряється та налагоджується. Після того, як програміст переконається, що підпрограми викликаються в правильній послідовності, тобто загальна структура програми вірна, підпрограми -"заглушки" послідовно замінюються на реально працюючі, причому розробка кожної підпрограми ведеться тим же методом, що і основної програми. Розробка закінчується тоді, коли не залишиться жодної "заглушки", яка не була б видалена.

Таким чином, «заглушки» дозволяють перевірити логіку програми верхнього рівня до реалізації наступного. Тобто на кожному кроці розробки програми існує працездатний «каркас», який поступово обростає деталями. Така послідовність гарантує, що на кожному етапі розробки програміст одночасно має справу з доступною для огляду і зрозумілою йому множиною фрагментів і може бути впевнений, що загальна структура всіх більш високих рівнів програми вірна. При супроводженні та внесенні змін у програму з'ясовується, в які саме процедури потрібно внести зміни, і вони вносяться, не зачіпаючи безпосередньо не пов'язані з ними частини програми. Це дозволяє гарантувати, що при внесенні змін і виправленні помилок не вийде з ладу якась частина програми, що знаходиться в даний момент поза зоною уваги програміста.

Таким чином, при структурному програмуванні програма ієрархічно структурується і розробляється шляхом послідовного уточнення на кожному рівні ієрархії. В основу цього процесу, окрім принципу ієрархічності, покладено принципи абстрагування.

*Абстрагування* — це спрощений опис системи, в якому зосереджують увагу на певних властивостях і деталях, а на інші не зважають. Вдалою є та абстракція, що підкреслює суттєві деталі і відкидає несуттєві. Під час низхідного проектування програми на верхніх рівнях абстракції деталі реалізації приховуються, а на нижніх рівнях вони описуються конкретною мовою програмування.

Фактично такий підхід збільшив структурованість програм – велика програма стала сукупністю процедур - підпрограм. Одна підпрограма, головна, розпочинала роботу всієї програми.

Одним із прийомів формалізованого підходу до низхідного процедурного проектування є метод ієрархічних діаграм, що позначається абревіатурою НІРО (від англ. *Hierarchical Input Processing Output* — діаграма входу, обробки, виходу). Згідно з цим методом структура всієї програми подається у вигляді дерева, в якому підпрограми зображуються вузлами, а їхні виклики — ребрами.

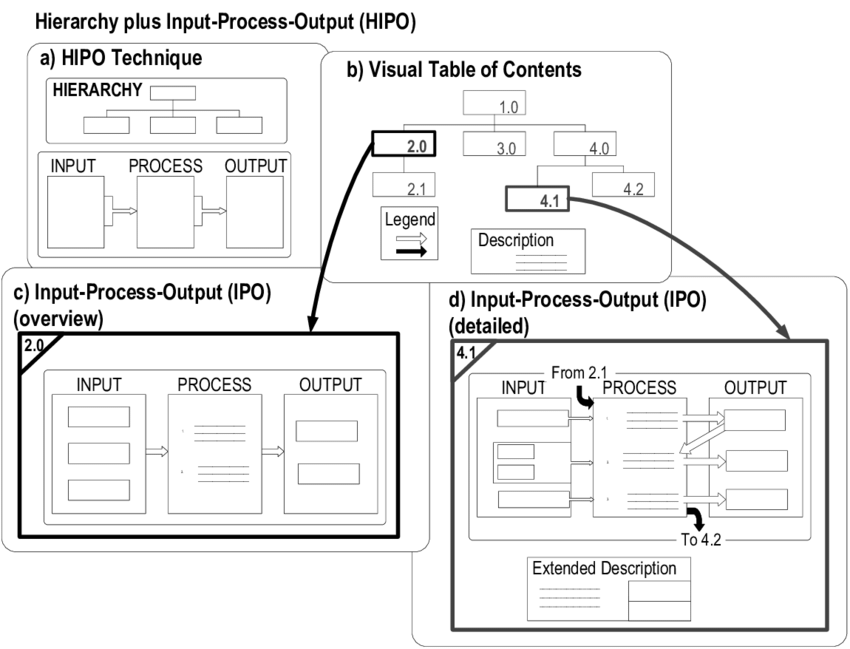


Рисунок 3 – Вигляд НІРО-діаграми

Методи низхідного проектування та блочного програмування регламентують процес побудови архітектури програм на макрорівні. Методи структурування, навпаки, працюють на мікрорівні, регламентуючи процес створення програмного коду.

Сутність методу *паралельного документування* полягає у тому, що при розробці програм документація повинна створюватися одночасно із програмуванням, зокрема, у вигляді коментарів до програми. Документування програмного тексту дозволяє будь-якому програмісту легко читати і розуміти його у процесі розробки та подальшого супроводу.

