КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

ДОСЛІДЖЕННЯ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ З ВИПАДКОВИМИ ЧИСЛАМИ

Виконав студент 4-го курсу
Назар ГНАТЮК _____

Науковий керівник:
професор, доктор фіз.-мат. наук
Анатолій ПАШКО _____

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики
«____»_____20__ р.,
протокол №____
Завідувач кафедри
Юрій КРАК _____

Київ-2023

ПЕРЕЛІК СКОРОЧЕНЬ ТА ВИЗНАЧЕНЬ

ГВЧ - генератор випадкових чисел.

AES - симетричний алгоритм блочного шифрування.

DES - алгоритм для симметричного шифрования.

NIST - пакет статистичних тестів.

ГПВЧ - генератор псевдовипадкових чисел.

ГВЧ - генератор випадкових чисел.

Windows Api - application programming interfaces.

MIDI - цифровий інтерфейс музичних інструментів.

TRS - роз'єм уживаний зазвичай в аудіо-відео пристроях.

ЗМІСТ

ВСТУП

За останнє десятиліття обчислювальні технології досягли висот, які деякий час здавалися неможливими. Їх щоденне використання пояснюється розширенням сфери застосування, масовим впровадженням, що забезпечує зручність використання та можливість вибору варіантів. У нашому житті, особливо в повсякденному житті програмістів, є Ситуації, що вимагають використання випадкових чи псевдовипадкових чисел. Ця проблема часто виникає Створюйте різні ігрові ситуації, які включають певний набір випадкових результатів. У цьому випадку має сенс використовувати набір спеціальних алгоритмів, результатом роботи яких є послідовність чисел. В даний час попит на генератори випадкових послідовностей із заданим розподілом ймовірностей, як і самі випадкові послідовності, зростає настільки швидко, що за кордоном з'явилися науково-виробничі компанії, котрі займаються виробництвом і продажем великої кількості випадкових чисел. Наприклад, компакт-диски поширюються по всьому світу з 1996 року Компакт-диск «Marsaglia Random Numbers CDROM», який містить 4,8 мільярда «дійсно випадкових» бітів. Практичне значення цієї роботи полягає в тому, що її результати можуть бути використані в криптографії, комп'ютерному моделюванні та інших галузях, де потрібні послідовності випадкових чисел.

Таким чином обрана **тема дослідження є актуальною. Об'єктом дослідження** є послідовності випадкових чисел. **Предметом дослідження** є генерація послідовностей випадкових чисел із використанням аудіоадаптера.

Метою даної кваліфікаційної роботи є створення бібліотеки для генерації випадкових чисел за допомогою аудіоадаптера. Для досягнення поставленої мети пропонуються такі завдання:

- Аналіз сучасних методів генерації послідовностей випадкових чисел;
- Порівняти існуючі реалізації генераторів випадкових чисел;
- Запропонувати функціональні вимоги до майбутнього програмного забезпечення;
- Алгоритми проєктування та структури даних;
- Вибір засобів розробки;
- Створити програмну реалізацію бібліотеки для генерації випадкових чисел.
- Проводити тестування та створювати демонстраційні програми для ілюстрації функціональності бібліотеки;
- Проаналізувати можливість використання згенерованих послідовностей у симетричних криптографічних алгоритмах.

РОЗДІЛ 1.
ГЕНЕРАТОРИ ВИПАДКОВИХ ТА ПСЕВДОВИПАДКОВИХ ЧИСЕЛ

1.1. Принцип роботи генераторів випадкових чисел.

Генератор випадкових чисел (від англ. Random number generator, скорочено RNG) — це алгоритм, який генерує послідовність випадкових чисел, елементи якої фактично не залежать один від одного і підкоряються заданому розподілу (зазвичай рівномірному). [1]

Існує кілька методів генерації послідовностей випадкових чисел: апаратний, табличний і програмний методи генерації послідовностей. Найпоширенішими є програмні генератори, в яких реалізована можливість отримання послідовності випадкових чисел за циклічною формулою. Але варто зауважити, що числа, отримані за допомогою цього методу, насправді є псевдовипадковими, тобто при належній підготовці існує ймовірність «розірвати». Оскільки алгоритми отримання даних послідовності завжди детерміновані, їх ще називають псевдовипадковими. До програмного генератора необхідно пред'явити такі вимоги:

– Створення випадкових чисел, статистично незалежних і рівномірно розподілених в діапазоні від нуля до одиниці, є критично важливим завданням.

– Здатність відтворювати серію довільних чисел.[0, 1];

– Процес передбачає встановлення самостійних прогресій довільних цифр, які називаються потоками випадкових чисел.

– Розподіл ресурсів процесора для роботи генератора програмного забезпечення має бути зведений до мінімуму.

– створення незалежних послідовностей випадкових чисел (потоків).

Важливо визнати, що більшість програмних генераторів генерують випадкові числа, рівномірно розподілені в діапазоні [0,1]. Це необхідно, тому що це дозволяє отримувати випадкові числа, які слідують майже будь-якій моделі розподілу, використовуючи ці уніфіковані числа як основу.

Якість виходу генераторів залежить від статистичних атрибутів послідовностей випадкових чисел, які він створює, зокрема, незалежності та випадковості. Щоб забезпечити властивості послідовностей, для проведення перевірок використовуються статистичні критерії. Щоб програмні генератори могли генерувати псевдовипадкові числа, вони повинні відтворювати ідентичні послідовності чисел при роботі з однаковими початковими умовами та параметрами. Це особливо важливо під час порівняння різних системних моделей і програм для налагодження. У деяких випадках, однак, відтворення послідовностей не є необхідним, наприклад, під час створення комп'ютерних ігор або моделювання систем. У минулому такі ігри, як «Тетріс», кожного разу починалися з однієї і тієї ж послідовності чисел, але згодом ця проблема була вирішена за допомогою комп'ютерного таймера, який отримує початкові значення для величин.

Дослідження й аналіз складної системи часто передбачає створення довгих числових послідовностей. Для ефективного вирішення проблем моделювання життєво важливо використовувати високопродуктивні алгоритми генерації послідовностей, які вимагають мінімальних комп'ютерних ресурсів. Інвестування значних ресурсів у генерацію послідовності може різко сповільнити аналіз складної системи та збільшити час роботи. Деякі недосвідчені дослідники вирішують використовувати той самий генератор під час моделювання складних систем, отримуючи доступ до нього з різних програмних місць. Однак це може призвести до виродження в процесі моделювання через те, що генерована послідовність перевищує аперіодичність. Тому при моделюванні різних випадкових факторів бажано мати незалежні випадкові послідовності (тобто набори значень), згенеровані тим самим алгоритмом, але з різними значеннями параметрів. У більшості генераторів псевдовипадкових чисел хг використовується рекурентний процес xi+1 = f(xi). Найпростішими та найстарішими з таких генераторів є генератори фон Неймана та Метрополіса, робота яких базується на методі середнього квадрата.

1.2. Алгоритми симетричної криптографії та роль генераторів випадкових чисел у їх реалізації.

Генератори випадкових чисел найчастіше використовуються в криптографії, оскільки випадковість і криптографія тісно пов’язані. Важко знайти належним чином розроблену програму шифрування, яка б не використовувала nonces. Криптографічні ключі та їх ініціалізація, характеристики криптографічних хешів та унікальні параметри при роботі з цифровими підписами розробниками системи слід вважати випадковими. При використанні генератора випадкових чисел у криптографічній системі, генератор випадкових чисел повинен відповідати таким вимогам:

- згенерована послідовність повинна мати якомога довший період;
- Згенерована послідовність не повинна мати прихованої періодичності;
- Отримана послідовність повинна мати рівномірний спектр. Розробка надійного та високоякісного генератора випадкових чисел часто залишається поза увагою. Якщо система шифрування може працювати на високому рівні, але ключі, згенеровані генератором паролів, легко зламати, тоді всі інші алгоритми безпеки легко подолати. Деякі продукти використовують такі генератори для генерації ключів за певним шаблоном. Про безпеку в цьому випадку говорити не доводиться. Важливою особливістю є те, що використання одного генератора в одній зоні забезпечує належний рівень захисту, але не в іншій. Тому варто підкреслити важливість криптографічного генератора випадкових чисел - якщо його не спроєктувати належним чином 8, він може легко стати найбільш вразливим елементом у системі. звернемо увагу Плюси апаратної HVC:

