동국대학교
dongguk university

제출일	2023.03.28	학과	컴퓨터공학전공
과목	컴퓨터보안	학번	2018112007
담당교수	김영부 교수님	이름	이승현

1) 실습 환경

(1)
운영 체제: Microsoft Windows 11 Home 64bit
프로세서 : Intel(R) Core(TM) i7-10510U @ 1.80GHz (8 CPUs), ~ 2.3GHz
메모리 : DDR4 16GB 2,667MHz
그래픽 카드 : Intel UHD Graphics

(2)
운영 체제: Microsoft Windows 10 Home 64bit
프로세서 : Intel(R) Core(TM) i7-7700HQ @ 2.80GHz (8 CPUs), ~ 2.8GHz
메모리 : DDR4 8GB 2,133MHz
그래픽 카드 : Intel HD Graphics 630, NVIDIA GeForce GTX 1050

2) 실습 진행

1. 문제 분석

- Brute Force Attack(무작위 대입 공격)이란 암호문의 암호 키를 찾기 위해 모든 경우의 수를 무작위로 대입하여 암호를 푸는 공격방법이다. 암호를 푸는 공격방법 중에는 간단하고 구현이 쉬운 방법이지만 모든 경우의 수를 대입한다는 점에서 암호문이 길고, 숫자와 영문자, 특수문자가 섞여 있는 경우에는 경우의 수가 기하급수적으로 커지기 때문에 시간적인 측면에서 적합하지 않지만 확실하게 암호를 알 수 있기에 한 번은 시도할 만한 공격방법이다. 따라서 이번 시간에는 무작위로 패스워드를 생성하고, Brute Force Attack을 각각의 패스워드에 대해 진행하여 패스워드의 유형과 길이에 따라 시간 복잡도가 어떻게 달라지고, 소요된 시간은 어떤지 살펴보는 과정을 가질 것이다.

2. 프로그램 설계 / 알고리즘

- 무작위 패스워드 생성

패스워드 유형을 (숫자), (영문자), (특수문자), (숫자 + 영문자), (숫자 + 특수문자), (영문자 + 특수문자), (숫자 + 영문자 + 특수문자)와 같이 나누고, 각 유형에 대하여 4~8문자를 가진 패스워드를 10개씩 생성한다.
패스워드 유형의 경우 스트링 벡터를 이용해 각 유형에 맞는 문자 셋을 저장하고, 필요에 맞게 문자 셋을 선택한다. 문자 셋을 선택하였다면 문자 셋 내의 문자를 이용해서 패스워드 생성을 시작하는데, random 헤더를 include하고 uniform_int_distribution 클래스로 인덱스를 무작위로 정해 문자 셋의 해당 인덱스에서 문자를 가져오게 된다. 그 후 for 문을 이용해서 가져온 문자를 차례대로 저장해 패스워드를 완성하고, 완성된 패스워드를 스트링 벡터에 저장하고, shuffle 함수를 이용해 무작위로 섞는다.

- Brute Force Attack

패스워드 유형 별로 공격을 진행한다. 처음에 숫자부터 시작하여 (숫자), (특수문자), (영문자), (숫자 + 특수문자), (숫자 + 영문자), (영문자 + 특수문자), (숫자 + 영문자 + 특수문자) 순으로 경우의 수가 적은 것부터 공격을 진행한다. 이는 처음부터 (숫자 + 영문자 + 특수문자)을 가지고 공격을 진행하는 경우 비교적 간단한 패스워드에도 오랜 시간이 소요될 수 있으므로 유형 별로 공격을 진행하는 것이다.
그리고 모든 경우의 수를 찾아야 하기에 for 문 아니면 재귀함수 호출을 이용하는 방법이 가장 무난하다고 생각했다. 다만 for 문을 사용하는 경우에는 위에서 요구하는 패스워드 길이가 4~8이기 때문에 8자리 패스워드의 경우 8중 for 문을 사용하게 되고, 코드가 지저분해 보일 것이다. 따라서 재귀함수 호출을 이용해 Brute Force Attack을 구현할 것이다. 처음에 길이가 4인 패스워드부터 공격을 시도하며, 길이가 4인 패스워드의 경우의 수를 전부 봤다면 5글자 패스워드 크래킹을 진행하고, 차례대로 한 글자씩 늘려가며 패스워드의 경우의 수를 확인한다. 중간에 패스워드를 발견했다면 flag를 true로 변경하고, 재귀함수를 도중에 종료할 수 있도록 처리한다.
패스워드 비교의 경우 단순히 자리별로 문자를 비교하면 되지만, 저는 운영 체제에서 패스워드가 해시로 저장되어 있다는 점에 착안하여, 처음에 무작위로 생성한 패스워드를 폴딩 법으로 간단하게 해싱하고, 공격할 때 또한

각 경우의 수에 해싱을 진행한다. 그 후 서로 해시값을 비교하고 같은 경우에 문자열까지 같은지 확인하고 문자열까지 완전히 같다면 공격을 중단한다.
또한, CPU보다 GPU가 병렬연산 면에서 더 뛰어나기 때문에 NVIDIA의 CUDA를 이용해 GPU에 직접 메모리를 할당하고 값을 GPU에 저장한 다음, GPU에서 Brute Force Attack을 진행한다. 여기서는 재귀함수가 아닌 for 문의 중첩을 이용해 구현해 볼 예정이며, CUDA C/C++를 이용하여 코드를 작성한다. 위에서 말했듯이 8중 for 문을 사용해야 하는데 처음부터 8중 for 문이 전부 작동하지 않는다. 각 for 문에 조건식을 달아 4글자 패스워드의 경우의 수를 알아볼 때는 밑에 존재하는 for 문이 작동하지 않도록 조건식을 지정하고, 글자가 늘어날 때마다 밑에 있는 for 문이 작동할 수 있도록 처리한다.

- GUI

winform을 이용하여 간단하게 GUI를 구현해보았다. winform은 마이크로소프트의 닷넷 프레임워크에서 제공하며, visual studio에서 GUI를 구성할 수 있다. C++과 C#을 이용해 GUI의 기능을 설정할 수 있으며, 나는 C++을 이용해 GUI를 구성하였다. 이때 CLI를 따로 설치해야 하므로 C#보다 번거로움은 있었지만, C#에 익숙지 않다면 CLI를 통해서 C++로도 구현할 수 있기에 이를 위해서라면 어쩔 수 없다. GUI를 구성하는 요소들은 왼쪽의 도구 상자에서 끌어다가 원하는 위치에 배치할 수 있으며, 요소별로 이벤트를 설정해줄 수 있다. 이 이벤트에는 자기가 작성한 함수도 포함하기 때문에 원하는 기능을 구현하여 요소에 지정할 수 있다. 나는 버튼을 배치한 다음, 패스워드 생성과 공격 기능을 버튼별로 부여하여 버튼을 클릭할 때마다 실행될 수 있도록 하였다. 그리고 패스워드를 생성하지 않은 상태에서 공격을 진행하지 않도록 처리를 했다.

