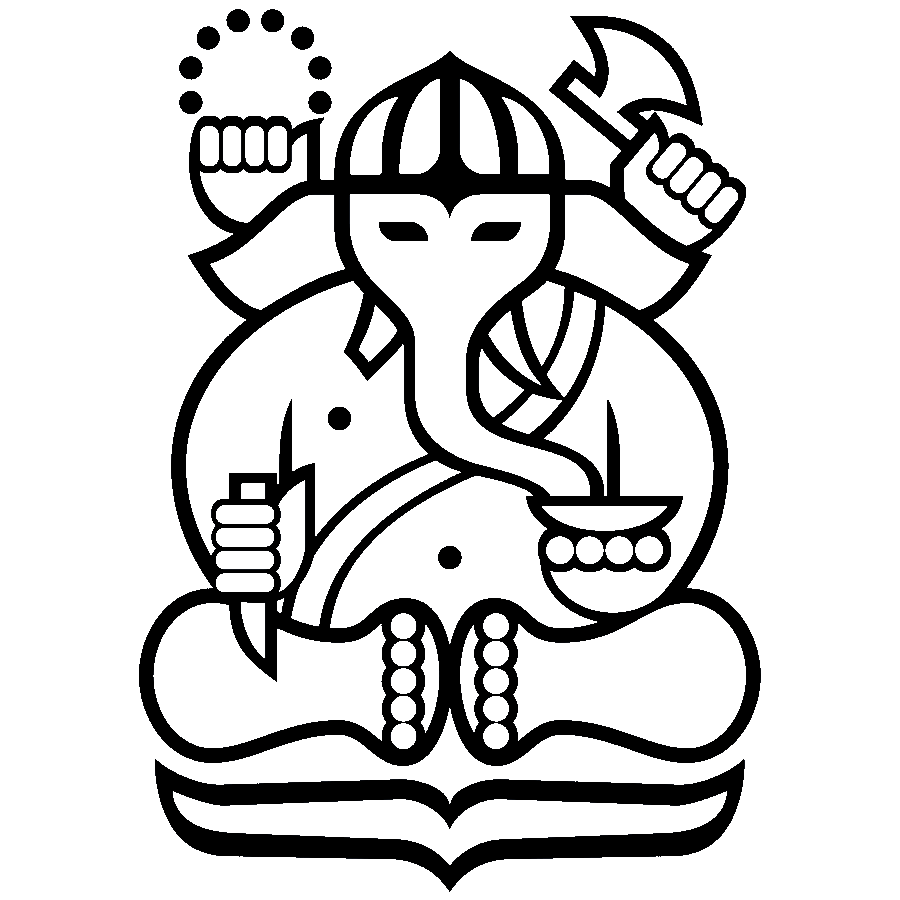
**IF2211 Strategi Algoritma**

**PROGRAM PENYELSAI PERMAINAN KARTU 24 DENGAN ALGORITMA BRUTE FORCE**

**Laporan Tugas Kecil**

****

Disusun oleh

Rinaldy Adin 13521134

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

## DAFTAR ISI

[DAFTAR ISI 2](#_Toc125535793)

[BAB I Pendahuluan dan Algoritma Penyelesaian 3](#_Toc125535794)

[1. Pendahuluan 3](#_Toc125535795)

[2. Algoritma Penyelesaian 3](#_Toc125535796)

[BAB II Source Code Program 6](#_Toc125535797)

[BAB III Hasil Pengujian 21](#_Toc125535798)

[BAB IV Cek List Program 35](#_Toc125535799)

[LAMPIRAN 36](#_Toc125535800)