- Доступність (дешевизна) – в разі великомасштабного виробництвавідбувається зниження затрат.
- Автоматизованість – при використанні апаратної генерації випадкового числа людина надає їй перевагу, оскільки це більш зручно іавтоматизована процедура є більш безпечною.
- Випадковість – при створенні початкових значень, коли використовуються часткові методи отримання випадковості (переміщення мишки, натискання клавіш), вони є не таким випадковим, так як випадкові числастворені апаратним генератором випадкових чисел. Вихідна послідовність апаратного генератора є істинно випадковою. [2]

Шифрування – практичний спосіб забезпечення надійності збереження та передавання інформації. Сучасні методи шифрування є переважно математичними перетвореннями (алгоритмами), в яких повідомлення, які необхідно зашифрувати, розглядаються як числа або алгебраїчні елементи в деякому просторі. Ці алгоритми відображають область «осмислених повідомлень» в область «безглуздих повідомлень». Повідомлення, що відносяться до числа «осмислених», а також дані, що входять до алгоритму шифрування називаються відкритим текстом (cleartext), а безглузде повідомлення, що є результатом роботи алгоритму шифрування, називається зашифрованим текстом (ci-phertext). Якщо знехтувати сенсом повідомлення, то вхідні дані алгоритму шифрування зручно називати вихідним текстом (plaintext), який не обов'язково має бути осмисленим. Наприклад, вихідне повідомлення може як зашифрований текст, так і випадковий шум. Отже, вихідний текст і зашифрований текст утворюють взаємопов'язану пару понять: перше поняття означає вхідне повідомлення, а друге – результат роботи алгоритму шифрування. [3]

Шифр Вернама (Veraam cipher) – один із найпростіших криптосистем. Припустимо, що повідомлення і ключ є рядки з n двійкових розрядів:

$$m = b1b2 \dots bn \in \{0,1\}\ n$$

$$k = k1k2 \dots kn \in U\ \{0,1\}\ n$$

Варто звернути увагу на $\in U$, число k витягується з рівномірно розподіленої генеральної сукупності. Шифрування виконується біт за бітом, а зашифрований рядок $c = c1c2 \ldots cn$ утворюється шляхом застосування побітової операції XOR («виняткове або») до кожного біту повідомлення і відповідного біту ключа:

$$ci = bi \oplus ki$$

де$1 \leq i \leq n$, а операція $\oplus$ визначена наступним чином:

$\oplus$	0	1
0	0	1
1	1	0

Розшифровка виконується аналогічно до шифрування, оскільки операція $\oplus$ є складанням по модулю 2, а значить, віднімання ідентично додаванню. Переставний шифр (transposition, or permutation cipher) перетворює повідомлення, переставляючи його елементи і не змінюючи їх. Переставні шифри разом із підстановлювальними утворюють важливу сукупність класичних шифрів, які широко застосовуються при створенні сучасних блокових шифрів.

Винахід алгоритму AES призвів до позитивних змін в стані прикладної криптографії.

По-перше, багаторазове шифрування, наприклад, за допомогою потрійного алгоритму DES, з появою алгоритму AES стало зайвим, а змінний 14 розмір ключа і блоків повідомлення, рівний 128, 192 і 256 біт, надає широкий вибір рішень, що забезпечують стійкість в різноманітних додатках. Оскільки в багаторазовому шифруванні використовується значна кількість ключів, створення AES полегшило управління криптографічними ключами і спростило процес розробки протоколів і систем захисту даних.

По-друге, широке використання алгоритму AES призвело до створення нових хеш-функцій, що мають порівнянну ступінь стійкості. Можна вважати шо алгоритми блокового шифрування нагадують хеш-функції.

1.3. Пакет NIST STS

Пакет NIST STS включає 16 статистичних тестів, призначених для перевірки припущень щодо випадковості послідовностей довільної довжини, створених генераторами випадкових чисел. Усі тести призначені для виявлення різних недоліків у випадковості. Основним принципом тесту є перевірка нульової гіпотези H0, тобто перевірена послідовність є випадковою. Альтернативною гіпотезою для Na є припущення, що тестова послідовність не є випадковою. Залежно від результатів застосування кожного тесту нульова гіпотеза або приймається, або відхиляється. Рішення про те, чи є дана послідовність 0 і 1 випадковою, базується на сумі всіх результатів тестування. [4]

Процедура перевірки однієї двійкової послідовності S виглядає наступним чином.

1. Ми пропонуємо нульову гіпотезу H0 - ми припускаємо, що задана двійкова послідовність S є випадковою. 1. Ми пропонуємо нульову гіпотезу H0 - ми припускаємо, що задана двійкова послідовність S є випадковою.

2. За послідовністю S обраховуємо статистику тесту c (S).

3. З використанням спеціальної функції і статистики тесту вираховується значення ймовірності $P = f\ (c\ (S))$, $P \in [0, 1]$.

4. Значення ймовірності P порівнюється з рівнем значущості □, □ ∈ [0,001, 0,01]. Гіпотеза H0 приймається, якщо □ ≥ □. В іншому випадку прийміть альтернативну гіпотезу.

Як згадувалося раніше, пакет включає 16 статистичних тестів. Але насправді, відповідно до різних вхідних параметрів, розраховується 189 значень ймовірності P, які можна розглядати як результат індивідуальної інспекційної роботи. У таблиці 1.1 наведено зведені дані для всіх тестів із зазначенням кількості обчислених значень ймовірності P, фізичного значення статистики тесту та дефектів, які тести повинні були виявити.

№ п/п	Статистичний тест	Кількість значень ймовірності P	Статистика тесту c(S)	Дефект, що виявляється
1	2	3	4	5
1	Частотний (монобітний) тест	1	Нормалізована абсолютна сума значень елементів послідовності	Занадто багато нулів або одиниць в послідовності
2	Частотний тест всередині блоку	1	Міра узгодження спостереженої кількості одиниць всередині блоку з теоретично очікуваним.	Локалізовані відхилення частоти появи одиниць в блоці від ідеального значення 1 /2 .
3	Перевірка накопичених сум	2	Максимальне відхилення значення накопиченої суми елементів послідовності від початкової точки відліку (точка 0)	Велике значення одиниць або нулів спочатку або в кінці двійковій послідовності.
4	Перевірка серій	1	Загальна кількість серій на всій довжині послідовності	Занадто швидка або надто повільна зміна знаку в ході генерації послідовності
5	Перевірка максимальної довжини серії в блоці	1	Міра узгодження спостережуваного значення максимальної довжини одиничної серії з теоретично очікуваним значенням.	Міра узгодження спостережуваного значення максимальної довжини одиничної серії з теоретично очікуваним значенням.
6	Перевірка рангу двійкової матриці	1	Міра узгодження спостережуваного значення рангів різного порядку з теоретично очікуваним	Відхилення емпіричного закону розподілу значень рангів матриці від теоретичного, що вказує на залежність символів в послідовності
7	Спектральний тест на основі дискретного перетворення Фур'є	1	Нормалізована різниця між спостережуваним і очікуваним кількістю частотних компонент, які перевищують 95% граничний рівень.	Виявлення періодичних складових (трендів) в двійковій послідовності.
8	Перевірка перекривних шаблонів	1	Міра узгодження спостереженої кількості перекривних шаблонів в послідовності з теоретичним значенням	Велика кількість m-бітових серій з одиниць в послідовності.
9	Універсальний тест Маурера	1	Сума логарифма відстані між lбітними шаблонами	Стисливість послідовності.
10	Ентропійний тест	1	Міра узгодження спостережуваного значення ентропії джерела з теоретично очікуваним для випадкового джерела	Нерівномірність розподілу mбітових слів в послідовності (регулярність властивостей джерела).
11	Перевірка випадкових відхилень	8	Міра узгодження спостереженої кількості візитів при випадковому блуканні в заданому стані всередині циклу з теоретично очікуваним.	Відхилення від теоретичного закону розподілу візитів в конкретний стан при випадковому блуканні
12	Перевірка випадкових відхилень (варіант)	18	Загальна кількість візитів в заданий стан при випадковому блуканні	Відхилення від теоретично очікуваної загальної кількості візитів при випадковому блуканні в заданому стані
13	Послідовний тест	2	Міра узгодження спостереженої кількості всіх зустрічних варіантів m-бітових шаблонів з теоретично очікуваним	Нерівномірність розподілу mбітових слів в послідовності.
14	Перевірка стиснення по алгоритму Лемпеля-Зива	1	Кількість непересічних різних слів в послідовності	Велика ступінь стиснення тестованої послідовності в порівнянні з очікуваним ступенем стиснення для випадкової послідовності

15	Перевірка неперекривних шаблонів	148	Міра узгодження спостереженої кількості неперіодичних шаблонів в послідовності з теоретичним значенням.	Велика кількість заданих неперіодичних шаблонів в послідовності.
16	Перевірка лінійної складності	1	Міра узгодження спостереженої кількості подій, які полягають у появі фіксованої довжини еквівалентного ЛРР для заданого блоку з теоретично очікуваним.	Відхилення емпіричного розподілу довжин еквівалентних ЛРР для послідовності фіксованої довжини від теоретичного закону розподілу для випадкової послідовності, що вказує на недостатню складність тестованої послідовності.

