**Київський національний університет імена Тараса Шевченка  
Факультет комп’ютерних наук та кібернетики**

**Алгоритми та складність  
Лабораторна робота №2**

**Алгоритм розбивки всіх гайок і болтів на відповідні пари за час Θ(n log n)**

**Звіт**

**Підготував:**студент групи К-29  
Дацюк Віталій Олегович

**Київ-2019**

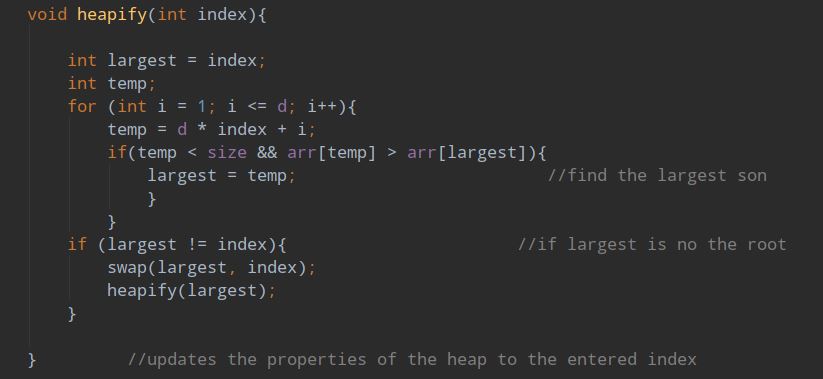
**1.Постановка завдання**Представте d-арну піраміду у вигляді масиву. Розробіть ефективні реалізації процедур Extract\_Max, Insert та Increase\_Key, призначених для роботи з d-арною незростаючою пірамідою.

**2.Основні модулі програми**Піраміда представлена ​​у вигляді масиву arr [0..n-1].Дочірні елементи для вузла з індексом i мають індекси:d \* i + 1, d \* i + 2, ..., d \* i + d.Так як піраміда незростаюча, то для кожної вершини піраміди ключі всіх її нащадків менше або дорівнюють власному ключу вершини.

1. **Оновлення властивості піраміди (heapify)**

Функція отримує індекс вершини **i**, з якої треба почати відновлення. Якщо максимальний ключ має поточна вершина **i**. Інакше знаходиться нащадок з максимальним ключем з індексом **p**. Міняємо місцями елементи **i** і **p**. Далі функція викликається рекурсивно для вузла **p**.

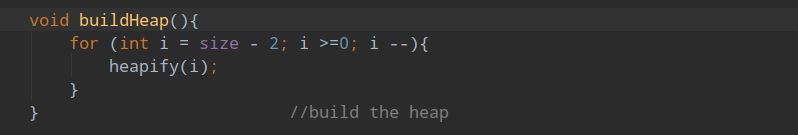
*Складність:O(n) = O (d \* log\_d n).*



1. **Построение кучи (buildHeap)**

Для всіх елементів масиву, починаючи з n-2 - го викликається функція heapify. Коли ми дійдемо до 0-го вузла, піддерева, що починаються у всіх нащадках будуть пірамідами. Отже, якщо ми відновимо властивості для 0-го вузла, то весь вхідний масив буде пірамідою.

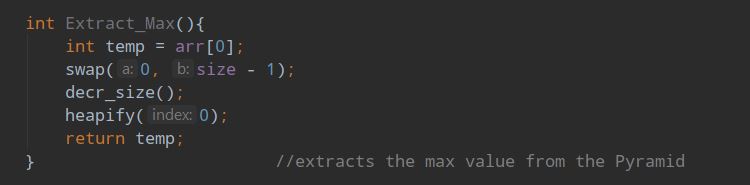
*Складність: O(n) = Σ\_ (i = 1) ^ n (d\*log\_d n).*



1. **Видалення вузла з максимальним ключем (ExtractMax)**

З властивостей піраміди слідує, що індекс вузла з максимальним ключем дорівнює 0. Міняємо 0-й і n-перший елементи місцями. Зменшуємо розмір купи на 1. Викликаємо heapify для вузла 0.

*Складність: O(n)= (d \* log\_d n).*



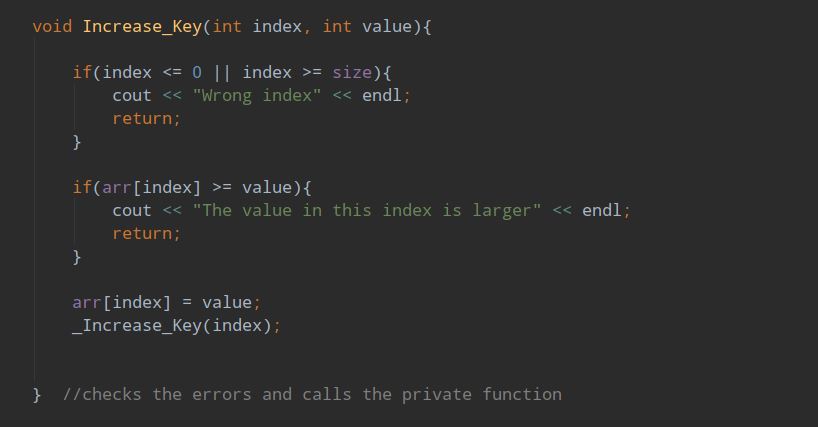
1. **Збільшення ключа (increaseKey)**

Індекс предка вершини i визначається за формулою:

p = (i - 1) / d.