## 

## BAB I Pendahuluan dan Algoritma Penyelesaian

### Pendahuluan

Permainan kartu 24 adalah permainan kartu aritmatika dengan tujuan mencari cara untuk mengubah 4 buah angka random sehingga mendapatkan hasil akhir sejumlah 24. Permainan Kartu 24 biasa dimainkan dengan menggunakan kartu remi. Kartu remi terdiri dari 52 kartu yang terbagi menjadi empat suit (sekop, hati, keriting, dan wajik) yang masing-masing terdiri dari 13 kartu (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). Yang perlu diperhatikan hanyalah nilai kartu yang didapat (As, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, dan King). As bernilai 1, Jack bernilai 11, Queen bernilai 12, King bernilai 13, sedangkan kartu bilangan memiliki nilai dari bilangan itu sendiri. Pada awal permainan moderator atau salah satu pemain mengambil 4 kartu dari dek yang sudah dikocok secara random. Permainan berakhir ketika pemain berhasil menemukan solusi untuk membuat kumpulan nilainya menjadi 24. Pengubahan nilai tersebut dapat dilakukan menggunakan operasi dasar matematika penjumlahan (+), pengurangan (-), perkalian (×), pembagian (/) dan tanda kurung ( () ). Tiap kartu harus digunakan tepat sekali dan urutan penggunaannya bebas.

### Algoritma Penyelesaian

Program yang telah dibuat bertujuan menyelesaikan permasalahan ini dengan menampilkan semua kemungkinan persamaan/solusi dari pilihan kartu yang dapat dihasilkan secara acak oleh program atau didapatkan dari masukan pengguna. Untuk memperoleh semua kemungkinan solusi, program menggunakan algoritma dengan strategi *brute force*, yaitu algoritma yang mencoba semua kemungkinan yang ada hingga mendapatkan hasil yang diinginkan. Dalam kasus ini, semua kemungkinan yang ada merupakan semua kemungkinan/permutasi persamaan yang terdiri dari empat angka dan tiga operator, sedangkan hasil yang diinginkan adalah semua persamaan yang menghasilkan nilai 24. Oleh karena itu, algoritma penyelesaian dapat dibagi menjadi algoritma penghasil permutasi persamaan dan algoritma menghitung nilai dari suatu persamaan.

Dalam implementasi program ini, permasalahan memperoleh semua permutasi yang ada dipecahkan dengan mencari permutasi urutan angka, serta permutasi susunan angka dan operator yang direpresentasikan dalam bentuk prefix, karena urutan operasi dalam representasi prefix sudah inheren dan tidak perlu ditambahkan kurung seperti representasi infix. Oleh karena itu, algoritma penyelesaian program ini dapat dijabarkan sebagai berikut,