Таблиця 1.

Тому в результаті тестування двійкової послідовності Сформуйте вектор значень ймовірностей P = {P1, P2, ..., P189}. аналізувати Компонент Pi цього вектора дозволяє вказати конкретний дефект Перевірте випадковість послідовності

1.4. Методика тестування генератора.

Розглянутий пакет статистичних тестів доступний та використовується для вирішення таких задач:

- виявлення Генератора Випадкових Чисел (ГПВЧ), які формують «погані» двійкові послідовності;
- розробка нових Генератора Випадкових Чисел (ГПВЧ);
- перевірка коректності реалізації Генератора Випадкових Чисел (ГПВЧ);
- вивчення генераторів, описаних в стандартах;
- дослідження ступенявипадковості реально використовуваних

Генератора Випадкових Чисел (ГПВЧ).

При вирішенні перерахованих завдань застосовують таку методику тестування генераторів.

1. З множини апаратних або програмних генераторів вибирають генератор G, який необхідно оцінити і прийняти рішення про те, що він формує випадкові послідовності. Генератор повинен генерувати двійкову послідовність $S = \{s1, s2, \ldots, sn\}$, $s\ i \in \{0,1\}$ довільної довжини n.

2. Для фіксованого значення n сформуйте набір із m двійкових послідовностей:

$S1 = \{s1, s2, \ldots, sn$

$\}$

$S2 = \{s1, s2, \ldots, sn$

$\}$

… … … … … … … … … …

$Sm = \{s1, s2, \ldots, sn$

$\}$

Таким чином, для тестування необхідно сформувати вибірку обсягом $N = m \times n$.

3. Кожна послідовність тестується за допомогою набору NIST STS. Таким чином формується статистичний портрет наступних типів генераторів

№ тесту j № пос-ті i	1	2	…	q
$S1$	$P1,1$	$P1,2$	…	$P1,q$
$S2$	$P2,1$	$P2,2$	…	$P2,q$
…	…	…	…	…
Sm	$Pm,1$	$Pm,2$	…	Pm,q

$$\begin{matrix} P11 & P12 & \ldots & P1q \\ P21 & P22 & \ldots & P2q \\ \vdots & \vdots & \ddots & \vdots \\ Pm1 & Pm2 & \cdots & Pmq \end{matrix}$$

Далі вводиться поняття статистичного портрета генератора.

Статистичний профіль генератора — це матриця □ × □, де m — кількість перевірених бінарних послідовностей, а q — кількість статистичних тестів, які використовуються для перевірки кожної послідовності. матричний елемент □□□ ∈ [0, 1], де □ = 1 , □ , □ = 1 , □ — значення ймовірності, отримане з j-го тесту для i-го тесту послідовності.

4. На основі отриманого статистичного профілю визначте частку послідовностей, які проходять кожен статистичний тест. Для цього встановлюють рівень значущості □ ∈ [0,001, 0,01] і обчислюють для кожного q-тесту значення P ймовірності перевищення заданого рівня □, коефіцієнт детермінації

$$rj = \frac{\#\{\mathrm{Pij} \geq \alpha | \mathrm{i} = 1,2,\ldots,\mathrm{m}\}}{\mathrm{m}}.$$

В результаті формується вектор коефіцієнтів $R = \{r1, r2, \ldots, rq\}$, елементи якого характеризують, в відсотковому співвідношенні, проходження послідовності Si всіх статистичних тестів. Правило 1. Вважається, що генератор G пройшов тестування по j-му тесту, якщо значення коефіцієнта rj знаходиться всередині довірчого інтервалу [r_{max}, r_{min}]. Межі довірчого інтервалу визначаються за виразом

$$r\max(\min) = \hat{p} \pm 3 \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{\mathrm{m}}}$$

де $\hat{p}$= 1 – α

1.5. Інтерфейс звукового адаптеру

Аудіоадаптери (звукові карти, звукові контролери, звукові карти) — це спеціальні електронні плати, які використовують апаратне та програмне забезпечення комп’ютера для запису та відтворення звуку. Звукова карта складається з двох або більше перетворювачів інформації:

• Аналоговий цифровий – перетворення аналогових звукових сигналів, таких як мова, музика, шум, у цифрові двійкові коди та збереження їх на магнітних носіях;

• Цифро-аналоговий - зворотне перетворення звуку, що зберігається в цифровій формі, в аналоговий сигнал, який потім можна відтворити за допомогою акустичної системи, вокодера або навушників. Професійні аудіоадаптери здатні до складної обробки звуку, забезпечують стереозвук, містять власне програмне забезпечення та зберігають масиви тембрів для різних інструментів. Звукові файли зазвичай дуже великі. Таким чином, трихвилинний стереозвуковий файл потребує приблизно 30 Мб пам'яті.

На сучасних ПК апаратна підтримка звуку має одну з таких форм:

• звукова карта, встановлена на роз'єм шини PCI, PCIe або підключена до порту USB або IEEE 1394;

• AC'97 або HD Audio чіп на системній платі.

До основних характеристик звукового адаптера належать якість звуку (стерео або моно, використання цифрової системи фільтрації, частотний діапазон тощо), бітрейт шини даних, кількість каналів відтворення та запису, наявність синтезатора та кількість його Звукові пристрої, Роз’єми використовуються незалежно від місця розташування Роз'єми для мікрофонів і акустичних систем, які також можуть містити роз'єми для підключення MIDI-пристроїв. З програмного погляду звуковіадаптери потребують використання драйверів, вбудованих в операційну систему чи тих, що містяться в конкретних програмах.

Класичний аудіоадаптер містить звуковий чіп, у якому цифро-аналоговий перетворювач перетворює згенерований або цифровий звук в аналоговий формат. Використовуйте стандартні роз’єми, такі як TRS або RCA, щоб направляти вихідний сигнал на підсилювачі, навушники або інші зовнішні пристрої. Якщо роз’єми на задній панелі комп’ютера перевищують кількість або розмір, їх можна зняти окремо. Відтворення звуку здійснюється за допомогою багатоканального цифро-аналогового перетворювача, який може одночасно відтворювати звуки різної висоти та гучності, звукові ефекти в реальному часі. Багатоканальне відтворення звуку використовується для синтезу звуку за допомогою бібліотек цифрових інструментів, які використовують невеликий обсяг постійної пам’яті або флеш-пам’яті та містять звукові зразки з інструментів MIDI. Іншим способом синтезування звуків є використання «аудіо-кодеків», даний спосіб потребує відповідного програмного забезпечення, сумісності з MIDI, та багатоканальної емуляції.

Переважна більшість аудіоадаптерів мають роз'єми для вхідних (input) та вихідних (output) сигналів. Часто звукові карти мають два вхідні роз’єми. Linein роз’єм – використовується для підключення пристроїв високого рівня сигналу (наприклад, магнітофон). Цифрова карта проводить дискретизацію даного сингала і зберігає його на жорсткому диску комп'ютера (збережений сигнал в подальшому можна оброблювати). Роз’єм microphone – призначений для підключення пристроїв низького рівня (наприклад, мікрофон). Звукові адаптери, що використовуються на професійному рівні мають декілька вхідних роз’ємів, що дозволяє проводити багатоканальний запис звуку.

