**2-Week Roadmap — Build HashCrack++ (Custom Hashcat in C++)**

| **Day** | **Focus** | **Key Outcomes** |
| --- | --- | --- |
| **Day 1** | Project Setup | Create project structure, compile with MinGW & CMake |
| **Day 2** | Hashing Functions | Implement MD5, SHA1, NTLM hashing in C++ |
| **Day 3** | Wordlist Attack | Load wordlist, compare hashes, crack simple passwords |
| **Day 4** | Brute-Force Engine (Basic) | Generate all combinations of characters recursively |
| **Day 5** | Code Modularization | Split into CLI, hash, attack modules (clean design) |
| **Day 6** | Command-Line Interface | Add CLI flags like --mode, --attack, --hash |
| **Day 7** | Multithreading (Intro) | Split brute-force tasks across CPU threads |

| Day | Focus | Key Outcomes |
| --- | --- | --- |
| Day 8 | Advanced Brute-Force | Add min/max length, charset options, optimized gen |
| Day 9 | NTLM Hashing | Learn and implement NTLM using UTF-16 + MD4 logic |
| Day10 | Advanced Threading | Task distribution, queue/pool for brute-force threads |
| Day11 | Crack Results & Output | Show found passwords, time taken, cracked % |
| Day12 | Error Handling + UX | Show clean error messages, input validation |
| Day13 | Cross-Platform Cleanup | Prepare Linux compatibility, wrap Windows-specifics |
| Day14 | Final Touches + GitHub README | Polish UI, finalize README.md, commit to GitHub |

**🧠 Learning Focus by Week**

📅 Week 1: Foundations

* C++ core structure, I/O, CLI parsing
* Hash function implementation
* Wordlist and brute-force attacks (single-threaded)
* Modular C++ design

📅 Week 2: Advanced Tool Building

* Threaded brute-force engine
* NTLM and wide string hashing
* Performance profiling & optimization
* GitHub publishing, ethical tool disclaimer

✅ Deliverables by End

* 🧱 Full-featured C++ password cracker: HashCrack++
* 🔧 Supports: MD5, SHA1, NTLM
* 🔐 Attacks: --attack=wordlist, --attack=brute
* ⚡ Multithreaded brute-force with custom charset/range
* 🖥️ Compiles via CMake on both Windows & Linux
* 📄 Proper README.md and .gitignore
* 🧪 Tested with real wordlists & known hashes

**Day 1:**

1.Run this command to compile:

**g++ src/main.cpp -o build/hashcrack.exe**

This tells g++ to compile main.cpp and output an executable in the build/ folder.

**📅 Day 1 Summary — Project Setup & Build Pipeline**

| **🧩 Task** | **✅ Description** | **💻 Command(s) Used** | **🛠️ Purpose** |
| --- | --- | --- | --- |
| Project Folder Setup | Created project directory and opened in VS Code | mkdir HashCrackPP cd HashCrackPP code . | Organized your workspace for the password cracker project |
| Git Initialization | Set up version control | git init | Track changes, commit history, and enable GitHub integration |
| .gitignore | Ignored unnecessary build/temp files | (manual file creation) | Keeps your Git repo clean (ignores .exe, /build/, etc.) |
| Folder Structure | Created standard C++ layout | mkdir src include wordlists build | Clean separation of source, headers, binaries, and wordlists |
| Basic Hello World | Created starter main.cpp | (manual file creation) | Verifies that compilation and setup works |
| Manual Build (g++) | Compiled using MinGW manually | g++ src/main.cpp -o build/hashcrack.exe | First build test — confirms C++ compiler works |
| Run Binary | Executed compiled file | ./build/hashcrack.exe | Confirmed output runs correctly |
| CMake Setup | Added cross-platform build support | (created CMakeLists.txt) | Enables modern, scalable build automation |
| CMake Configure | Generated build files for MinGW | cmake .. -G "MinGW Makefiles" | Instructs CMake to use MinGW instead of default NMake |
| CMake Build | Compiled using CMake | cmake --build . | Easy, scalable way to build full project (like VS solutions) |
| Run CMake Binary | Ran CMake-compiled file | ./crackjack.exe | Final confirmation that full build pipeline works |

