```
--------------------------------------------------------------------------------
================================================================================
HACKING C++
https://hackingcpp.com/cpp/beginners_guide.html
================================================================================

--------------------------------------------------------------------------------
Input & Output
--------------------------------------------------------------------------------

    ----------------------------------------------------------------------------
    Command Line Arguments
    ----------------------------------------------------------------------------

    -  What & Why?
        - Space-separated strings behind program call
        - Used to send information to a program when it starts

    - How To Access in C++

            $ main.cpp

                #include <iostream>

                int main (int const argc, char const*const* argv) {
                    for (int i = 0; i < argc; ++i) {
                        std::cout << argv[i] << '\n';
                    }
                }

        - Names "argc" and "argv" are only a convention     // ???
        - Each element of argv is a C-string:
            - A C-array of char
        - argv
            - Is a C-array of C-strings
        - argv[0]
            - Contains the program call (platform dependent)

    - Conversion to std::string, int, ...

                #include <iostream>

                int main (int const argc, char const*const* argv) {
                    if (argc < 3) {
                        std::cerr << "Usage: " << argv[0]
                                  << " <word> <times>\n";
                        return EXIT_FAILURE;
                    }

                    auto word  = std::string(argv[1]);
                    // atoi: convert string to integer
                    int  times = atoi(argv[2]);

                    for (int i = 0; i < times; ++i) {
                        std::cout << word << ' ';
                    }
                    std::cout << '\n';
                }

    - String -> Number Conversion Functions
        - C-sttings

                #include <cstdlib>

                int    atoi  (char const*);
                long   atoll (char const*);
                double atof  (char const*);
        - C++11

                #include <string>

                int    stoi (std::string const&);
                long   stol (std::string const&);
                float  stof (std::string const&);
                double stod (std::string const&);

    - Command Line Argument Parsing Libs
        - e.g., CLIPP
        - Check

                https://hackingcpp.com/cpp/lang/command_line_arguments.html


    ----------------------------------------------------------------------------
    File Input & Output
    ----------------------------------------------------------------------------

    - Write Text File
        - with std::ofstream (output file stream)

                #include <fstream>      // file stream header

                int main () {
                    std::ofstream os {"squares.txt"};   // open file

                    // if stream OK = can write to file
                    if (os.good()) {
                        for (int x = 1; x <= 6; ++x) {
                            // write x space x^2 newline
                            os << x << ' ' << (x*x) << '\n';
                        }
                    }
                }   // file automatically closed

    - Read Text File
        - with std::ifstream (input file stream)

                #include <iostream>
                #include <fstream>      // file stream header

                int main () {
                    std::ifstream is {"squares.txt"};   // open file

                    // if stream OK = file readable
                    if (is.good()) {
                        double x, y;
                        // as long as any 2 values readable
                        while (is >> x >> y) {
                            //print pairs (x,y)
                            cout << x << "^2 = " << y "\n";
                        }
                    }
                }   // file automatically closed

    - Open/Close Files
        - At creation/destruction

                int main (int const argc, char const*const* argv) {
                    if (argc > 1) {
                        // with C-string
                        std::ofstream os { argv[1] };
                        ...
                    } // automatically closed
                    else {
                        // with std::string C++11
                        std::string fn = "test.txt";
                        std::ofstream os { fn };
                        ...
                    } // automatically closed
                }

        - With open and close

                void bar () {
                    std::ofstream os;
                    os.open("squares.txt");
                    ...
                    os.close();
                    // open another file:
                    os.open("test.txt");
                    ...
                    os.close();
                }

    - File Open Modes
        - Default

                ifstream is {"in.txt", ios::in};
                ofstream os {"out.txt", ios::out};  (overwrite existing file)

        - Append to existing file

                ofstream os {"log.txt", ios::app};

        - Binary

                ifstream is {"in.jpg", ios::binary};
                ofstream os {"out.jpg", ios::binary};

            - Example

                #include <iostream>
                #include <fstream>
                #include <cstdint>

                int main (int argc, char* argv[]) {
                    if (argc < 3) {
                        std::cerr << "usage: " << argv[0] <<
                                  << " <integer> <filename>\n";
                        return 0;
                    }
                    std:string filename {argv[2]};
                    {   // write binary
                        std::uint64_t i = atoi(argv[1]);
                        std::cout << "writing: " << i << " to " << filename
                                  << '\n';
                        std::ofstream os {filename, std::ios::binary};
                        if (os.good()) {
                            os.write(reinterpret_cast<char const*>(&i),
                                sizeof(i));
                        }
                    }
                    {   // read binary
                        std::uint64_t i = 0;
                        std::ifstream is {filename, std::ios::binary};
                        if (is.good()) {
                            is.read(reinterpret_cast<char*>(&i),
                                sizeof(i));
                            std::cout << "read as: " << i << '\n';
                        }
                    }
                }
```

```
--------------------------------------------------------------------------------
Stream Input & Output
--------------------------------------------------------------------------------

Custom I/O

- Example: Point Coordinate I/O
    - By overloading two functions with names operator<< and operator>>

            struct point { int x; int y; };

            std::ostream& operator << (std::ostream& os, point const& p) {
                return os << '(' << p.x << ',' << p.y << ')';
            }

            std::istream& operator >> (std::istream& is, point& p) {
                return is >> p.x >> p.y;
            }

            point p {1,2};
            cout << p << '\n';  // prints (1,2)
            ...