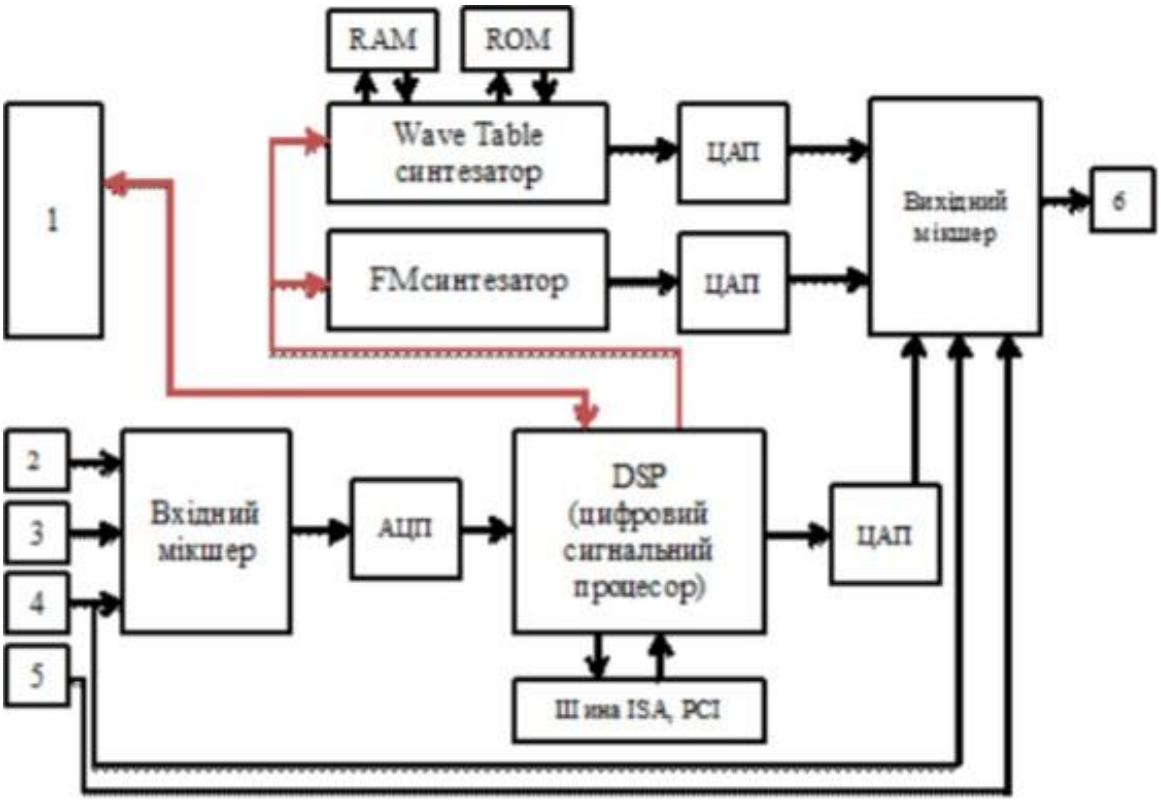

Рис.1.1Блок-схема типової звукової карти

ВИСНОВКИ ДО РОЗДІЛУ 1

Необхідність генерувати випадкові числа виникає в багатьох прикладних задачах. Завдання, які пред'являють найвищі вимоги до якості випадкових чисел, пов'язані з комп'ютерним моделюванням і криптографією. Криптографічні алгоритми використовують генератори випадкових чисел для шифрування ключів, тому для них якість використовуваного HVDC є критичним аспектом, оскільки існує ризик злому ключів шифрування, в результаті чого конфіденційна інформація буде втрачена або зловмисники отримають доступ до системи. і т.д. У цих випадках У цьому випадку недоцільно використовувати стандартний детермінований генератор псевдовипадкових чисел, який надається багатьма бібліотеками, через ризик повторного генерування послідовності. Атаки на HVDC почастішали, злом алгоритмів генерації чисел набирає обертів, тому необхідно створювати такі послідовності, які не зможуть підхопити, тобто розгадати алгоритм і відтворити попередні умови, тому що тепер це можливо кожному отримати інформацію про алгоритми, які використовуються для генерації випадкових чисел, а також про алгоритми злому для цих алгоритмів. Тому необхідно створити нові алгоритми, які використовують джерела ентропії. На мій погляд, найкращим вибором джерела ентропії є шум звукової карти, оскільки точно відтворити записаний звук практично неможливо. Для кожного ноутбука, кожного аудіоадаптера звукова доріжка буде різною, тому результуюча послідовність буде унікальною та не повторюваною. Генератор, що розробляється, відноситься до четвертого типу надвисокої частоти, який використовує особливості роботи ПК.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ.

2.1. Алгоритм роботи бібліотеки.

Програма викликає функцію зі зв’язаної бібліотеки та передає їй необхідний список аргументів, указаних користувачем за бажанням. Кількість елементів, мінімальний і максимальний елементи (бажаний діапазон, який використовуватиме користувач) передаються як параметри. Після отримання параметрів програма переходить до бібліотеки, де записується звук, який буде використовуватися для генерації послідовності випадкових чисел. Запис відбувається у програмі допоміжного вікна, з якого бібліотека отримує масив із записами. Наступним кроком є перезапис масиву в буфері пам'яті на створений динамічний масив. Під час перезапису елементи звукового буфера перевіряються на повторюваність. Ця перевірка пов’язана з характеристиками файлу хвилі та фактично записаного звуку. Після завершення операції збору звуку в бібліотеці буде відкрито файл із записаним звуком, і відповідно до побажань користувача буде створено масив відповідного розміру. Щоб програма вивела в якості кінцевого результату масив чисел у заданому діапазоні, необхідно шукати мінімальний і максимальний елементи масиву. Нормалізує елементи до вказаного діапазону за допомогою масиву звуків і елементів, знайдених відповідно до розгорнутої форми в масиві. Нормалізований масив передається користувачеві в основну програму, і користувач використовує згенеровану послідовність випадкових чисел відповідно до своїх потреб. Алгоритм бібліотеки, показаний на загальній блок-схемі бібліотеки, насправді є блок-схемою алгоритму, який нормалізує записаний користувачем шум до потрібного діапазону.