3. 소스 코드 / 주석

(1) 패스워드 생성 및 Brute Force Attack (code.h)

```
#include <iostream>
#include <string>
#include <vector>
#include <chrono>
#include <random>
#include <thread>
using namespace std;
using namespace System;
class BruteForce
{
public:
    vector<string> character_set = { "0123456789",
                                     "!\"#$%&'()*+,-./:;<=.?@[\\]^_`{|}~",
                                     "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ",
                                     "0123456789!\"#$%&'()*+,-./:;<=.?@[\\]^_`{|}~",
                                     "0123456789aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ",
                                     "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ!\"#$%&'()*+,-./:;<=.?@[\\]^_`{|}~",
                                     "0123456789aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ!\"#$%&'()*+,-./:;<=.?@[\\]^_`{|}~"
    }; //각 패스워드 유형별 문자셋 저장
    vector<string> mode_name = { "Number", "Special", "Alpha", "Number + Special", "Number + Alpha", "Alpha + Special", "Number + Alpha + Special" };
    //패스워드 유형별 모드 출력하기 위한 스트링 벡터

    int hashval =0; //target password의 해시 값을 저장하기 위한 변수
    bool check =false; //password를 알아낸 경우 함수를 도중에 중지하기 위한 플래그 변수
public:
        BruteForce()
        {
        }
        int Password_character(int size)
        {
            random_device rd;
            mt19937 gen(rd());
            uniform_int_distribution<int> digit(0, size -1);    //문자셋에서 가져올 문자의 인덱스를 균일한 확률로 무작위로 선택
            int character = digit(gen);                          //인덱스를 저장
            return character;                                    //인덱스 반환
        }
        int mode(bool Number, bool Alpha, bool Special)
        {
            if (Number && Alpha && Special) return 6;            //패스워드 생성에서 숫자, 알파벳, 특수문자 모두 선택했을 경우
            else if (Alpha && Special) return 5;                 //패스워드 생성에서 알파벳, 특수문자 선택했을 경우
            else if (Number && Alpha) return 4;                  //패스워드 생성에서 숫자, 알파벳 선택했을 경우
            else if (Number && Special) return 3;                //패스워드 생성에서 숫자, 특수문자 선택했을 경우
            else if (Alpha) return 2;                            //패스워드 생성에서 알파벳 선택했을 경우
            else if (Special) return 1;                          //패스워드 생성에서 특수문자 선택했을 경우
            else if (Number) return 0;                           //패스워드 생성에서 숫자 선택했을 경우
            else return -1;                                      //패스워드 생성에서 아무것도 선택하지 않은 경우, 모든 유형의 패스워드 전부 생성하도록 처리
        }
        vector<string> Make_password(bool Number, bool Alpha, bool Special, System::Windows::Forms::TextBox^ box)
        {
            vector<string> password_set;
            string password ="";
            if (mode(Number, Alpha, Special) !=-1)
            {
                for (int length =4; length <=8; length++) //4 ~ 8자리의 패스워드 생성
                {
                    for (int count =0; count <10; count++) //총 10개의 패스워드 생성
                    {
                        for (int index =0; index < length; index++)
                        {
                            password += character_set[mode(Number, Alpha, Special)][Password_character(character_set[mode(Number, Alpha, Special)].length())];
                            //선택한 유형의 문자 셋에서 무작위로 문자를 선택해 추가
                        }
                        password_set.push_back(password);    //벡터에 만들어진 패스워드 저장
                        password.clear();    //새로운 패스워드를 생성하기 위해 초기화
                    }
                }
                box->Text ="Generating ("+ ToGcString(mode_name[mode(Number, Alpha, Special)]) +") Passwords";
```

```
            //GUI에 선택한 유형의 패스워드 생성이 끝났음을 출력
        }
        else
        {
            for (int length =4; length <=8; length++) //4 ~ 8자리의 패스워드 생성
            {
                for (int charset =0; charset < character_set.size(); charset++) //모든 유형의 패스워드 생성
                {
                    for (int count =0; count <10; count++) //총 10개의 패스워드 생성
                    {
                        for (int index =0; index < length; index++)
                        {
                            password += character_set[charset][Password_character(character_set[charset].length())]; //유형별로 순회하면서 문자 셋에서 무작위로 문자를 선택해 추가
                        }
                        password_set.push_back(password); //벡터에 만들어진 패스워드 저장
                        password.clear(); //새로운 패스워드를 생성하기 위해 초기화
                    }
                }
            }
            random_shuffle(password_set.begin(), password_set.end()); //무작위로 패스워드 섞음
            box->Text ="Generating All Category Passwords";
            //GUI에 모든 유형의 패스워드 생성이 끝났음을 출력
        }
        return password_set; //생성한 패스워드들이 저장된 벡터 반환
    }
    void find(int count, int index, string password, string origin, int charset, int characters, string& text)
    {
        if (count ==0)
        {
            int sum =0;
            for (int i =0; i < characters; i++)
            {
                sum += (int)password[i]; //target password의 해시값 구하기
            }
            if (sum == hashval && password == origin)
            {
                DateTime dt; //시간을 출력하기 위한 클래스 선언
                text +="["+ ToStdString(dt.Now.ToString("yyyy-MM-dd hh:mm:ss")) +"] "+"Hit! : "+ password + ToStdString(Environment::NewLine);
                //크래킹이 끝난 시점의 시간 출력
                check =true; //플래그를 true로 지정하여 재귀적으로 호출한 함수들 모두 종료
            }
        }
        else
        {
            for (int j =0; j < character_set[charset].size(); j++)
            {
                if (check ==true) return; //플래그가 true라면 함수 종료
                password[index] = character_set[charset][j]; //현재 진행중인 인덱스의 자리에서 문자 값을 변경
                find(count -1, index +1, password, origin, charset, characters, text); //다음 인덱스를 처리하기 위해 함수 재귀적으로 호출
            }
        }
    }
    void Attack(vector<string> password_set, System::Windows::Forms::TextBox^ box)
    {
        int count =0; //현재 몇번째 패스워드 공격인지 명시하기 위한 벼수
        string text =""; //GUI에 메세지를 출력하기 위한 변수
        for (auto pw : password_set) //저장된 패스워드 전체 순회
        {
            DateTime dt; //시간을 출력하기 위한 클래스 선언
            auto start = chrono::high_resolution_clock::now(); //공격 시작 시간 저장
            check =false;   //플래그를 초기화
            text += to_string(++count) +"번째 Cracking 할 Password : "+ pw + ToStdString(Environment::NewLine);
            //패스워드 비교를 위해 target password 출력
            text +="["+ ToStdString(dt.Now.ToString("yyyy-MM-dd hh:mm:ss"))+"] "+ to_string(count) +"번째 패스워드 진행 시작"+ ToStdString(Environment::NewLine);
            //패스워드 공격 시작 시간 출력
            hashval =0; //해시값 초기화
            string password =""; //공격이 끝난 후 패스워드를 출력하기 위한 변수
            for (auto c : pw) hashval += (int)c; //target password의 해시값 저장
            for (int characters =4; characters <=8; characters++) //4 ~ 8자리의 패스워드 탐색
            {
                for (int charset =0; charset < character_set.size(); charset++) //모든 유형의 패스워드 탐색
                {
                    for (int i =0; i < characters; ++i) password += character_set[charset][0];
                    //패스워드 초기화
                    find(characters, 0, password, pw, charset, characters, text);
                    //패스워드 공격 시작
                    password.clear();
                    //다음 패스워드 탐색을 위한 초기화
                    if (check ==true) break;
                    //패스워드를 찾은 경우 반복문에서 탈출
                }
                if (check ==true) break; //패스워드를 찾은 경우 반복문에서 탈출
                cout <<endl;
            }
            auto stop = chrono::high_resolution_clock::now();
            auto duration = chrono::duration_cast<chrono::seconds>(stop - start); //패스워드 공격 끝난 시간 저장
            text +="Duration of time : "+ to_string(duration.count()) +" seconds"+ ToStdString(Environment::NewLine + Environment::NewLine);
            //공격 시작 시간과 끝난 시간의 차로 소요시간 연산
            box->AppendText(ToGcString(text)); //GUI에 패스워드 공격 과정 출력
        }
        box->Text +="End Attack"; //공격이 전부 끝났음을 출력
    }
     private: String^ ToGcString(string std_string) {
            return gcnew String(std_string.data()); //std::string 에서 System::String으로 변환
            }                                       //winform에서 std::string을 출력할 수 없어서 System::String으로 변환 필요
     private: string ToStdString(String^ gc_string) {
         using namespace Runtime::InteropServices;
         const char* chars = (const char*)(Marshal::StringToHGlobalAnsi(gc_string)).ToPointer();
         string std_string = chars;
         Marshal::FreeHGlobal(IntPtr((void*)chars));
         return std_string; //System::String을 std::string으로 변환
     }
};
```

(2) GUI (GUI.h, GUI.cpp)