1. Pencarian permutasi persamaan, yang dapat dibagi menjadi
   1. Pencarian permutasi urutan angka, yang terdiri dari langkah-langkah
      1. Menerima masukan 4 angka yang dapat digunakan.
      2. Melakukan nested looping dengan kedalaman 4 yang akan mencoba semua permutasi angka, program akan mencatat angka sebelumnya yang sudah digunakan agar permutasi berupa acakan dari angka masukan.
      3. Permutasi yang dihasilkan akan dimasukkan ke dalam array hasil, tetapi sebelum dimasukkan ke dalam array, diperiksa terlebih dahulu apakah permutasi tersebut sudah tercatat dalam array atau belum, jika belum permutasi dimasukkan ke dalam array hasil.
   2. Pencarian permutasi susunan angka dan operator, yang terdiri dari langkah-langkah:
      1. Memanggil fungsi rekursi yang akan menghasilkan persamaan prefix yang diawali dengan operator *op*. Dengan *op* berupa operator tambah (+), kurang (-), kali (×), dan bagi (/).
      2. Dalam fungsi tersebut, akan terjadi rekursi yang akan menghasilkan kemungkinan susunan persamaan yang direpresentasikan secara prefix dengan variabel , , , dan sebagai placeholder/pengganti untuk angka. Contoh, “”.
      3. Fungsi rekursi menerima masukan elemen yang ingin dimasukkan ke dalam susunan, variabel yang belum terpakai, urutan/index saat ini, susunan yang sedang dihasilkan, jumlah operator, dan jumlah variabel/angka.
      4. Aturan pemasukkan elemen berdasarkan pengamatan bahwa dalam persamaan prefix, jumlah operator dari awal hingga suatu elemen, kecuali elemen terakhir, selalu lebih banyak atau sama dengan jumlah angka. Oleh karena itu, pemasukkan elemen mengikuti aturan berikut,
         1. Apabila jumlah operator kurang dari 3, masukkan operator ke dalam index berikutnya.
         2. Apabila jumlah angka kurang dari jumlah operator, masukkan angka ke dalam index berikutya.
         3. Apabila elemen berikutnya merupakan elemen terakhir, masukkan angka ke dalam index berikutya.
      5. Setiap permutasi susunan operator yang dihasilkan dimasukkan ke dalam array hasil susunan operator.
2. Menghitung nilai dari suatu persamaan yang dihasilkan, yang terdiri dari langkah-langkah
   1. Hasilkan permutasi persamaan dengan memasukkan permutasi angka ke dalam permutasi susunan operator.
   2. Evaluasi nilai dari persamaan tersebut dengan memanfaatkan struktur data stack. Program melakukan itertasi elemen persamaan dari akhir ke awal
      1. Jika elemen yang ditemukan merupakan angka, masukkan angka tersebut ke dalam stack.
      2. Jika elemen yang ditemukan merupakan operator, jalan kan operasi tersebut terhadap 2 elemen teratas pada stack, hilangkan 2 elemen tersebut dari stack, dan masukkan hasil operasi ke dalam stack. Misal ditemukan operator bagi (/) dan elemen teratas pada stack adalah x dan elemen kedua teratas adalah y, maka x dan y akan dihilangkan dari stack, lalu nilai dari y / x akan dimasukkan ke dalam stack.
   3. Jika permutasi persamaan tersebut menghasilkan nilai 24, ubahlah terlebih dahulu persamaan tersebut ke dalam bentuk infix, untuk melakukan hal tersebut program menggunakan struktur data stack yang menyimpan data string dan akan melakukan itertasi elemen persamaan dari akhir ke awal
      1. Jika elemen yang ditemukan merupakan angka, *casting* angka tersebut menjadi string dan masukkan ke dalam stack.
      2. Jika elemen yang ditemukan merupakan operator, hilangkan dan simpan nilai 2 elemen teratas pada stack, lakukan konkatenasi sehingga menghasilkan string persamaan infix antara dua elemen tersebut. Jika iterasi belum sampai elemen pertama tambahkan tanda kurung pada persamaan tersebut.
   4. Masukkan string persamaan infix ke dalam list yang menyimpan persamaan-persamaan yang menghasilkan nilai 24.