Рис.2.1. Блок-схема загального алгоритму роботи

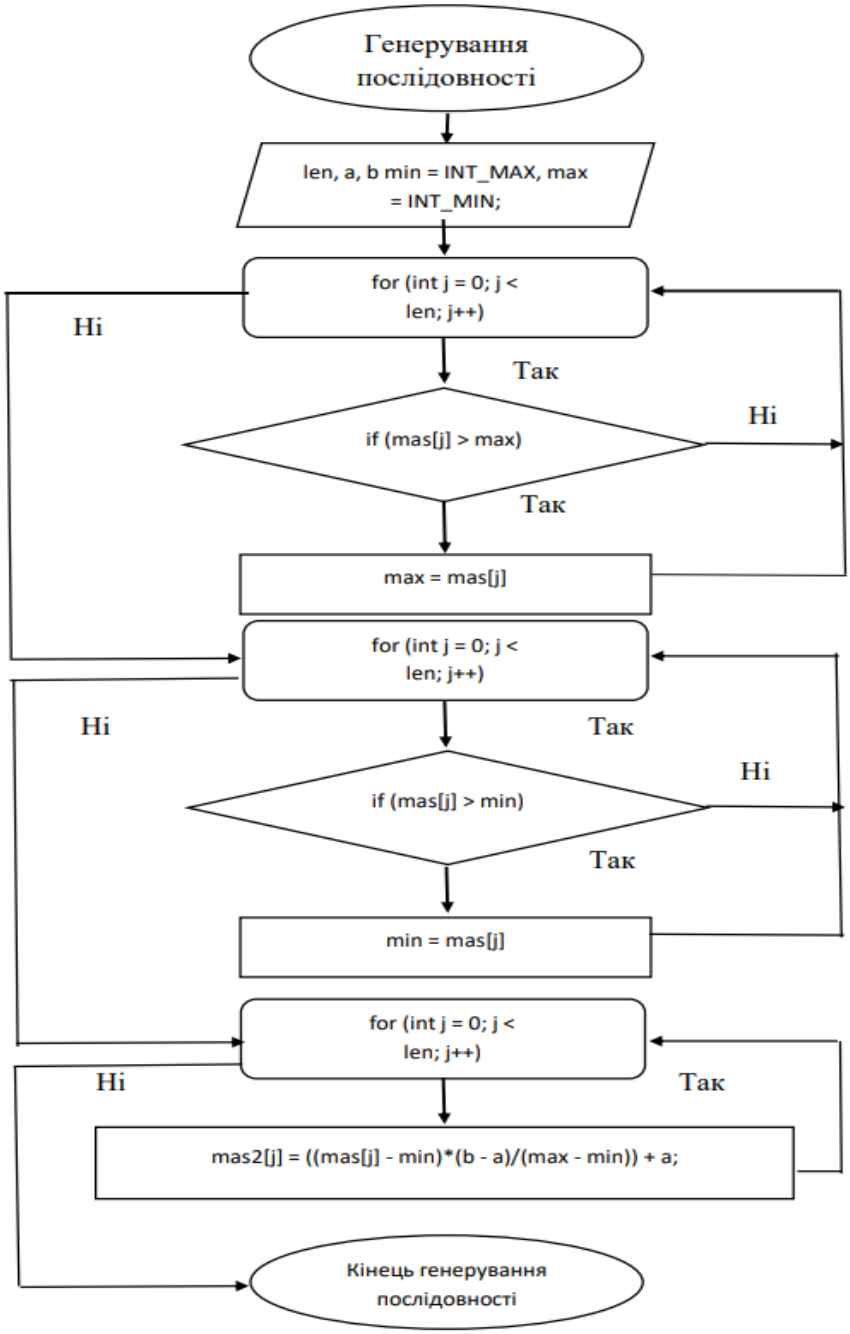

Рис.2.2. Блок-схема генерування послідовності випадкових чисел

2.2. Етапи створення бібліотеки

Щоб реалізувати та продемонструвати бібліотеку генерації випадкових чисел на основі шуму звукової карти за допомогою Visual Studio 2017, необхідно виконати такі кроки:

1) Створення проєкту статичної бібліотеки.

2) Додайте клас до статичної бібліотеки.

3) Створіть консольну програму, яка посилається на статичну бібліотеку.

4) Використовуйте функції статичної бібліотеки в додатку. Розгляньте можливість створення проєкту статичної бібліотеки. Для цього потрібно:

1) У рядку меню виберіть «Файл», «Створити», «Проєкт».

2) У лівій області діалогового вікна «Створити проєкт» розгорніть «Встановлено, шаблони, Visual C++» і виберіть «Win32».

3) У центральній області виберіть «Консольний додаток Win32».

4) Вкажіть назву створеного проєкту (створена бібліотека матиме назву «Бібліотека») і назва рішення («Бібліотека_ср»).

5) На сторінці «Параметри програми» в розділі «Тип програми» необхідно вибрати статичну бібліотеку. Наступним кроком буде додавання класів до створеної бібліотеки.

Для цього:

1) Ми створюємо заголовний файл для нового класу. Відкрийте контекстне меню проєкту в провіднику рішень і виберіть «Додати, новий елемент». У діалоговому вікні «Додати новий елемент» виберіть «Файл заголовка» (.h). Вказує назву файлу заголовка "srgenerate.h".

2) Ми додали клас MyGenerate, який оголошує функцію для генерації випадкових чисел, які в майбутньому будуть викликатися з програм, що використовують бібліотеку.

3) Ми оголошуємо функцію, вказуємо потрібний тип і оголошуємо параметри, які будуть їй передаватися при виклику.

4) Створюємо вихідний файл для нового класу. Обираємо «Файл

C++(.cpp)». Вказуємо ім’я створюваного файлу («source.cpp»).

5) Реалізуємо генератор випадкових чисел за допомогою створеного файлу.

6) Компілюємо статичну бібліотеку. В результаті буде створена статична бібліотека, яка може використовуватись іншими програмами.

Щоб продемонструвати, як працює бібліотека, вам потрібно створити консольну програму, яка посилається на статичну бібліотеку. Для цього потрібно:

1) У рядку меню вибрати «Файл», «Створити», «Проєкт».

2) У лівій області діалогового вікна «створити проєкт» розгорнути «Встановлені, Шаблони, Visual C++» і потім обрати «Win32».

3) У центральній області виберіть «Консольний додаток Win32».

4) Вкажіть назву програми «Приклад». У списку поруч із полем «Рішення» виберіть «Додати до рішення». Тож до рішення, що містить статичну бібліотеку, буде додано новий проєкт.

5) У розділі «Тип програми» на сторінці «Параметри програми» необхідно вибрати консольну програму.

Використання функціональності зі статичної бібліотеки в додатку:

1) Щоб використовувати процес генерації випадкових чисел зі статичної бібліотеки, ви повинні посилатися на цю бібліотеку. Для цього необхідно відкрити контекстне меню проєкту в Solution Explorer і вибрати пункт «Посилання». У діалоговому вікні «Сторінки властивостей» розгорніть вузол «Загальні властивості», виберіть «.NET Framework and Links» і натисніть кнопку «Додати нове посилання».

2) Діалогове вікно «Додати нове посилання» містить список бібліотек, для яких можна вставити посилання. На вкладці «Проєкти» перелічено всі проєкти поточного рішення та бібліотеки (якщо такі є). На цій вкладці встановіть прапорець «Library_sr» і натисніть кнопку «ОК».

3) Щоб створити посилання на файл заголовка "srgenerate.h", вам

потрібно змінити шлях до каталогу файлу посилання. Для цього в діалозі бібліотеки «Вікно властивостей» по черзі розгортаємо вузли «Властивості конфігурації, C/C++» і виберіть «Загальні». в польових умовах «Додатковий каталог для включних файлів» вказує шлях до каталогу бібліотеки.

4) Клас MyGenerate тепер можна використовувати в програмі відповідно до встановленого синтаксису та параметрів.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другій частині кваліфікаційної роботи формулюється загальний план роботи запропонованої бібліотеки, визначаються завдання, які необхідно виконати. Склалися етапи бібліотечної роботи, в яких надати:

1) Запис. Взаємодія зі звуковою картою за допомогою інструментів Windows API. Запишіть отриманий звук у файл, який бібліотека використовуватиме як основу для створення справді випадкової послідовності. Масив динамічних звуків фільтрується, щоб видалити дублікати (ідентичні елементи поруч один з одним) перед записом у файл, оскільки в іншому випадку результуюча послідовність, імовірно, міститиме багато дублікатів.

2) Елементи нормалізуються до вказаного діапазону. Файл, отриманий під час виклику програми запису з бібліотеки, використовується для розміщення його елементів у визначених користувачем діапазонах.

3) Клієнтська програма. Скористайтеся бібліотекою, продемонструйте її функціональність і перевірте інформацію, що міститься в записаних звукових файлах.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ.

1.1. Опис основних функцій і структур даних.

Розроблена програма, яка відповідає за запис звуків, які використовуватимуться в послідовностях майбутніх поколінь, розроблена з використанням Windows API з причин, описаних у другому розділі цієї статті. Ця програма використовує структури даних .

```
BOOL CALLBACK DlgProc (HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
 static BOOL bRecording, bEnding;
 static DWORD dwDataLength, dwRepetitions = 1;
 static HWAVEIN hWaveIn;
 static HWAVEOUT hWaveOut;
 static PBYTE pBuffer1, pBuffer2, pSaveBuffer,
pNewBuffer;
 static PWAVEHDR pWaveHdr1, pWaveHdr2;
 static TCHAR szOpenError[] = TEXT ("Помилка при
відкритті звукового файлу");
 static TCHAR szMemError [] = TEXT ("Помилка при
виділенні пам’яті");
 static WAVEFORMATEX waveform;
```

Оскільки користувачеві не потрібно відображати вікно (так як вся програма побудована на системі повідомлень, повністю відмовитися від створення вікна неможливо), його необхідно приховати за допомогою функції:

```
ShowWindow (hwnd, SW_MINIMIZE);
```

Де SW_MINIMIZE відповідає за видимість вікна.

Наступним кроком є відкриття пристрою введення за допомогою функції waveInOpen, в її параметрах вказуємо формат аудіопотоку і викликаємо метод повідомлення про виконання операції.
Створюємо в програмі два буфери з заголовками, заповнюємо заголовки буферів за встановленими правилами.

```
pWaveHdr1 = (PWAVEHDR)malloc(sizeof(WAVEHDR));
pWaveHdr2 = (PWAVEHDR)malloc(sizeof(WAVEHDR));
```

Також необхідно виділити пам'ять для масиву, в якому зберігатиметься буфер:

```
pSaveBuffer = (PBYTE)malloc(1);
```

Підготовлюємо створені буфери до передачі драйверу за допомогою функції waveInPrepareHeader:

```
waveInPrepareHeader(hWaveIn, pWaveHdr1, sizeof(WAVEHDR));
waveInPrepareHeader(hWaveIn, pWaveHdr2, sizeof(WAVEHDR));
```

Після підготовки буферів передаємо буфери до звукової карти задопомогою функції waveInAddBuffer:

```
waveInAddBuffer(hWaveIn, pWaveHdr1, sizeof(WAVEHDR));
waveInAddBuffer(hWaveIn, pWaveHdr2, sizeof(WAVEHDR));
```

Запис потоку запускається функцією waveInStart. Після цього драйвер запускає аналого-цифровий перетворювач і починає заповнювати буфер для передачі.