<GUI.h>

```
#include "code.h"
#pragma once

namespace BruteForceAttack {
                using namespace System;
                using namespace System::ComponentModel;
                using namespace System::Collections;
                using namespace System::Windows::Forms;
                using namespace System::Data;
                using namespace System::Drawing;
                std::vector<std::string> character_set; //패스워드를 저장하기 위한 벡터
                bool Number =false, Alpha =false, Special =false;
                //선택한 유형이 무엇인지 판별하기 위한 플래그
                /// <summary>
                /// GUI에 대한 요약입니다.
```

```
/// </summary>
public ref class GUI : public System::Windows::Forms::Form
{
public:
	GUI(void)
	{
		InitializeComponent();
		CheckForIllegalCrossThreadCalls =false;
		textBox1->Select(textBox1->Text->Length, 0);
		textBox1->ScrollToCaret();
		textBox1->WordWrap =true;
		//
		//TODO: 생성자 코드를 여기에 추가합니다.
		//
	}
protected:
	/// <summary>
	/// 사용 중인 모든 리소스를 정리합니다.
	/// </summary>
	~GUI()
	{
		if (components)
		{
			delete components;
		}
	}
private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::Button^ button2;
private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::CheckBox^ checkBox1;
private: System::Windows::Forms::CheckBox^ checkBox2;
private: System::Windows::Forms::CheckBox^ checkBox3;
private: System::Windows::Forms::Button^ button3;
protected:
private:
	/// <summary>
	/// 필수 디자이너 변수입니다.
	/// </summary>
	System::ComponentModel::Container^ components;
#pragma region Windows Form Designer generated code
	/// <summary>
	/// 디자이너 지원에 필요한 메서드입니다.
	/// 이 메서드의 내용을 코드 편집기로 수정하지 마세요.
	/// </summary>
	void InitializeComponent(void)
	{
		this->button1 = (gcnew System::Windows::Forms::Button());
		this->button2 = (gcnew System::Windows::Forms::Button());
		this->textBox1 = (gcnew System::Windows::Forms::TextBox());
		this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());
		this->checkBox2 = (gcnew System::Windows::Forms::CheckBox());
		this->checkBox3 = (gcnew System::Windows::Forms::CheckBox());
		this->button3 = (gcnew System::Windows::Forms::Button());
		this->SuspendLayout();
		//
		// button1
		//
		this->button1->Location =System::Drawing::Point(476, 37);
		this->button1->Name = L"button1";
		this->button1->Size =System::Drawing::Size(163, 93);
		this->button1->TabIndex =0;
		this->button1->Text = L"Generate Passwords";
		this->button1->UseVisualStyleBackColor =true;
		this->button1->Click += gcnew System::EventHandler(this, &GUI::button1_Click);
		//
		// button2
		//
		this->button2->Location =System::Drawing::Point(476, 185);
		this->button2->Name = L"button2";
		this->button2->Size =System::Drawing::Size(163, 93);
		this->button2->TabIndex =1;
		this->button2->Text = L"Start Cracking";
		this->button2->UseVisualStyleBackColor =true;
		this->button2->Click += gcnew System::EventHandler(this, &GUI::button2_Click);
		//
		// textBox1
		//
		this->textBox1->Location =System::Drawing::Point(53, 185);
		this->textBox1->Multiline =true;
		this->textBox1->Name = L"textBox1";
		this->textBox1->ReadOnly =true;
		this->textBox1->ScrollBars =System::Windows::Forms::ScrollBars::Vertical;
		this->textBox1->Size =System::Drawing::Size(354, 205);
		this->textBox1->TabIndex =2;
		this->textBox1->TextChanged += gcnew System::EventHandler(this, &GUI::textBox1_TextChanged);
		//
		// checkBox1
		//
		this->checkBox1->AutoSize =true;
		this->checkBox1->Location =System::Drawing::Point(88, 76);
		this->checkBox1->Name = L"checkBox1";
		this->checkBox1->Size =System::Drawing::Size(69, 16);
		this->checkBox1->TabIndex =3;
		this->checkBox1->Text = L"Number";
		this->checkBox1->UseVisualStyleBackColor =true;
		this->checkBox1->CheckedChanged += gcnew System::EventHandler(this, &GUI::checkBox1_CheckedChanged);
		//
		// checkBox2
		//
		this->checkBox2->AutoSize =true;
		this->checkBox2->Location =System::Drawing::Point(192, 76);
		this->checkBox2->Name = L"checkBox2";
		this->checkBox2->Size =System::Drawing::Size(56, 16);
		this->checkBox2->TabIndex =4;
		this->checkBox2->Text = L"Alpha";
		this->checkBox2->UseVisualStyleBackColor =true;
		this->checkBox2->CheckedChanged += gcnew System::EventHandler(this, &GUI::checkBox2_CheckedChanged);
		//
		// checkBox3
		//
		this->checkBox3->AutoSize =true;
		this->checkBox3->Location =System::Drawing::Point(302, 76);
		this->checkBox3->Name = L"checkBox3";
		this->checkBox3->Size =System::Drawing::Size(66, 16);
		this->checkBox3->TabIndex =5;
		this->checkBox3->Text = L"Special";
		this->checkBox3->UseVisualStyleBackColor =true;
		this->checkBox3->CheckedChanged += gcnew System::EventHandler(this, &GUI::checkBox3_CheckedChanged);
		//
		// button3
		//
		this->button3->Location =System::Drawing::Point(476, 297);
		this->button3->Name = L"button3";
		this->button3->Size =System::Drawing::Size(163, 93);
		this->button3->TabIndex =6;
		this->button3->Text = L"Exit";
```

```
                this->button3->UseVisualStyleBackColor = true;
                this->button3->Click += gcnew System::EventHandler(this, &GUI::button3_Click);
                //
                // GUI
                //
                this->AutoScaleDimensions = System::Drawing::SizeF(7, 12);
                this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                this->ClientSize = System::Drawing::Size(736, 525);
                this->Controls->Add(this->button3);
                this->Controls->Add(this->checkBox3);
                this->Controls->Add(this->checkBox2);
                this->Controls->Add(this->checkBox1);
                this->Controls->Add(this->textBox1);
                this->Controls->Add(this->button2);
                this->Controls->Add(this->button1);
                this->Name = L"GUI";
                this->Text = L"Brute Force Attack";
                this->Load += gcnew System::EventHandler(this, &GUI::GUI_Load);
                this->ResumeLayout(false);
                this->PerformLayout();
            }
#pragma endregion
        private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
                BruteForce b;
                character_set.clear(); //패스워드가 저장된 벡터 초기화
                character_set = b.Make_password(Number, Alpha, Special, textBox1);
                //선택한 유형에 맞게 패스워드 생성
        }
        private: void run()
        {
                BruteForce b;
                b.Attack(character_set, textBox1);                  //패스워드 공격 시작
        }

        private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
                textBox1->Text ="";                 //GUI에 출력하기 위한 변수
                if (character_set.size() !=0)       System::Threading::Tasks::Task::Factory->StartNew(gcnew Action(this, &GUI::run));
                        //스레드를 사용하여 멈춤현상 없이 패스워드가 생성될때마다 GUI에 출력할 수 있도록 처리
                else textBox1->AppendText("You don't have passwords");
        }
        private: System::Void textBox1_TextChanged(System::Object^ sender, System::EventArgs^ e) {
        }
        private: System::Void checkBox1_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {
        Number =!(Number); //숫자 유형 플래그 값 변경
}
        private: System::Void checkBox2_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {
        Alpha =!(Alpha); //알파벳 유형 플래그 값 변경
}
        private: System::Void checkBox3_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {
        Special =!(Special); //특수문자 유형 플래그 값 변경
}
        private: void exit_program(){
                exit(1); //프로그램 종료
        }
        private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
                System::Threading::Tasks::Task::Factory->StartNew(gcnew Action(this, &GUI::exit_program));
                //스레드를 사용하여 곧바로 프로그램을 종료할 수 있도록 처리
}
        private: System::Void GUI_Load(System::Object^ sender, System::EventArgs^ e) {
}
        };
}
```

<GUI.cpp>

```
#include "GUI.h"
using namespace System;
using namespace System::Windows::Forms;
[STAThreadAttribute]
void main() {
        Application::SetCompatibleTextRenderingDefault(false);
        Application::EnableVisualStyles();
        BruteForceAttack::GUI form;     // 프로젝트내 GUI 파일 선언
        Application::Run(% form); //GUI 실행
}
```

(3) CUDA C/C++로 구현(BF_CUDA.cu)