Впровадження принципів структурного програмування зробило тексти програм, навіть досить великих, більш читабельними. Серйозно полегшилось розуміння програм, з'явилася можливість розробки програм в нормальному промисловому режимі, коли програму може без особливих труднощів зрозуміти не тільки її автор, а й інші програмісти. Все це полегшило і прискорило процес розробки програмного забезпечення.

**Переваги структурного програмування**

Структурне програмування часто розглядають як ідеальний підхід до програмування, оскільки воно дає змогу створювати структурований і легко підтримуваний код. Це особливо корисно у великих і складних додатках, де організація коду важлива як для програмістів, так і для кінцевих користувачів.

Структурне програмування має низку переваг перед неструктурованими методами, зокрема:

* Код краще читається;
* Код стає простішим;
* Код стає надійнішим, що означає менше помилок або збоїв;
* Покращено обслуговування коду, тобто його легше змінити або оновити в міру необхідності.

Переваги структурного програмування особливо яскраво проявляються у великих проєктах. Завдяки структурності:

* легко простежується логіка програми, що важливо для розробників, які не брали участі в проєкті від самого початку;
* фрагменти коду, оформлені як процедури, легко використовувати повторно;
* підтримувати проєкт можна без участі перших авторів через роки.

***Методи та концепції структурного програмування***

Існує кілька методів і концепцій, які використовуються в структурному програмуванні для створення добре структурованого та ефективного коду.

1. Принцип єдиної відповідальності. Цей принцип свідчить, що в кожного модуля або класу має бути тільки одна причина для зміни. Кожен модуль має чітку відповідальність і не відповідає за кілька областей програми.
2. Об’єктно-орієнтоване програмування. Об’єктно-орієнтоване програмування – це тип програмування, у якому використовуються об’єкти, що включають у себе як дані, так і функції або методи, які працюють із цими даними. Об’єктно-орієнтоване програмування часто використовується для створення складних і модульних програм.
3. Абстракція функції або методу: це процес створення високорівневого інтерфейсу для фрагмента коду, який може використовуватися іншими частинами програми. Це дозволяє іншим частинам програми взаємодіяти з кодом без необхідності знати конкретні деталі.
4. Інкапсуляція: це процес приховування деталей структури даних або об’єкта від іншої частини програми. Це дає можливість використовувати об’єкт або структуру без необхідності вникати в її внутрішню роботу.
5. Спадкування: це створення нового класу або модуля, який успадковує властивості та методи іншого класу або модуля. У такий спосіб можна створювати сімейство пов’язаних класів або модулів зі спільними характеристиками.
6. Поліморфізм. Це здатність програми поводитися з об’єктами різних класів так, як якщо б вони належали до одного й того самого класу. Це дозволяє програмі використовувати один метод для роботи з кількома класами.

Є багато інших методів і концепцій, які трапляються в структурному програмуванні, але ці найчастіше використовують для створення надійного та ефективного програмного забезпечення.

**Мови структурного програмування**

Структурне програмування використовують різні мови, але найпопулярніші це C, C++, Java і Python. Ці мови використовують конструкції структурного програмування, як-от оператори if-else, цикли та функції, що дає змогу програмістам створювати з їхньою допомогою добре структурований та ефективний код.

Крім того, багато мов програмування мають спеціальні методи та концепції, що підтримують структурне програмування, наприклад об’єктно-орієнтоване програмування, інкапсуляцію, успадкування та поліморфізм. Ці методи та концепції допомагають програмістам створювати надійне та модульне програмне забезпечення, яке легко підтримувати та розширювати.

Загалом, використання принципів і методів структурного програмування може поліпшити якість і зручність супроводу програмного забезпечення на будь-якій мові програмування, що робить його важливим інструментом для освоєння розробниками.

***Для самостійного вивчення (2 години)*:** Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

***Контрольні запитання***.

1. Які оператори реалізують розгалуження у програмі?
2. Як діє умовний оператор if?
3. Як працює оператор-перемикач switch?
4. Які оператори циклу використовуються у C++?
5. Як працює оператор циклу for?
6. Пояснити на прикладах використання циклу з передумовою і циклу з післяумовою.
7. Які оператори з розглянутих є операторами керування у мові C++?
8. Наведіть власні приклади умовного оператору **if** у двох формах.
9. Наведіть власний приклад оператор-перемикача **switch**.
10. Наведіть власні приклади всіх форм оператору циклу.
11. Наведіть власні приклади використання операторів переривання **break** та продовження циклу **continue.**

***Література***

1. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
2. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
3. Бондарев В. М. Программирование на С++: Учеб. пособие. — Харьков: СМИТ, 2004г. — 294 с. URL: <https://www.rulit.me/author/bondarev-v-m/programmirovanie-na-c-get-161082.html>
4. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>
5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с.