Якщо буфер заповнений, програма може обробити дані. Щоб відстежувати,

коли буфер заповнюється, обробіть повідомлення MM_WIM_DATA, яке надсилає драйвер, коли він заповнюється Буфер вікна під час накопичення звукової доріжки:

```
pNewBuffer = (PBYTE) realloc (pSaveBuffer, dwDataLength +
 ((PWAVEHDR)lParam) -> dwBytesRecorded);
```

Далі необхідно знову використати функцію waveInAddBuffer, щоб передати буфер драйверу для запису. Процес зупиняється (призупиняється), коли буфер заповнюється вказаною максимальною кількістю елементів. Після запису, а відповідно, перед закриттям програми, необхідно вивести отриманий масив у файл, попередньо обробивши його, щоб програма могла обробляти файли wave. Створіть допоміжний масив, довжина якого відповідає довжині буфера із записаним звуком. У нього записуються лише ті елементи, які не є дублікатами, тобто елементи, що стоять перед ним у черзі, що очікує на запис, йому не дорівнюють. У цьому випадку буде виключена ситуація, коли у вихідному масиві занадто багато повторів елементів. Цей цикл відповідає за:

```
while (j < ARRAY_LEN)
{
     if (pSaveBuffer[i] != pSaveBuffer[i - 1])
     {
        mas[j] = pSaveBuffer[i];
        j++;
     }
     i++;
}
```

де pSaveBuffer — буфер, що містить елементи записаного звуку, а mas — новий масив, до якого потрапляють лише вибрані елементи. Цикл завершується, коли весь буфер буде просканово і не залишиться

жодного елемента.

Згенеровані елементи записуються у файл для подальшого використання бібліотекою генерації випадкових чисел:

```
fwrite(mas, sizeof(char), ARRAY_LEN, f);
```

Після перенесення масиву у файл звільніть масив і поверніться до головної програми. Оскільки більше інформації з додатка не надходитиме, і немає необхідності залишати його відкритим, ми припиняємо його роботу, відправивши повідомлення:

```
SendMessage(hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L);
```

при виклику якого відбувається звільнення буферів та заголовків, викликається функція, що коректно завершує роботу додатку:

```
free(pWaveHdr1);
free(pWaveHdr2);
free(pSaveBuffer);
EndDialog(hwnd, 0);
```

Далі розглянемо, власне, бібліотеку та її складові. Відповідно до етапівстворення бібліотеки, що описані у пункті 2.1, розглянемо створення заголовку для класу, що використовуватиме бібліотека:

Де оголошено, що майбутні функції будуть заповнені та використані в майбутньому. Функція є статичною і має тип unsigned char*, оскільки вона повертає в основну програму масив, що містить згенеровану послідовність елементів.

```
namespace Generate
{
       class MyGenerate
       {
          public:
               static unsigned char* generateNumbers(int len,
int a, int b);
       };
}
```

у функцію будуть передані такі параметри:

• len – кількість елементів (розмір масиву);

• a – початок інтервалу (мінімальний елемент);

• b — кінець інтервалу (найбільший елемент).

Далі створіть вихідний файл для нового класу. Першою його функцією є запуск програми для запису звуку, оскільки тільки після цього створюється файл нормалізації елемента. Для цього викликаємо функцію:

```
WinExec(«Record.exe», SW_SHOW);
```

Після завершення роботи програми запису в бібліотеку нормалізуйте елементи, тобто приведіть зчитані з файлу елементи в необхідний діапазон і поверніть нормалізовані елементи користувачеві. Спочатку необхідно шукати найбільший і найменший елементи зчитаної послідовності, оскільки вони необхідні формулі для приведення послідовності в потрібну послідовність:

```
for (int j = 0; j <len; j++)
{
     if (mas[j] > max)
     {
          max = mas[j];
     }
}
for (int j = 0; j < len; j++)
{
     if (mas[j] < min)
     {
          min = mas[j];
     }
}
```

Далі нормалізуйте елементи відповідно до формули та пе

```
for (int j = 0; j < len; j++)
      {
         mas2[j] = ((mas[j] - min)*(b - a) / (max - min)) + a;
      }
```

ре за п

ишіть нормалізовані елементи в новий масив:

Після перезапису закрийте файл, з якого були зчитані елементи, і передайте отриманий масив програмі, яка звертається до бібліотеки генерації випадкових чисел.

1.2. Дослідження характеру розподілу випадкових чисел.

У ході дипломної роботи досліджено розподіл випадкових чисел з використанням шуму звукової карти як джерела ентропії. Для вивчення характеру розподілу було використано 10 000 випадкових чисел від 1 до 100. Кількість елементів відкладено вздовж осі OY, а елементи – вздовж осі OX, тобто з використанням шуму карти послідовності, створеного звуком, як джерела ентропії. Тому послідовність нормалізується до зазначеного діапазону. Щоб побудувати графік, необхідно порахувати кількість входжень кожного значення в послідовності. Розрахунок здійснюється за формулою: =COUNTED($A$1:$NTP$1; B3), Серед них $A$1:$NTP$1 — нормалізований список елементів звукового масиву, а B3 (B4, B5...B102) — значення елемента 1 (2, 3...100) відповідно) які вибрано в списку.

Для демонстрації роботи бібліотеки були створені цифрові послідовності та записані для них звуки в різних умовах. Наприклад, на записах, зроблених у галасливій аудиторії, видно наступну послідовність цифр (рис. 3.1):

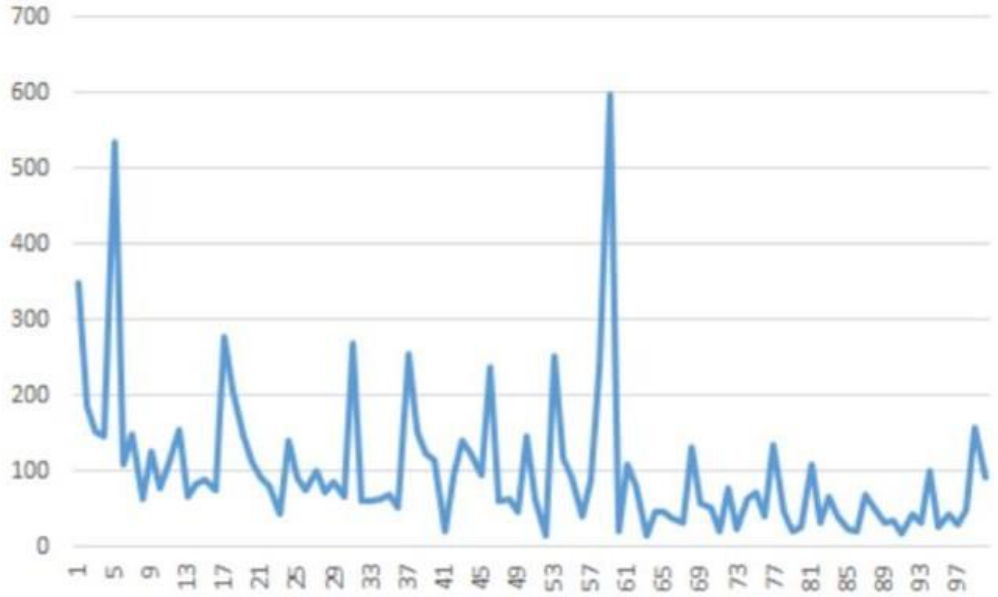

Рис.3.1 Розподіл звуку у шумній аудиторії

При записі звуку в тихій аудиторії, графік розподілу відповідної послідовності такий. (Рис.3.2)