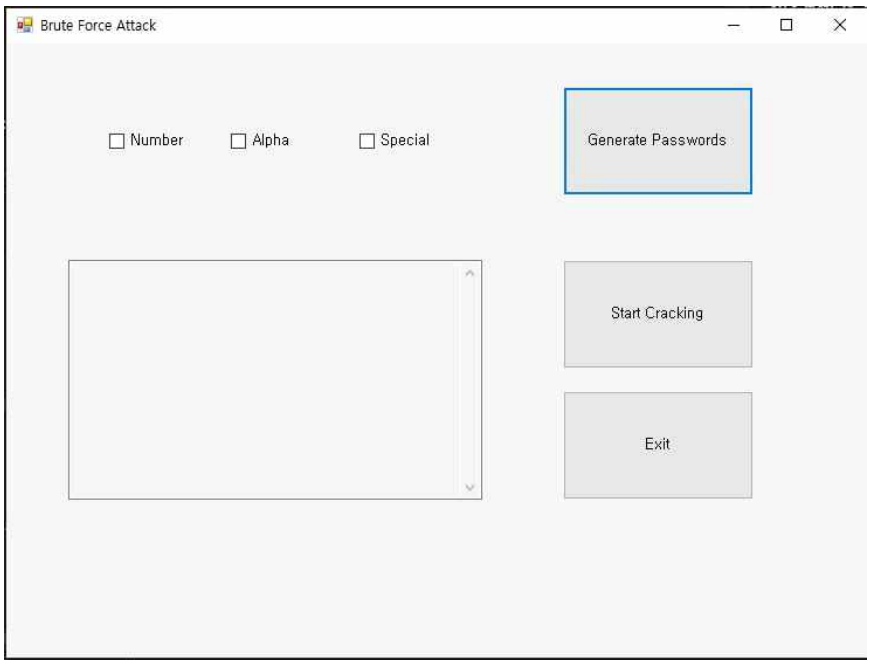
```
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <random>
#include <vector>
#include <string>
#include <chrono>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#define PW_LENGTH 8 // 암호 길이
#define NUM_CHARS 94 // 사용 가능한 문자 개수
using namespace std;
vector<string> password_set;
vector<string> character_set = { "0123456789",
                                "!\"#$%&'()*+,-./::<=.?@[\\]^_`{|}~",
                                "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ",
                                "0123456789!\"#$%&'()*+,-./::<=.?@[\\]^_`{|}~",
                                "0123456789aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ",
                                "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ!\"#$%&'()*+,-./::<=.?@[\\]^_`{|}~",
                                "0123456789aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ!\"#$%&'()*+,-./::<=.?@[\\]^_`{|}~"
};//각 패스워드 유형별 문자셋 저장
int Password_character(int size)
{
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> digit(0, size -1); //문자셋에서 가져올 문자의 인덱스를 균일한 확률로 무작위로 선택
    int character = digit(gen); //인덱스를 저장
    return character;   //인덱스 반환
}
void Make_password()
{
    string password ="";
    for (int count =0; count <10; count++) //총 10개의 패스워드 생성
    {
        for (int length =4; length <=8; length++) //4 ~ 8자리의 패스워드 생성
        {
            for (int charset =0; charset < character_set.size(); charset++) //모든 유형의 패스워드 생성
            {
                for (int index =0; index < length; index++)
                {
                    password += character_set[charset][Password_character(character_set[charset].length())];
                    //유형별로 순회하면서 문자 셋에서 무작위로 문자를 선택해 추가
                }
```

```
                    password_set.push_back(password); //벡터에 만들어진 패스워드 저장
                    password.clear(); //새로운 패스워드를 생성하기 위해 초기화
                }
            }
        }
    random_shuffle(password_set.begin(), password_set.end()); //무작위로 패스워드 섞음
}
__device__ int strncmp(const char* s1, const char* s2, size_t n) {
    unsigned char uc1, uc2;
    if (n ==0)
        return 0; //비교할 문자가 0개라면 0반환
    while (n-->0 &&*s1 ==*s2) {
        if (n ==0 ||*s1 =='\0') //앞에서 n개의 문자가 같거나 s2의 앞부분이 s1과 같다면 0반환
            return 0;
        s1++;
        s2++;
    }
    uc1 = (*(unsigned char*)s1);
    uc2 = (*(unsigned char*)s2);
    return ((uc1 < uc2) ? -1 : (uc1 > uc2)); //s1이 더 크다면 1, s2가 더 크다면 -1 반환
}
__device__ void my_strcpy(char* s1, char* s2)
{
    int i =0;
    for (i =0; s2[i] !='\0'; i++)
        s1[i] = s2[i]; //s2의 문자를 s1에 복사
    s1[i] ='\0'; //맨 끝에는 null 저장
}
__global__ void bruteForceAttackKernel(char* target, char* result, int *length) {
    // 스레드 인덱스 계산
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    // 가능한 모든 암호를 생성하여 확인
    char pw[PW_LENGTH +1];
    pw[PW_LENGTH] ='\0';
    for (int i =0; i < NUM_CHARS; ++i) {
        for (int j =0; j < NUM_CHARS; ++j) {
            for (int k =0; k < NUM_CHARS; ++k) {
                for (int l =0; l < NUM_CHARS; ++l) {
                    pw[0] = i +33;
                    pw[1] = j +33;
                    pw[2] = k +33;
                    pw[3] = l +33;
                    if (*length ==4 && strncmp(pw, target, 4) ==0) {
                        //패스워드 길이가 4이고, 앞의 문자 4개가 같은 경우
                        my_strcpy(result, pw); //결과 값 복사
                        return;                             //함수 종료
                    }
                    if (*length >4) //길이가 4보다 큰 경우
                    {
                        for (int m =0; m < NUM_CHARS; ++m)
                        {
                            pw[4] = m +33;
                            if (*length ==5 && strncmp(pw, target, 5) ==0) {
                                //패스워드 길이가 5이고, 앞의 문자 5개가 같은 경우
                                my_strcpy(result, pw); //결과 값 복사
                                return;                             //함수 종료
                            }
                            if (*length >5)     //길이가 5보다 큰 경우
                            {
                                for (int n =0; n < NUM_CHARS; ++n)
                                {
                                    pw[5] = n +33;
                                    if (*length ==6 && strncmp(pw, target, 6) ==0) {
                                        //패스워드 길이가 6이고, 앞의 문자 6개가 같은 경우
                                        my_strcpy(result, pw); //결과 값 복사
                                        return;                             //함수 종료
                                    }
                                    if (*length >6) //길이가 6보다 큰 경우
                                    {
                                        for (int o =0; o < NUM_CHARS; ++o)
                                        {
                                            pw[6] = o +33;
                                            if (*length ==7 && strncmp(pw, target, 7) ==0) {
                                                //패스워드 길이가 7이고, 앞의 문자 7개가 같은 경우
                                                my_strcpy(result, pw); //결과 값 복사
                                                return;                             //함수 종료
                                            }
                                            if (*length >7) //길이가 7보다 큰 경우
                                            {
                                                for (int p =0; p < NUM_CHARS; ++p)
                                                {
                                                    pw[7] = p +33;
                                                    if (*length ==8 && strncmp(pw, target, 8) ==0) {
                                                        //패스워드 길이가 8이고, 앞의 문자 8개가 같은 경우
                                                        my_strcpy(result, pw); //결과 값 복사
                                                        return;                             //함수 종료
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
int main() {
    Make_password(); //패스워드 생성
    int count =0;
    for (auto a : password_set) //만든 패스워드 순회
    {
        auto start = chrono::high_resolution_clock::now(); //공격 시작 시간 저장
        cout <<"Cracking Password "<<++count <<": "<< a <<endl;   //현재 크래킹할 패스워드 출력
        const char* password = a.c_str();   //패스워드를 const char*에 저장
        int strlen = a.length();  //패스워드 길이  저장
        char target[PW_LENGTH +1]; // 찾을 암호
        strcpy(target, password);   //패스워드 복사
        char result[PW_LENGTH +1] =""; // 찾은 암호
        char* d_target, * d_result; //GPU 메모리에 패스워드 데이터를 옮기기 위한 변수
        int* length;     //GPU 메모리에 패스워드 길이 값을 옮기기 위한 변수
        cudaMalloc((void**)&d_target, sizeof(char) * (PW_LENGTH +1));  //GPU 메모리에 target password를 저장할 메모리 할당
        cudaMalloc((void**)&d_result, sizeof(char) * (PW_LENGTH +1));  //GPU 메모리에 크래킹 후 결과 값을 저장할 메모리 할당
        cudaMalloc((void**)&length, sizeof(int));   //GPU 메모리에 패스워드 길이를 저장하기 위한 메모리 할당
        cudaMemcpy(d_target, target, sizeof(char) * (PW_LENGTH +1), cudaMemcpyHostToDevice);    //target password를 GPU 메모리에 복사
        cudaMemcpy(length, &strlen, sizeof(int), cudaMemcpyHostToDevice);   //패스워드 길이를 GPU 메모리에 복사
        int block_size =256;    //block 사이즈 설정
        int grid_size = (NUM_CHARS * NUM_CHARS * NUM_CHARS * NUM_CHARS + block_size -1) / block_size; //grid 사이즈 설정
        bruteForceAttackKernel <<<grid_size, block_size >>> (d_target, d_result, length);  //GPU에서 Brute Force Attack 실행
        cudaMemcpy(result, d_result, sizeof(char) * (PW_LENGTH +1), cudaMemcpyDeviceToHost); //결과 값을 다시 CPU 메모리에 복사
        cudaFree(d_target); //GPU 메모리 반환
        cudaFree(d_result); //GPU 메모리 반환
        std::cout <<"Found password: "<< result <<endl; //결과 값 출력
```