## BAB II Source Code Program

Source code program disimpan dalam repository github <https://github.com/Rinaldy-Adin/Tucil1_13521134>. Source code serta file header adalah sebagai berikut,

|  |
| --- |
| main.cpp |
| #include <chrono>  #include <iomanip>  #include <iostream>  #include <vector>  #include "cliui.h"  #include "evaluate\_exp.h"  #include "generate\_exp.h"  #include "generate\_perms.h"  using std::cout;  using std::endl;  using std::vector;  int main() {  *// Splash screen on start*      splashScreen();      cout << endl << endl;  *// Initialize loop variables*      bool programRunning = true;      vector<string> menuChoices = {"User input", "Random", "Exit program"};      while (programRunning) {  *// Input method menu*          header("Menu");          cout << endl;          int choice = menuPrompt("Choose card input method:", menuChoices);          if (choice == 1 || choice == 2) { *// User chooses to continue*  *// Initialize permutation variables*              int cardNums[4];              int permutations[25][4];              char possibleExpressions[400][7];              int nPermutations;              int nExpressions;              for (int i = 0; i < 25; i++) {                  for (int j = 0; j < 4; j++) {                      permutations[i][j] = 0;                  }              }              for (int i = 0; i < 400; i++) {                  for (int j = 0; j < 7; j++) {                      possibleExpressions[i][j] = 0;                  }              }              if (choice == 1) { *// Cards chosen by user*                  cout << "\nEnter 4 card values separated by spaces (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, "                          "J, Q, K):\n";                  readInput(cardNums);                  cout << endl;              } else { *// Cards randomized*                  for (int i = 0; i < 4; i++) {                      cardNums[i] = rand() % 13 + 1;                  }                  cout << endl << endl;              }              if (cardNums[0] != -1) { *// Input valid*  *// Display cards chosen by user/random*                  header("Cards");                  printCards(cardNums);                  waitForEnter();                  cout << endl << endl;  *// Start algorithm clock*                  auto begin = std::chrono::high\_resolution\_clock::now();  *// Generate & evaluate expressions equal to 24*                  generatePermutations(cardNums, permutations, &nPermutations);                  generateExpressions(possibleExpressions, &nExpressions);                  vector<string> expressions = evaluateExpressions(permutations,  nPermutations, possibleExpressions, nExpressions);  *// End algorithm clock and calculate elapsed time*                  auto end = std::chrono::high\_resolution\_clock::now();                  auto elapsed = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);  *// Display solutions*                  header("Solutions");                  cout << expressions.size() << " solutions found:" << endl << endl;                  for (string expression : expressions) {                      cout << expression << endl;                  }  *// Display elapsed time*                  cout << "\nSolution algorithm elapsed for " << std::fixed << std::setprecision(3)                       << elapsed.count() \* 1e-6 << " miliseconds\n";  *// Ask user to save to file*                  askToSave(cardNums, expressions, elapsed.count());              } else { *// Input Invalid*  *// Return to menu*                  cout << "\nInput invalid\n";                  waitForEnter();              }              cout << endl << endl;          } else { *// User chooses to exit program*              programRunning = false;          }      }      return 0;  } |