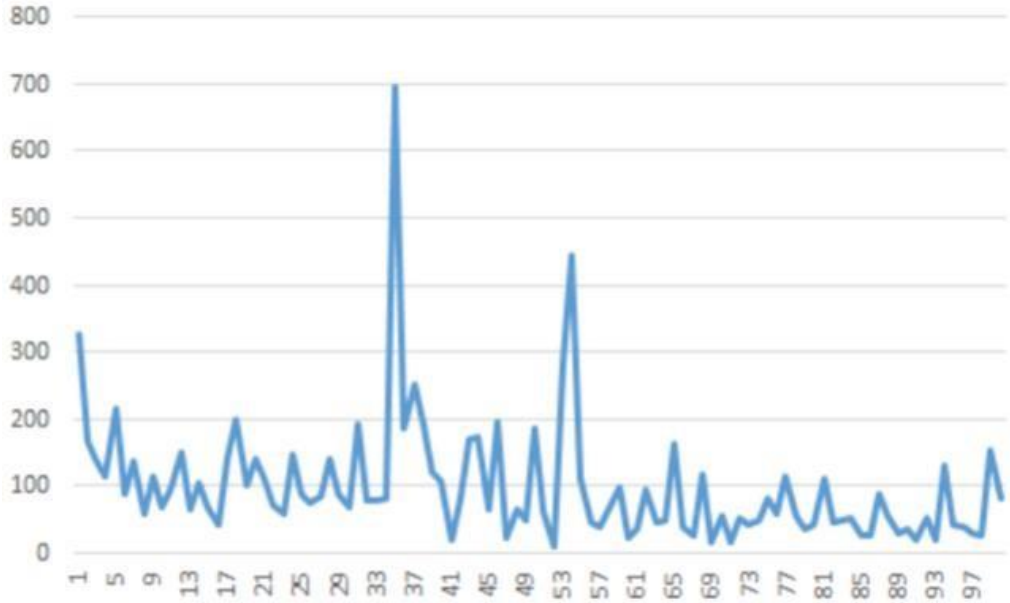

Рис.3.2 Розподіл звуку в тихій аудиторії

Оскільки звук, отриманий бібліотекою, що використовується для створення послідовності випадкових чисел, є випадковим, важко відстежити характеристики отриманого звуку. Тому виникає проблема перевірити, до якого типу розподілу належить отриманий ряд. Згідно з визначенням рівномірного розподілу передбачається, що отримана послідовність відповідає рівномірному розподілу: неперервна величина X рівномірно розподілена на інтервалі (a, b), якщо всі її можливі значення знаходяться в цьому інтервалі, і його розподіл імовірності щільності постійний.

Групи	Сер. інт.	К-сть, f_i	$x_i * f_i$	Нак. част, S	$\lvert x - x_{cp}\rvert * f_i$	$(x - x_{cp})^2 * f_i$	Частота, f_i/f
1 - 10	5.5	12.35	67.925	12.35	514.268	21414.663	0.115
10 - 20	15	13.98	209.7	26.33	449.333	14442.039	0.13
20 - 30	25	10.53	263.25	36.86	233.146	5162.102	0.098
30 - 40	35	12.52	438.2	49.38	152.007	1845.525	0.117
40 - 50	45	9.28	417.6	58.66	19.869	42.542	0.0864
50 - 60	55	9.53	524.15	68.19	74.895	588.596	0.0887
60 - 70	65	13.27	862.55	81.46	236.988	4232.34	0.124
70 - 80	75	5.21	390.75	86.67	145.145	4043.578	0.0485
80 - 90	85	8.1	688.5	94.77	306.657	11609.703	0.0754
90 - 100	95	12.64	1200.8	107.41	604.937	28951.601	0.118
Всього		107.41	5063.425		2737.243	92332.689	1

Для оцінки ряду розподілу знайдемо такі показники:

1) Середня зважена (вибіркова середня):

$$\bar{x} = \frac{\Sigma x_i * fi}{\Sigma f_i} = \frac{5063.425}{107.41} = 47.14$$

2) Розмах варіації − різниця між максимальним і мінімальним значеннями ознаки первинного ряду:

$$R = xmax - xmin = 100 - 1 = 99$$

3) Дисперсія − характеризує міру розкиду близько її середнього значення (міра розсіювання, тобто відхилення від середнього):

$$D = \frac{(x_i - \bar{x})2f_i}{\sum fi} = \frac{92332.69}{107.41} = 859.63$$

4) Незміщена оцінка дисперсії:

$$S^2 = \frac{(x_i - \bar{x})2fi}{\sum fi - 1} = \frac{92332.69}{107.41} = 867.71$$

i	n_i	n_i^*	$n_i - n_i^*$	$(n_i - n_i^*)^2$	$(n_i - n_i^*)^2/n_i^*$
1	12,35	14,4266	-2,08	4,31	0,29891087
2	13,98	9,5179	4,46	19,91	2,091883337
3	10,53	9,5179	1,01	1,02	0,107623153
4	12,52	9,5179	3,00	9,01	0,946911021
5	9,28	9,5179	-0,24	0,06	0,005946313
6	9,53	9,5179	0,01	0,00	1,53826E-05
7	13,27	9,5179	3,75	14,08	1,479134516
8	5,21	9,5179	-4,31	18,56	1,949800104
9	8,10	9,5179	-1,42	2,01	0,21122731
10	12,64	8,3798	4,26	18,15	2,165839762
Всього	107,41				9,257291769

5) Визначити межі критичної ділянки. Оскільки статистика Пірсона вимірює різницю між емпіричним і теоретичним розподілами, більші значення Kspos забезпечують сильніші спростування основної гіпотези. Тому критична область для цієї статистики завжди знаходиться праворуч: [□□□; +∞]. Його межу □□□ = □ 2 (□ − □ − 1; □) можна знайти з таблиці розподілу □ 2 і заданих значень s, k (кількість інтервалів), r = 2 (параметри a і b).

□□□ = 14,06714; Kspos = 9,26

Спостережувані значення статистики Пірсона не потрапляють у критичну область: Кспос < □□□, тому немає підстав відхиляти основну гіпотезу. Справедливо вважати, що дана вибірка має однакову регулярність.

1.3. Тестування.

Варто відзначити, що генератори випадкових чисел мають свої переваги і недоліки, тому кожен користувач може вибрати генератор випадкових чисел відповідно до своїх побажань і завдань. Деякі користувачі ставлять собі за мету швидко генерувати випадкові числа, де стійкість послідовності до злому не є обов’язковою, інші намагаються уникнути навіть найменшої ймовірності злому, деякі прагнуть генерувати випадкові числа якомога швидше, не замислюючись. Щодо стабільності послідовності , це залежить лише від часу отримання випадкового числа. Виходячи з цього, необхідно протестувати і з'ясувати, за яких обставин необхідно використовувати розроблену бібліотеку. Також було завдання висвітлити плюси та мінуси використання генератора випадкових чисел на основі шуму звукової карти. Як будь-який апаратний генератор, він приймає будь-яку інформацію ззовні як «сировину» для майбутніх згенерованих послідовностей і не підкоряється формулам із власними особливостями. До переваг таких генераторів, як і створених ними, можна віднести високу стабільність результуючої послідовності, оскільки неможливо точно відтворити початкові умови, а пряма залежність генерованої послідовності від дій користувача дозволяє не думати про як працює генератор, від чого це залежить, адже користувач сам є джерелом ентропії. Основна проблема апаратних генераторів випадкових чисел полягає в тому, що вони відносно повільні порівняно з генераторами псевдовипадкових чисел. Тому бібліотеки генерації випадкової послідовності, засновані на генерації шуму звукової карти, працюють повільніше, оскільки в процесі своєї роботи їм доводиться записувати звуки, на основі яких створюються послідовності. Іншою проблемою, пов’язаною з апаратними генераторами випадкових чисел, є зсув математичного сподівання послідовності вихідних бітів (коли Деякі числа в послідовності більше, ніж інші, наприклад, у двійковій системі більше одиниць, ніж нулів).

Це викликано особливостями фізичного процесу, який використовується в генераторі шуму. Ця проблема вирішується за допомогою спеціальних алгоритмів, які дозволяють зрівняти кількість нулів і одиниць у досить великій вибірці чисел. Тести показують, що після стандартної нормалізації вхідної послідовності вихід містить велику кількість ідентичних елементів, тобто мінімальне повторення елемента, тобто значення, яке відповідає початку діапазону вихідної послідовності.

При формуванні послідовності з 10000 чисел в діапазоні від 1 до 100 отримати результат, який містить велику кількість найменших елементів (відповідних числу 1) , що проілюстровано на Рис.3.3.