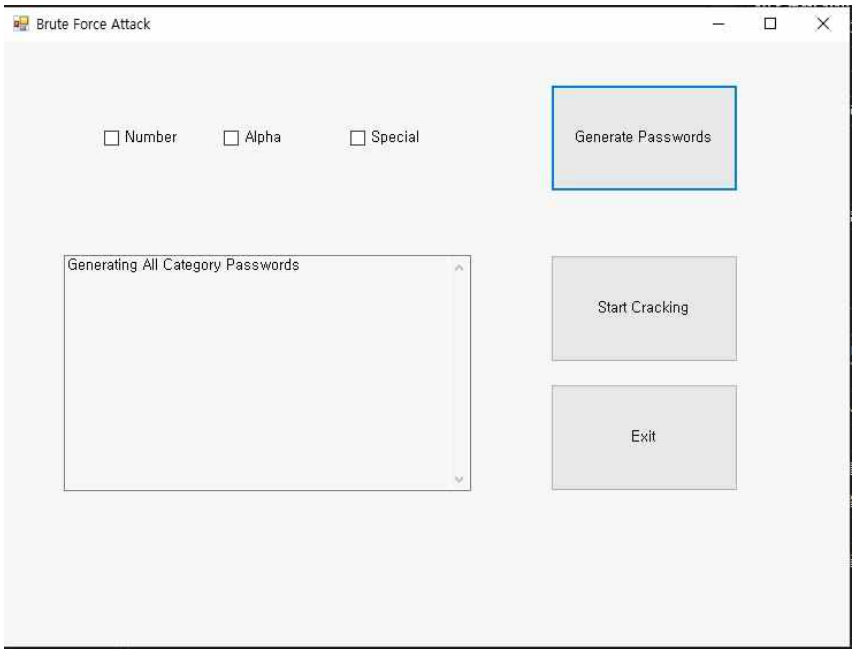
```
        auto stop = chrono::high_resolution_clock::now(); //공격 마친 시간 저장
        auto duration = chrono::duration_cast<chrono::seconds>(stop - start); //소요 시간 계산
        cout <<"Duration of time : "<< duration.count() <<" seconds"<<endl; //소요시간 출력
        cout <<"##############################################"<<endl <<endl;
    }
    return 0;
}
```

4. 결과 / 결과 분석

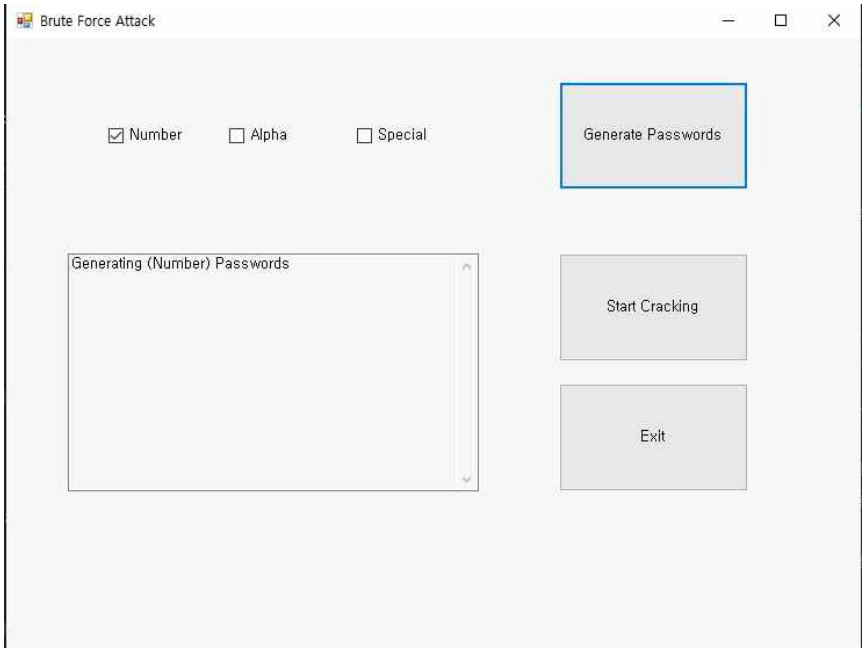
(1) Brute Force Attack && GUI (C++)



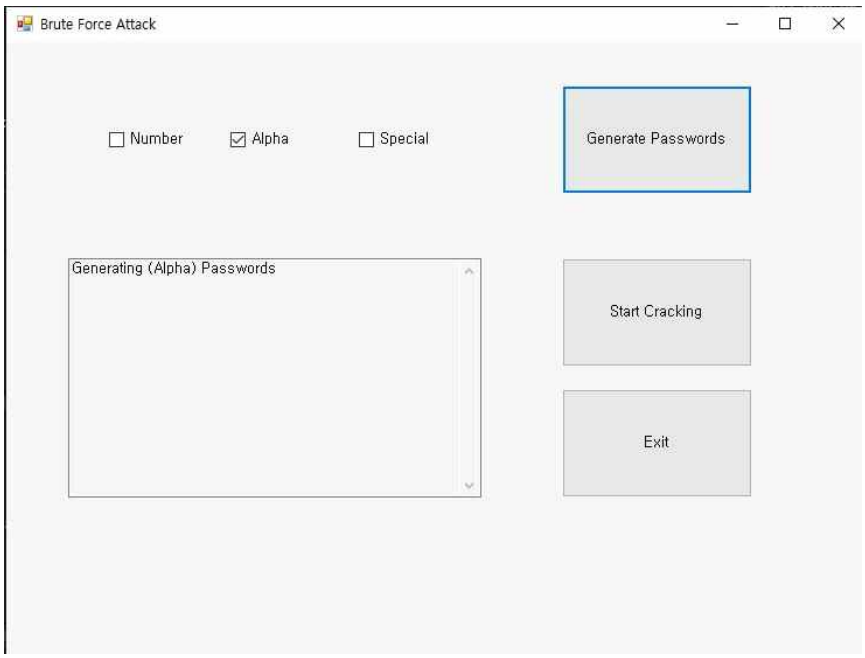
- c++과 winform으로 구현한 GUI이다. 위에서 패스워드 유형을 선택해 생성할 수 있고 Start Cracking 버튼을 클릭하여 공격을 시작할 수 있다. Exit 버튼을 클릭하면 프로그램은 종료한다.



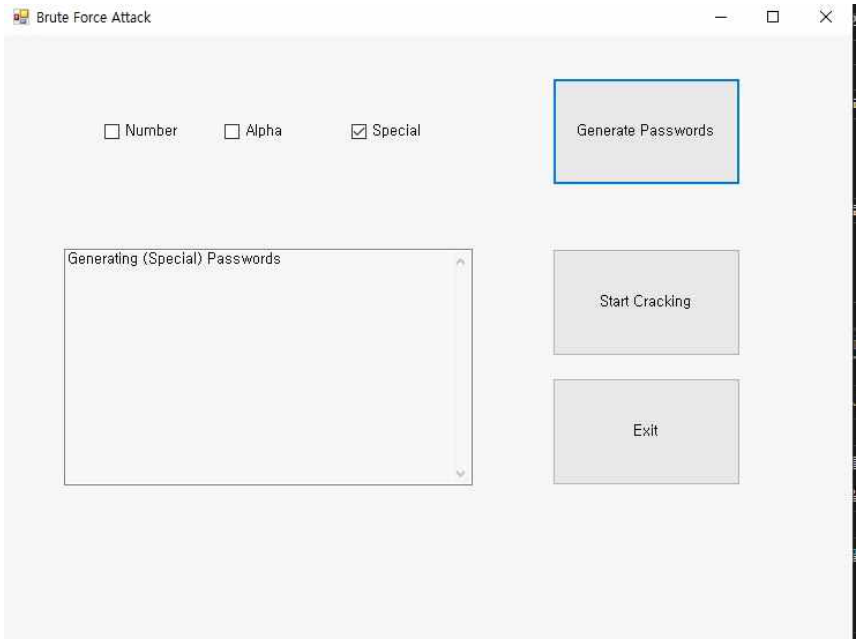
- 유형을 아무것도 선택하지 않고 패스워드 생성 버튼을 클릭하면, 모든 유형의 패스워드를 4~8문자의 길이로 10개씩 생성한다. 출력창에는 모든 유형의 패스워드를 생성했다고 메시지가 출력된다.



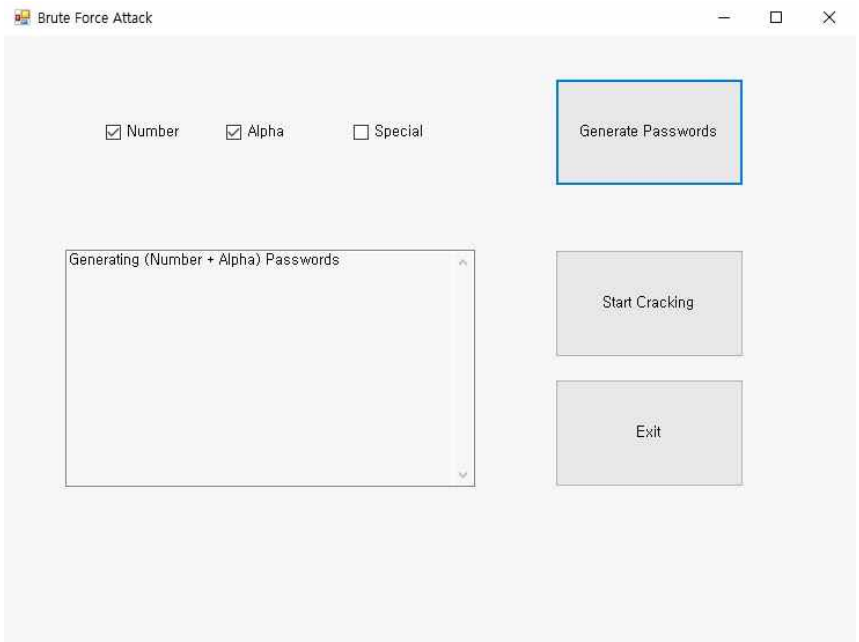
- Number를 체크 하고 패스워드 생성 버튼을 클릭하면 숫자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 숫자 유형의 패스워드를 생성했다고 메시지가 출력된다.