**Day 2**

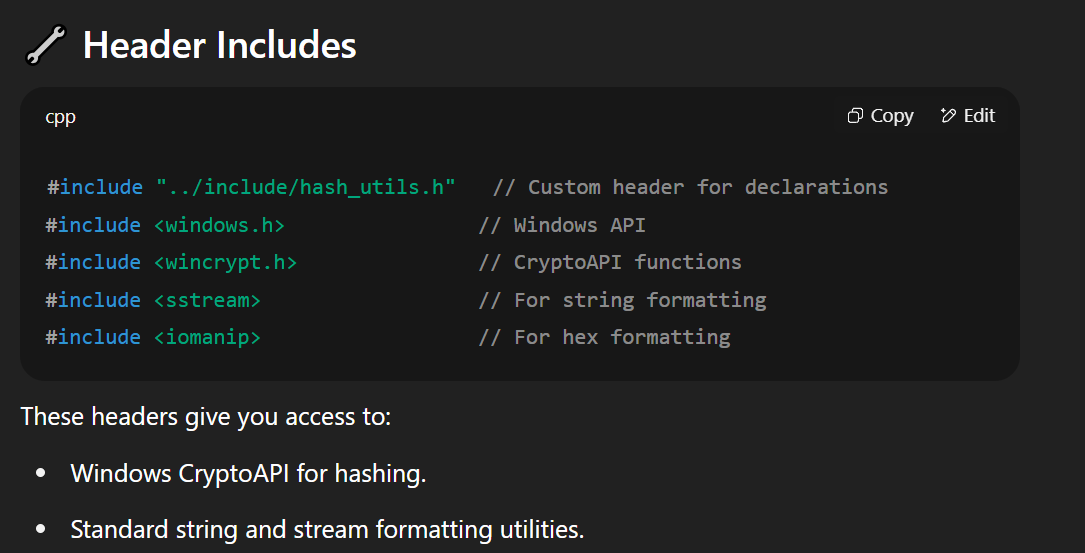
**Step 1: Create Header File for Hashing**

We'll define the interface for md5() and sha1().

1. In your include/ folder, create a new file:  
   hash\_utils.h

**Step 2: Create and Implement hash\_utils.cpp**

1. In your src/ folder, create a new file:  
   hash\_utils.cpp
2. Paste the following code using Windows CryptoAPI

****

**A screen shot of a computer program

AI-generated content may be incorrect.**

**🧠 Example:**

If hash = {0xAB, 0xCD}, output will be "abcd"

A computer screen shot of a program code

AI-generated content may be incorrect.

**📅 Day 2 Summary — Hashing Fundamentals**

| **✅ Task** | **Description** | **Result** |
| --- | --- | --- |
| Created hash\_utils.h/.cpp | Defined and implemented hashing functions | ✅ |
| Used Windows CryptoAPI | Secure, native hash calculations | ✅ |
| Implemented NTLM correctly | Converted to UTF-16LE + MD4 | ✅ |
| Updated main.cpp | Took input, printed all hashes | ✅ |
| Fixed CMake build issue | Included all .cpp files | ✅ |

**✅ Day 3: Summary Table**

| **✅ Task** | **Description** |
| --- | --- |
| wordlist\_cracker.h/.cpp | Created reusable cracking function |
| main.cpp CLI update | Added mode to crack a hash using wordlist |
| CMake build | Included all .cpp files in build |
| Crack tested (NTLM, MD5) | Successfully cracked known hashes |
| Wordlist flexible input | Supports custom paths like ../wordlists/test.txt |

**Sample Crack Commands**

**MD5:**

bash

CopyEdit

./crackjack --mode crack --algo md5 --hash 21232f297a57a5a743894a0e4a801fc3 --wordlist ../wordlists/test.txt

**SHA1:**

bash

CopyEdit

./crackjack --mode crack --algo sha1 --hash 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8 --wordlist ../wordlists/test.txt

**SHA256:**

bash

CopyEdit

./crackjack --mode crack --algo sha256 --hash 8d969eef6ecad3c29a3a629280e686cff8ca6d1b9ee9a2a8f7cd3cfbfa2b0a6c --wordlist ../wordlists/test.txt

**SHA512:**

bash

CopyEdit

./crackjack --mode crack --algo sha512 --hash ef92b7...3b5 --wordlist ../wordlists/test.txt

**NTLM:**

bash

CopyEdit

./crackjack --mode crack --algo ntlm --hash 68e109f0f40ca72a15e05cc22786f8e6 --wordlist ../wor