            cin >> p;           // reads 2 ints into p.x and p.y
            ...

- Stream Operators
    - Operator functions for stream input/output of objects of type T:

            std::ostream operator << (std::ostream& os, T const& x) {
                // write to stream ...
                return os;
            }

            std::istream operator >> (std::istream& is, T& x) {
                // read from stream ...
                return is;
            }

        - Operators << and >> return a reference (to their stream parameter)
          to allow operator chaining:

            cin  >> x >> y;  <->  operator>>( operator>>(cin,  x), y)
            cout << x << y;  <->  operator<<( operator<<(cout, x), y)

        - There are no default stream operations in the standard library for
          containers like std::vector
            - Because there are too many possible use cases:
                - Just print values ... separated by what?
                - Format output as plain text / XML / ...
                - (De-)serialize container
                - ...

- (Some) Standard Library Stream Types

    istream         input stream        reference istream& binds to any
                                        other kind of std::input stream
    ostream         output stream       reference ostream& binds to any
                                        other kind of std::output stream
    ifstream        input file stream   extracted data is read from a file
    ofstream        output file stream  inserted data is stored in a file
    ostringstream   output string strm  inserted data is stored in a string
                                        buffer
    istringstream   input string strm   extracted data is read from a string
                                        buffer


Utilities
```

```
    - Read Lines With getline

                std::getline (istream&, string&, stopat='\n')

        - Reads until the next stopat character (default = end of line)

                string s;
                getline(cin, s);            // read entire line
                getline(cin, s, '\t');      // read until next tab
                getline(cin, s, 'a');       // read until next 'a'

    - Skip Forward With ignore

                std::istream::ignore(n, c)

        - Forwards stream by n characters
        - Until stop character c

                // skip next 8 characters
                cin.ignore(8);

                // skip by max. 10 characters or until after '='
                cin.ignore(10, '=');

                // skip until after next newline character
                //      needs: #include <limits>
                //      Does not work? outputs all chars before first ' '
                cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    - Formatting Manipulators

                #include <iomanip>

                std::setprecision(n)        // n number of digits
                std::fixed                  // fixed number of decimals
                std::scientific             // scientific notation
                std::boolalpha              // bools as strings


    ----------------------------------------------------------------------------
    Recover From Input Stream Errors
    ----------------------------------------------------------------------------

    What's The Problem?

    - Example: Successive Inputs

                int main() {
                    cout << "i? ";
                    int i = 0;
                    cin >> i;           // <- 1st

                    cout << "j? ";
                    int j = 0;
                    cint >> j;          // <- 2nd

                    cout << "i: " << i << ", "
                         << "j: " << j << '\n';
                }

        - Invalid input for i
            -> j not read!

    - Why Does This Happen?
        - If cin in the following code fragment

                    int i = 0;
```

```
                  cin >> i;

        reads characters that cannot be converted to an int:
          (1) cin’s FAILBIT is set
          (2) cin’s buffer content is NOT discarded and still contains the
              problematic input
          (3) any following attempt to read an int from cin will also fail

    Solution: Reset Input Stream After Error

          (1) Clear cin’s failbit
          (2) Clear cin’s input buffer

      - Example

              void reset_cin () {
                  // Clear all error status bits
                  cin.clear();
                  // Clear input buffer
                  cin.ignore(numeric_limits<streamsize>::max(), ’\n’);
              }

              int main () {
                  cout << "i? ";
                  int i = 0;
                  cin >> i;           // <- 1st

                  if (cin.fail()) reset_cin();

                  cout << "j? ";
                  int j = 0;
                  cin >> j;           // <- 2nd

                  cout << "i: " << i << ", "
                       << "j: " << j << ’\n’;
              }
```