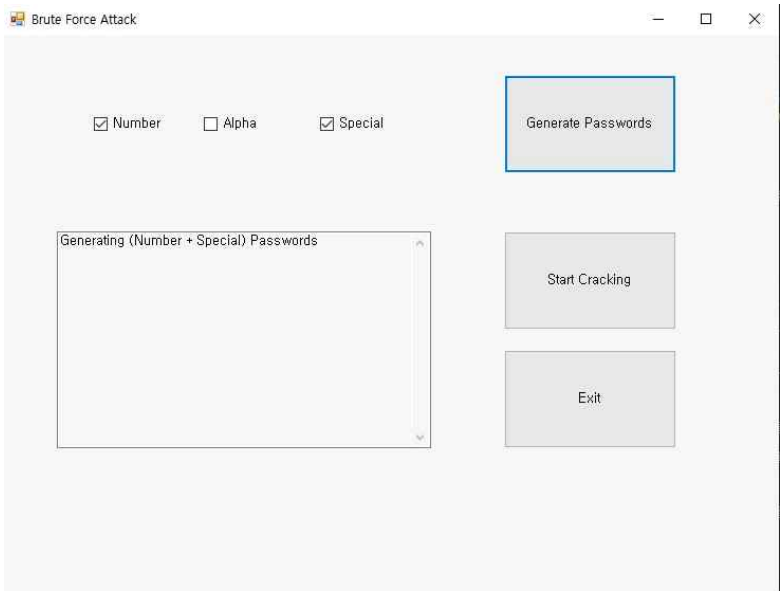
- Alpha를 체크 하고 패스워드 생성 버튼을 클릭하면 영문자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 영문자 유형의 패스워드를 생성했다고 메시지가 출력된다.



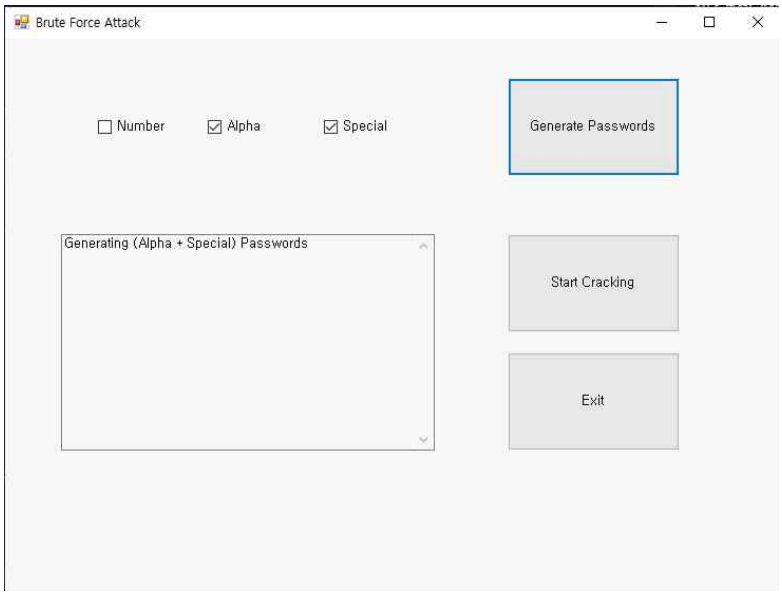
- Special을 체크 하고 패스워드 생성 버튼을 클릭하면 숫자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 숫자 유형의 패스워드를 생성했다고 메시지가 출력된다.



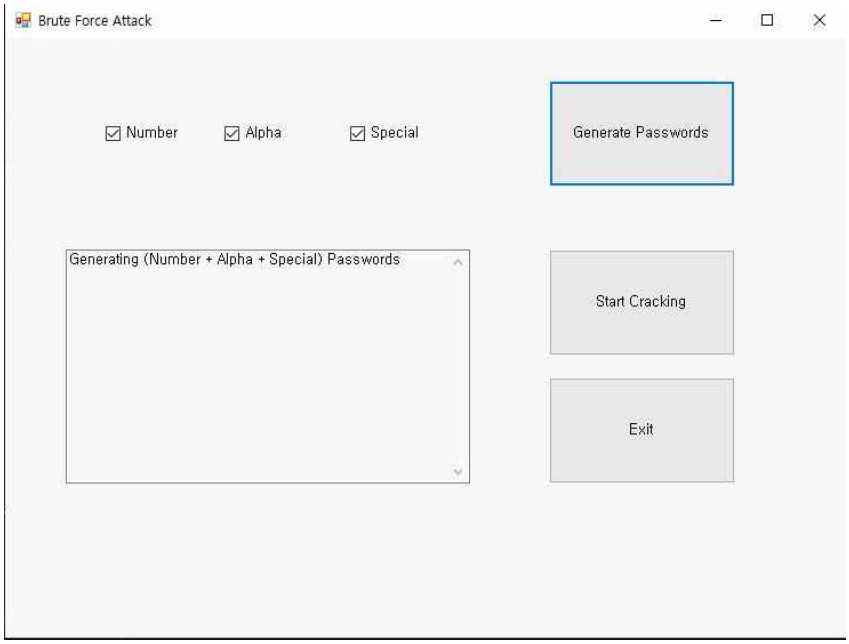
- Number, Alpha를 체크 하고 패스워드 생성 버튼을 클릭하면 숫자 + 영문자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 숫자 + 영문자 유형의 패스워드를 생성했다고 메시지가 출력된다.



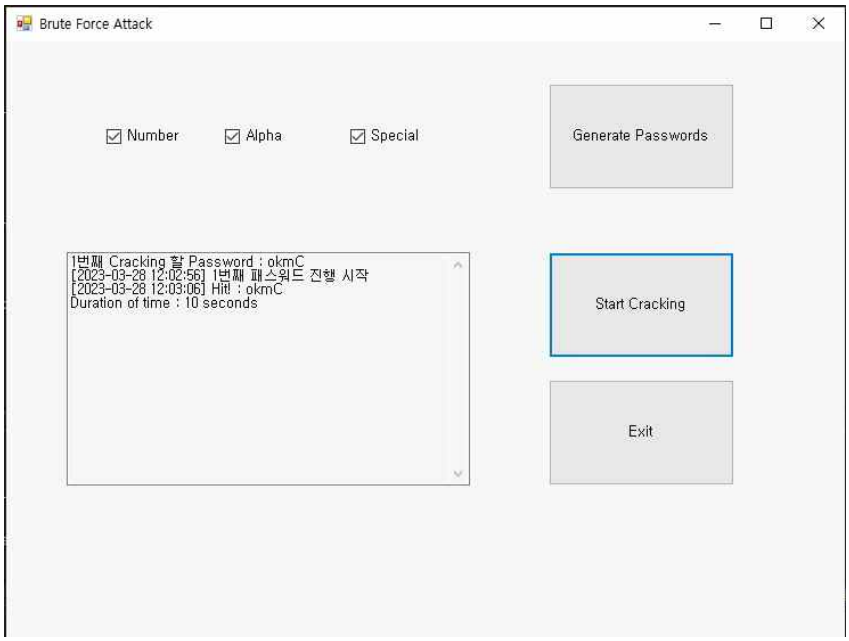
- Number, Special을 체크 하고 패스워드 생성 버튼을 클릭하면 숫자 + 특수문자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 숫자 + 특수문자 유형의 패스워드를 생성했다고 메시지가 출력된다.



- Alpha, Special을 체크 하고 패스워드 생성 버튼을 클릭하면 영문자 + 특수문자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 영문자 + 특수문자 유형의 패스워드를 생성했다고 메시지가 출력된다.



- Number, Alpha, Special을 체크 하고 패스워드 생성 버튼을 클릭하면 숫자 + 영문자 + 특수문자 유형의 패스워드를 4~8문자 길이로 10개씩 생성하며, 출력창에는 숫자 + 영문자 + 특수문자 유형의 패스워드를 생성했다고 메시지가 출력된다.



- 오른쪽의 Start Cracking 버튼을 클릭하여 Brute Force Attack을 시작한다.