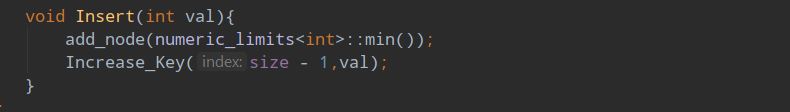
Нехай треба замінити ключ елемента **i** на більше значення **v**. Йдемо по нащадкам вершини **i** до тих пір, поки не прийшли в корінь (вершину 0) або, поки не зустріли вершину, в якій виконуються властивості піраміди, за умови, що значення її ключа буде v. Тобто, якщо значення ключа предка менше **v** ми міняємо поточний ключ на ключ предка і викликаємо цю ж функцію для предка.

*Складність: O(n) = (log\_d n).*



1. **Вставка елемента (insert)**  
   Нехай треба вставити елемент зі значенням **v**. Збільшимо розмір купи на 1. Припускаючи, що до цього там стоїть мінімальне значення для відповідного типу( в нашому випадку це цілочисельний тип даних), викликаємо функцію **increaseKey** з параметром **v** для останньої вершини.

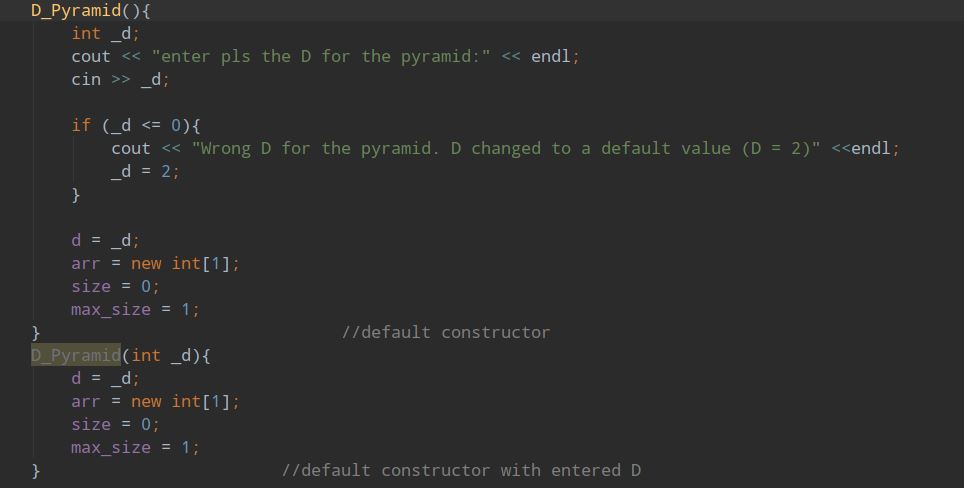
*Складність: O(n) = (log\_d n).*



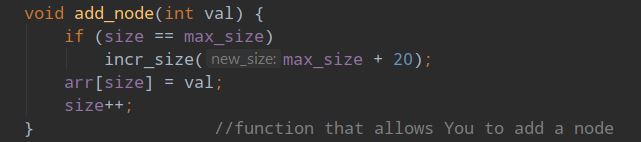
**3.Інтерфейс користувача**

До інтерфейсу користувача належать такі модулі:

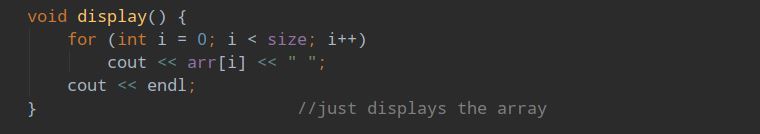
* Усі інші описані вже вище функції.
* Два конструктори. Один приймає параметр значення введеного d для піраміди, інший запитує користувача ввести дане значення

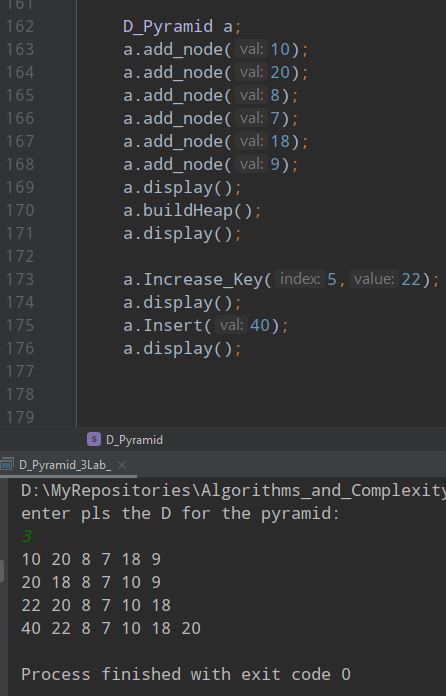


* Функцію додавання елементу до масиву. Функція перевіряє чи достатньо місця для цього елемента, а якщо ні, то виділяє нове



* Функцію виводу масиву на екран



**4.Тестові приклади**

**Література:**

1. <https://www.geeksforgeeks.org/cpp-program-for-heap-sort/>
2. <https://ru.wikipedia.org/wiki/%D0%9A%D1%83%D1%87%D0%B0_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)>