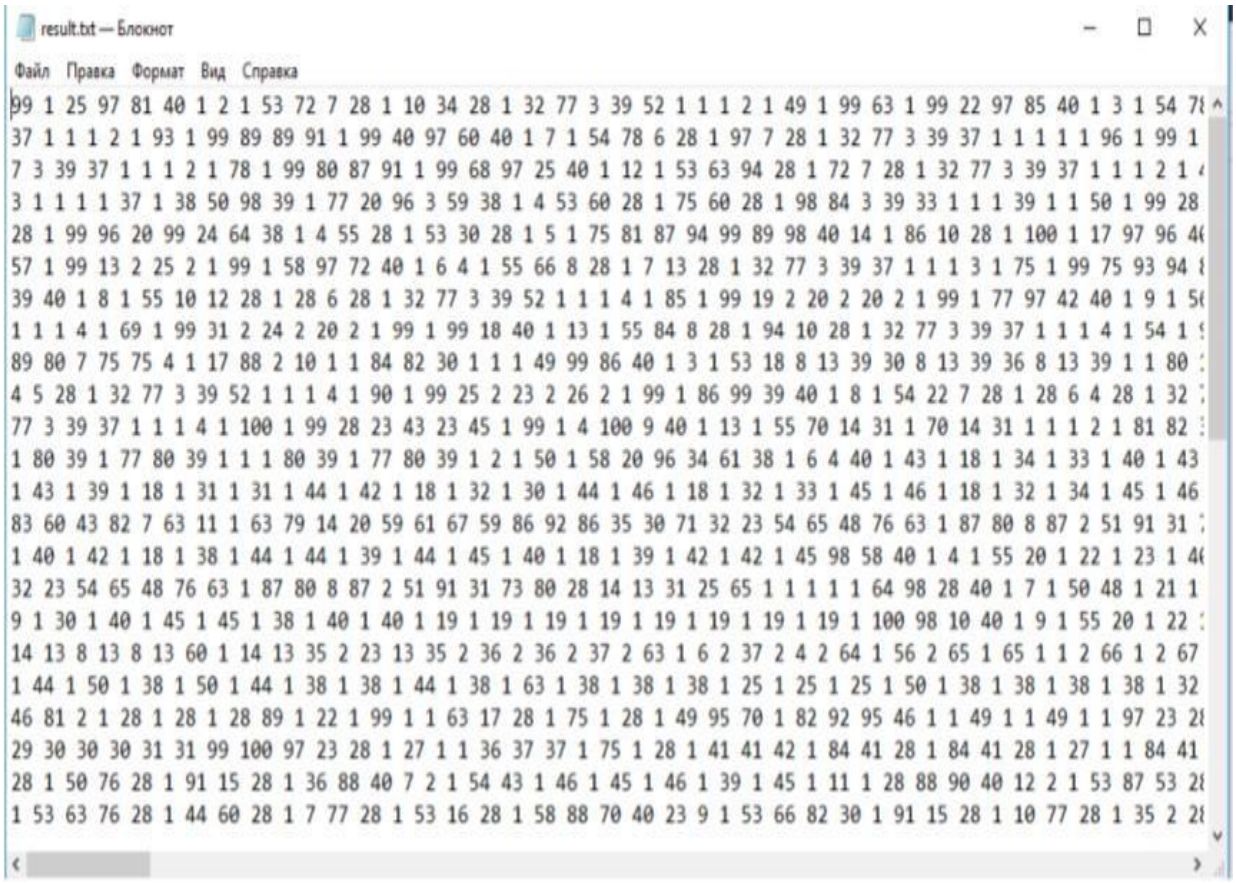

Рис.3.5 Результат роботи бібліотеки при діапазоні (1; 100)

Щоб переконатися, що найменших елементів справді забагато, для підтвердження цього припущення був побудований графік Графік проілюстровано на Рис.3.6.

Рис.3.6 Графік розподілу елементів

Як видно з графіка, найменший елемент повторюється звичайну кількість разів, однак, згідно із записаним звуком, кількість повторюваних елементів стрибає на значення 14. Тим не менш, розраховані значення не відрізняються більше допустимого діапазону. Найбільша кількість повторень для значення 14 становить 662, друге за значенням значення 44 має кількість повторень 417, тоді як значення 1 має максимальну кількість повторень, що дорівнює 3171 перед застосуванням алгоритму, наступне значення 28 має число повторень 593, максимальна кількість повторень Різниця до наступного повторення становить 2624 (245 при застосуванні алгоритму).

ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі описано основні структури даних і функції, що використовуються в програмі, яка записує звук, бібліотеку, яка споживає записаний звук і нормалізує його до потрібного діапазону, і програму, яка використовує функції, описані в бібліотеці. Бібліотека викликає програму для запису звуку та обробки отриманої інформації. Функція генерації випадкових чисел із параметрами довжини масиву та інтервалу також зіграла найважливішу роль і завершила всі завдання. Результати роботи над цією бібліотекою показують, що якщо користувач має на меті генерувати випадкові послідовності символів (наприклад, англійський алфавіт), то бібліотека може генерувати не тільки числа, але й символи. У результаті досліджень властивостей розподілу було встановлено, що використана послідовність відповідає рівномірному розподілу послідовностей. Тести показують плюси та мінуси використання бібліотеки випадкових чисел із шумом звукової карти як джерелом ентропії. Основна перевага полягає в тому, що згенерована послідовність є дійсно випадковою, оскільки звук, який використовується для генерації, записується щоразу, коли викликається функція генератора випадкових чисел. Недоліки включають проблему з усіма дійсно випадковими генераторами випадкових чисел - усі вони покладаються на певну дію ззовні. Один із головних недоліків, випадок спотворення математичних очікувань у послідовностях, був виявлений і подоланий завдяки виключенню «перенасичених елементів» у послідовності.

ВИСНОВКИ

Необхідність генерувати випадкові числа виникає в багатьох прикладних задачах. Завдання, які пред'являють найвищі вимоги до якості випадкових чисел, пов'язані з комп'ютерним моделюванням і криптографією. У цих випадках недоцільно використовувати стандартний детермінований генератор псевдовипадкових чисел, тому необхідно створити генератор випадкових чисел з джерелом ентропії. У ході кваліфікаційної роботи була розроблена бібліотека генерації випадкових чисел на основі шуму звукової карти. Бібліотека написана з використанням стандартних інструментів мови програмування C++, а додаток для запису використовує функції Windows API, які надають усі функції, необхідні для обробки звуку та забезпечують достатню ефективність і швидкість. Алгоритм генератора передбачає запис звуків у вторинній програмі та передачу їх до бібліотеки, яка нормалізує записані елементи до діапазонів, введених користувачем під час виклику функції.

Під час тестування бібліотеки були виявлені сильні та слабкі сторони обраного алгоритму та ентропії. Головним недоліком цієї бібліотеки є її час виконання, який довший, ніж у бібліотек, які генерують псевдовипадкові послідовності. Однак цей недолік компенсується тим, що вихідна послідовність має справді випадковий характер, залежно від записаного звуку. Згенеровану бібліотеку генерації випадкових чисел можна використовувати для проєктів, які потребують високоякісних послідовностей випадкових чисел. У майбутньому генератор випадкових чисел можна буде вдосконалити, використовуючи більш чіткий звук (для кращої випадковості чисел), надаючи можливість вибору розподілу, за яким користувач генерує випадкові числа.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Апаратний генератор випадкових чисел – Режим доступу: https://uk.wikipedia.org/wiki/
2. Вихор Мерсенна [Електронний ресурс] – Режим доступу: http://znaimo.com.ua/Вихор_Мерсенна
3. Генератор псевдовипадкових чисел – Режим доступу: http://znaimo.com.ua/Генератор_псевдовипадкових_чисел#link13
4. Генератор псевдовипадкових чисел – Режим доступу: https://uk.wikipedia.org/wiki/Генератор_псевдовипадкових_чисел
5. Методи статистичних випробувань – Режим доступу: http://bibl.com.ua/informatika/5566/index.html?page=2
6. Mother Of All – Режим доступу: https://www.codecogs.com/library/statistics/random/motherofall.php
7. Randomus Anomalus, Генератор випадкових чисел з апаратним джереломентропії – Режим доступу: http://konkurs.khnu.km.ua/wp-content/uploads/sites/25/2018/04/ RandomusAnomalus.pdf
8. Windows API – Режим доступу: https://ru.wikipedia.org/wiki/Windows_API
9. Xorshift – Режим доступу: https://en.wikipedia.org/wiki/Xorshift