(1) 4글자 패스워드

```
2번째 Cracking 할 Password : 7998
[2023-03-25 11:43:53] 2번째 패스워드 진행 시작
[2023-03-25 11:43:53] Hit! : 7998
Duration of time : 0 seconds
```

- 숫자로 이루어진 4글자 패스워드를 크래킹한 모습이다. 공격이 시도하자마자 공격이 성공한 모습을 볼 수 있었다.

```
22번째 Cracking 할 Password : fROw
[2023-03-25 11:44:21] 22번째 패스워드 진행 시작
[2023-03-25 11:44:27] Hit! : fROw
Duration of time : 6 seconds
```

- 영문자로 이루어진 4글자 패스워드를 크래킹한 모습이다. 숫자와 달리 각 자리에 올 수 있는 경우의 수가 몇 배 차이나기 때문에 6초라는 시간이 소요되었다.

```
18번째 Cracking 할 Password : .!#{
[2023-03-25 11:44:06] 18번째 패스워드 진행 시작
[2023-03-25 11:44:07] Hit! : .!#{
Duration of time : 1 seconds
```

- 특수문자로 이루어진 4글자 패스워드를 크래킹한 모습이다. 영문자에 비하면 각 자리에 수 있는 경우의 수가 적기 때문에 1초가 소요된 모습을 볼 수 있다.

```
43번째 Cracking 할 Password : 6tWp
[2023-03-25 11:48:13] 43번째 패스워드 진행 시작
[2023-03-25 11:48:31] Hit! : 6tWp
Duration of time : 18 seconds
```

- 숫자, 영어로 이루어진 4글자 패스워드를 크래킹한 모습이다. 영문자와 숫자가 조합되었기 때문에 각 자리에 올 수 있는 문자의 수는 62개로 늘어나 18초라는 시간이 소요된 것이다.

```
38번째 Cracking 할 Password : @7%9
[2023-03-25 11:46:50] 38번째 패스워드 진행 시작
[2023-03-25 11:47:10] Hit! : @7%9
Duration of time : 20 seconds
```

- 숫자, 특수문자로 이루어진 4글자 패스워드를 크래킹한 모습이다. 특수문자와 숫자가 조합되었기 때문에 각 자리에 올 수 있는 문자의 수는 42개로 늘어나 20초라는 시간이 소요되었다.

```
51번째 Cracking 할 Password : ?K₩N
[2023-03-25 11:50:21] 51번째 패스워드 진행 시작
[2023-03-25 11:52:04] Hit! : ?K₩N
Duration of time : 103 seconds
```

- 영문자, 특수문자로 이루어진 4글자 패스워드를 크래킹한 모습이다. 각 자리에 84개의 문자가 올 수 있으므로 앞의 경우와 달리 1분 넘어서 공격이 끝난 모습을 볼 수 있다.

```
68번째 Cracking 할 Password : ?L3S
[2023-03-26 12:12:52] 68번째 패스워드 진행 시작
[2023-03-26 12:16:54] Hit! : ?L3S
Duration of time : 241 seconds
```

- 숫자, 영문자, 특수문자로 이루어진 4글자 패스워드를 크래킹한 모습이다. 각 자리에 94개라는 문자가 올 수 있기에 경우의 수가 급격히 증가해 4분이 소요된 모습을 볼 수 있다.

(2) 5글자 패스워드

```
72번째 Cracking 할 Password : 87745
[2023-03-26 12:22:59] 72번째 패스워드 진행 시작
[2023-03-26 12:26:51] Hit! : 87745
Duration of time : 231 seconds
```

- 숫자로 이루어진 5글자 패스워드를 크래킹한 모습이다. 5글자이지만 숫자만으로 이루어져 있기에 비교적 짧은 시간에 공격이 끝난 모습을 볼 수 있다.

```
82번째 Cracking 할 Password : (/-^^
[2023-03-26 01:02:57] 82번째 패스워드 진행 시작
[2023-03-26 01:07:11] Hit! : (/-^^
Duration of time : 254 seconds
```

- 특수문자로 이루어진 5글자 패스워드를 크래킹한 모습이다. 숫자보다 조금 오래 걸렸지만, 마찬가지로 각 자리에 올 수 있는 경우의 수가 비교적 적기 때문에 그렇게 많은 시간이 소요되지 않은 모습을 볼 수 있다.

```
91번째 Cracking 할 Password : yPENX
[2023-03-26 01:46:35] 91번째 패스워드 진행 시작
[2023-03-26 02:00:30] Hit! : yPENX
Duration of time : 834 seconds
```

- 영어로 이루어진 5글자 패스워드를 크래킹한 모습이다. 각 자리에 올 수 있는 문자의 수가 52가지에 자릿수도 5자리이기 때문에 공격에 걸리는 시간이 앞 사례와 비교하면 많이 증가한 모습을 볼 수 있다.

```
113번째 Cracking 할 Password : B3TD1
[2023-03-26 04:57:32] 113번째 패스워드 진행 시작
[2023-03-26 05:13:49] Hit! : B3TD1
Duration of time : 977 seconds
```

- 숫자, 영어로 이루어진 5글자 패스워드를 크래킹한 모습이다. 각 자리에 62개의 문자가 올 수 있고, 자릿수 또한 5자리로 늘어났기 때문에 약 20분 가까이 공격이 실행된 모습을 볼 수 있다.

```
101번째 Cracking 할 Password : <~~49
[2023-03-26 02:55:00] 101번째 패스워드 진행 시작
[2023-03-26 03:07:05] Hit! : <~~49
Duration of time : 725 seconds
```

- 숫자, 특수문자로 이루어진 5글자 패스워드를 크래킹한 모습이다. 각 자리에 42개의 문자가 올 수 있고, 자릿수가 5자리로 늘어났기에 시간이 급격하게 증가하였지만, 영문자가 오는 경우보다는 빠른 모습을 볼 수 있다.

```
121번째 Cracking 할 Password : q(CMb
[2023-03-26 06:36:12] 121번째 패스워드 진행 시작
[2023-03-26 07:36:21] Hit! : q(CMb
Duration of time : 3608 seconds
```

- 영문자, 특수문자로 이루어진 5글자 패스워드를 크래킹한 모습이다. 영문자와 특수문자가 조합되었기 때문에 각 자리에 84개의 문자가 올 수 있고, 자릿수가 늘어남에 따라 앞의 경우와 달리 1시간이 소요된 모습을 볼 수 있다.

```
137번째 Cracking 할 Password : *5f[3
[2023-03-27 02:59:52] 137번째 패스워드 진행 시작
[2023-03-27 06:29:43] Hit! : *5f[3
Duration of time : 12591 seconds
```

-숫자, 영문자, 특수문자로 이루어진 5글자 패스워드를 크래킹한 모습이다. 각 자리에 94개의 문자가 올 수 있고 자릿수가 5자리이기 때문에 경우의 수가 94의 5승이고, 결국 공격을 끝마치는데 약 3시간 30분이 소요된 모습을 볼 수 있다.

(3) 6자리 패스워드

```
142번째 Cracking 할 Password : 934955
[2023-03-27 04:12:13] 142번째 패스워드 진행 시작
[2023-03-27 08:15:06] Hit! : 934955
Duration of time : 14572 seconds
```

- 숫자로 이루어진 6글자 패스워드를 크래킹한 모습이다. 각 자리에 올 수 있는 문자의 수는 적지만 자릿수가 6자리로 늘어났기 때문에 숫자만 이루어져 있어도 한번 크래킹하는 데 4시간이 소요되는 모습을 볼 수 있다.

```
1번째 Cracking 할 Password : 3442
[2023-03-25 11:43:53] 1번째 패스워드 진행 시작
[2023-03-25 11:43:53] Hit! : 3442
Duration of time : 0 seconds
```

```
143번째 Cracking 할 Password : 972287
[2023-03-27 08:15:06] 143번째 패스워드 진행 시작
[2023-03-28 12:45:30] Hit! : 972287
Duration of time : 16223 seconds
```

- 3월 25일 오후 11시 43분부터 3월 28일 자정까지 약 2일간 143개의 패스워드를 크래킹한 모습을 볼 수 있다.

(2) Brute Force Attack (CUDA C/C++)

- CUDA C/C++로 구현한 코드를 실행한 모습이다. 현재 콘솔 창에서는 나타나지 않지만 병렬 처리를 진행하는 모습이며 각 자리에서 병렬연산을 수행하여 빠르게 패스워드를 찾는 것을 기대했었으나, 여러 스레드가 같은 작업을 동시에 수행하기 때문에 오히려 비효율적인 모습만 보여주고 있습니다. 그래서 비교적 간단한 패스워드를 뚫는 것도 많은 시간이 소요되었으며, 이대로 더 돌리는 것은 의미 없다고 판단하여 중단했습니다. 각 자리를 나누어 연산을 시도했었어야 했는데 for 문으로 묶여 있어서 병렬연산이 기대와 달리 수행되었다고 생각합니다.

<결과 분석>

- 패스워드가 4글자인 경우에는 패스워드 당 짧으면 1초에서 길면 몇 분 정도 소요되지 않았지만, 5글자로 늘어났을 때 패스워드별로 짧아도 몇 분, 길면 몇 시간이 소요되는 모습을 볼 수 있었다. 6글자로 된 패스워드의 경우에는 며칠이 소요될 것이라는 생각이 드는데, 예전에 John the Ripper로 b4n4n4를 크래킹 시도했을 때 며칠이 지나도 크래킹이 되지 않았기 때문에다. 7~8문자의 패스워드는 집에서 흔히 쓰는 시스템으로는 며칠 걸릴지 예상이 안 가며 최악의 경우 몇 달을 바라볼 수도 있을 것이다.
- 또한, 패스워드의 유형별로도 걸리는 시간이 차이가 나는데 숫자만으로 이루어졌을 때 각 자리당 10개의 경우의 수가 존재하지만, 영문자는 52개, 특수문자는 32개가 존재하여 경우의 수가 많이 증가한다. 또한, 이들을 조합해서 사용하는 경우에는 경우의 수가 더욱 증가하며 다 조합할 경우 자리당 94개의 경우의 수가 나올 수 있다.
- Brute Force Attack 함수에 스레드를 적용하였더니 스레드가 완전히 실행이 끝나야 화면에 출력되기 때문에 공격을 마친 경우에 공격 시작 시각과 끝나는 시간이 동시에 출력된다.
- CUDA C/C++로 코딩을 한 경우를 살펴보면 병렬 처리를 함에도 패스워드 하나를 찾는 데도 오래 걸리는 모습을 볼 수 있는데 왜 그런지 변수를 찍어서 살펴보니 패스워드 각각의 자리에 대해 병렬처리하는 것을 기대했는데 실제로는 같은 작업을 여러 스레드가 동시에 처리하기 때문에 오히려 비효율적인 모습을 보여줬다.
- 이를 이용하여 패스워드 길이에 따라 경우의 수와 시간 복잡도를 구해보면 다음과 같다.
  1) 4글자
     a) 숫자 : 10 ^ 4 = 10,000
     b) 영문자 : 52 ^ 4 = 7,311,616
     c) 특수문자 : 32 ^ 4 = 1,048,576
     d) 숫자 + 영문자 : 62 ^ 4 = 14,776,336
     e) 숫자 + 특수문자 : 42 ^ 4 = 3,111,696
     f) 영문자 + 특수문자 : 84 ^ 4 = 49,787,136
     g) 숫자 + 영문자 + 특수문자 : 94 ^ 4 = 78,074,896

2) 5글자
    a) 숫자 : 10 ^ 5 = 100,000
    b) 영문자 : 52 ^ 5 = 380,204,032
    c) 특수문자 : 32 ^ 5 = 33,554,432
    d) 숫자 + 영문자 : 62 ^ 5 = 916,132,832
    e) 숫자 + 특수문자 : 42 ^ 5 = 130,691,232
    f) 영문자 + 특수문자 : 84 ^ 5 = 4,182,119,424
    g) 숫자 + 영문자 + 특수문자 : 94 ^ 5 = 7,339,040,224

3) 6글자
    a) 숫자 : 10 ^ 6 = 1,000,000
    b) 영문자 : 52 ^ 6 = 19,770,609,664
    c) 특수문자 : 32 ^ 6 = 1,073,741,824
    d) 숫자 + 영문자 : 62 ^ 6 = 56,800,235,584
    e) 숫자 + 특수문자 : 42 ^ 6 = 5,489,031,744
    f) 영문자 + 특수문자 : 84 ^ 6 = 351,298,031,616
    g) 숫자 + 영문자 + 특수문자 : 94 ^ 4 = 689,869,781,056

4) 7글자
    a) 숫자 : 10 ^ 7 = 10,000,000
    b) 영문자 : 52 ^ 7 = 1,028,071,702,528
    c) 특수문자 : 32 ^ 7 = 34,359,738,368
    d) 숫자 + 영문자 : 62 ^ 7 = 3,521,614,606,208
    e) 숫자 + 특수문자 : 42 ^ 7 = 230,539,333,248
    f) 영문자 + 특수문자 : 84 ^ 7 = 29,509,034,655,744
    g) 숫자 + 영문자 + 특수문자 : 94 ^ 7 = 64,847,759,419,264

5) 8글자
    a) 숫자 : 10 ^ 8 = 100,000,000
    b) 영문자 : 52 ^ 8 = 53,459,728,531,456
    c) 특수문자 : 32 ^ 8 = 1,099,511,627,776
    d) 숫자 + 영문자 : 62 ^ 8 = 218,340,105,584,896
    e) 숫자 + 특수문자 : 42 ^ 8 = 9,682,651,996,416
    f) 영문자 + 특수문자 : 84 ^ 8 = 2,478,758,911,082,496
    g) 숫자 + 영문자 + 특수문자 : 94 ^ 8 = 6,095,689,385,410,816

숫자만 이루어졌을 때 8자리 패스워드라도 어느 정도 크래킹이 오래 소요되지 않지만, 여기에 영문자와 특수문자가 섞일 때는 경우의 수가 엄청나게 증가한다. 특히나 숫자 + 영문자 + 특수문자의 조합 같은 경우 약 6000조라는 경우의 수가 도출되기 때문에 일반적인 가정용 컴퓨터 환경에서는 이를 뚫기가 힘들 것이다. 이 조합이 아니더라도 패스워드 길이가 길고 서로 다른 문자 셋이 조합되어 있다면 Brute Force Attack으로는 뚫기 전에 공격자가 포기할 것이다. 이 경우의 수를 이용해 시간 복잡도를 계산하게 되면 문자 셋에 있는 m개의 문자로 길이 n의 패스워드를 생성할 경우 O(m ^ n)의 시간 복잡도를 가지게 될 것이다. n이 증가한다면 시간 복잡도가 커질 것이고, m도 커질 때 더욱 시간 복잡도가 증가할 것이다.

3) 느낀 점

지난 실습에 이어서 이번에도 Brute Force Attack을 해보게 되었는데 저번과 같이 이미 구현된 프로그램을 이용하는 것이 아니라, 직접 제 손으로 구현해서 크래킹을 시도하게 되었습니다. 처음에는 제 손으로 직접 코딩해서 프로그램을 돌려보는 것이 John the Ripper를 돌리는 것보다 비효율적이라는 생각을 했었지만, 어차피 패스워드의 길이가 길어지면 모든 경우의 수를 보는 시간이 서로 오래 걸리기 때문에 거기서 거기라는 생각이 들었습니다. 아무래도 숫자, 영문자, 특수문자를 섞어서 길이가 긴 패스워드를 공격하려고 하면 6000조라는 경우의 수를 봐야 하므로 실습 시간 내에는 전부 찾아내기는 어렵다고 생각했습니다. 그래서 공격하는 시간을 줄이기 위해 CPU 대신에 GPU에서 연산을 진행하려고 CUDA를 이용하는 방법을 찾아서 해봤는데 확실히 CUDA 가 병렬 처리는 CPU보다 뛰어난 것을 알게 되었습니다. 왜냐하면, CUDA C/C++로 구현하는 과정에서 중간에 공격하는 과정을 출력했었는데 한가지 작업을 여러 스레드가 동시에 처리하여 현재 경우의 수를 여러 번 출력되기 때문입니다. 이 때문에 나중에 스레드가 작동하는 원리를 알고 잘 배분해서 병렬 처리가 잘 이루어질 수 있도록 해봐야겠습니다. 그리고 CUDA C/C++이 C/C++과 문법이 매우 다를 줄 알았는데 생각보다 문법의 차이가 덜했습니다. 그러나 GPU에서 연산하도록 GPU에 메모리 할당을 하고, 복사해야 하며, GPU에서 연산을 수행하도록 코딩할 때 저희가 평소에 많이 사용하는 라이브러리들의 함수를 대다수 지원하지 않기 때문에 직접 구현해서 사용해야 하는 번거로움이 있었습니다. 하지만 이 덕분에 C에 대해 복습할 수 있었고, 저수준 프로그래밍을 어느 정도 해보는 기회를 가질 수 있었습니다. 그리고 평소에 GUI를 만들 기회가 없었었고, 이전에 JAVA 배울 때 swing으로 간단하게 구현한 게 전부였기 때문에 이번 기회에 C++를 이용해서 GUI 구성하는 기회를 얻고 배워볼 수 있었습니다. 이번 실습 때 새로운 시도를 많이 해볼 수 있어서 좋았고, 나중에 GPU 병렬연산을 제대로 구현해보고 싶습니다.