|  |
| --- |
| cliui.h |
| #include <string>  #include <vector>  using std::string;  using std::vector;  *// Read cards based off user input*  void readInput(int *cardNums*[]);  *// Display splash screen*  void splashScreen();  *// Display header with centered title*  void header(string *str*);  *// Display menu and recieve valid input*  int menuPrompt(string *prompt*, vector<string> *choices*);  *// Display cards*  void printCards(int *cardNums*[]);  *// Ask for user to press enter to continue program*  void waitForEnter();  *// Ask for user to save solution results into file*  void askToSave(int *cardNums*[], vector<string> *solutions*, int *nanoseconds*); |
| cliui.cpp |
| #include "cliui.h"  #include <cstdio>  #include <fstream>  #include <iomanip>  #include <iostream>  #include <limits>  #include <string>  using std::cin;  using std::cout;  using std::endl;  using std::getline;  using std::ifstream;  using std::ofstream;  using std::string;  *// Read cards based off user input*  void readInput(int *cardNums*[]) {  *// Initialize variables*      int i = 0;      int idx = 0;      bool spaceSeparated = true;  *// Read line from input as string*      string str;      getline(cin >> std::ws, str);      char c;  *// Iterate over string to ensure valid input*      while (idx < str.length()) {          c = str[idx];          if (c > '1' && c <= '9' && spaceSeparated) { *// Card values under 10*  *cardNums*[i % 4] = c - '0';              spaceSeparated = false;          } else if (c == '1' && spaceSeparated) { *// Handle 10*              idx++;              c = str[idx];              if (c == '0') {  *cardNums*[i % 4] = 10;                  spaceSeparated = false;              } else {  *cardNums*[0] = *cardNums*[1] = *cardNums*[2] = *cardNums*[3] = -1;                  return;              }          } else { *// Handle A J Q K and space character*              if ((c == 'A' || c == 'a') && spaceSeparated) {  *cardNums*[i % 4] = 1;                  spaceSeparated = false;              } else if ((c == 'J' || c == 'j') && spaceSeparated) {  *cardNums*[i % 4] = 11;                  spaceSeparated = false;              } else if ((c == 'Q' || c == 'q') && spaceSeparated) {  *cardNums*[i % 4] = 12;                  spaceSeparated = false;              } else if ((c == 'K' || c == 'k') && spaceSeparated) {  *cardNums*[i % 4] = 13;                  spaceSeparated = false;              } else if (c == ' ') {                  spaceSeparated = true;                  i--;              } else {  *cardNums*[0] = *cardNums*[1] = *cardNums*[2] = *cardNums*[3] = -1;              }          }          i++;          idx++;      }  *// Return invalid input if the number of cards given is not 4*      if (i != 4 || idx < str.length()) {  *cardNums*[0] = *cardNums*[1] = *cardNums*[2] = *cardNums*[3] = -1;          return;      }  }  *// Display splash screen*  void splashScreen() {      cout << R"(  .\_\_\_  \_\_\_.      \_\_\_       \_\_  \_\_\_  \_\_\_\_\_\_\_     \_\_  .\_\_\_\_\_\_\_\_\_\_\_.    \_\_\_    \_  \_  |   \/   |     /   \     |  |/  / |   \_\_\_\_|   |  | |           |   |\_\_ \  | || |  |  \  /  |    /  ^  \    |  '  /  |  |\_\_      |  | `---|  |----`      ) | | || |\_  |  |\/|  |   /  /\_\  \   |    <   |   \_\_|     |  |     |  |          / /  |\_\_   \_|  |  |  |  |  /  \_\_\_\_\_  \  |  .  \  |  |\_\_\_\_    |  |     |  |         / /\_     | |  |\_\_|  |\_\_| /\_\_/     \\_\_\ |\_\_|\\_\_\ |\_\_\_\_\_\_\_|   |\_\_|     |\_\_|        |\_\_\_\_|    |\_|      )" << endl;  }  *// Display header with centered title*  void header(string *str*) {      for (int i = 0; i < 20; i++) {          cout << "=";      }      cout << "\n";      cout << "==";      for (int i = 2; i < 10 - (*str*.length() / 2); i++) {          cout << " ";      }      cout << *str*;      for (int i = 10 - (*str*.length() / 2) + *str*.length(); i < 18; i++) {          cout << " ";      }      cout << "==\n";      for (int i = 0; i < 20; i++) {          cout << "=";      }      cout << "\n";  }  *// Display menu and recieve valid input*  int menuPrompt(string *prompt*, vector<string> *choices*) {      while (true) {          cout << *prompt* << endl;          for (int i = 1; i <= *choices*.size(); i++) {              cout << i << ". ";              cout << *choices*[i - 1] << endl;          }          cout << endl;          cout << "Enter command (";          for (int i = 1; i <= *choices*.size(); i++) {              cout << i;              if (i != *choices*.size())                  cout << "/";          }          cout << "):" << endl;          int choice;          cin >> choice;          if (choice < 1 || choice > *choices*.size()) {              cout << endl << "Invalid input" << endl << endl;          } else {              cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');              return choice;          }      }  }  *// Display cards*  void printCards(int *cardNums*[]) {      cout << "----- ----- ----- -----" << endl;      cout << "|   | |   | |   | |   |" << endl;      for (int i = 0; i < 4; i++) {          cout << "|";          if (*cardNums*[i] == 11) {              cout << " J";          } else if (*cardNums*[i] == 1) {              cout << " A";          } else if (*cardNums*[i] == 12) {              cout << " Q";          } else if (*cardNums*[i] == 13) {              cout << " K";          } else if (*cardNums*[i] == 10) {              cout << "10";          } else {              cout << " " << *cardNums*[i];          }          cout << " | ";      }      cout << endl;      cout << "|   | |   | |   | |   |" << endl;      cout << "----- ----- ----- -----" << endl;  }  *// Ask for user to press enter to continue program*  void waitForEnter() {      cout << "Continue? (press enter)";      cin.ignore(INT\_MAX, '\n');  }  *// Ask for user to save solution results into file*  void askToSave(int *cardNums*[], vector<string> *solutions*, int *nanoseconds*) {      string str;      do { *// Validate input*          cout << "\nWould you like to save your results to a file?(y/n): ";          getline(cin >> std::ws, str);      } while (str[0] != 'y' && str[0] != 'Y' && str[0] != 'n' && str[0] != 'N');  *// Write to file if user inputs Y*      if (str[0] == 'y' && str[0] != 'Y') {          do {              cout << "Enter the path to your file: ";              getline(cin >> std::ws, str);              ofstream outfile;              try {                  outfile.open(str);                  outfile << "Cards: ";                  for (int i = 0; i < 4; i++) {                      switch (*cardNums*[i]) {                          case 1:                              outfile << "A ";                              break;                          case 11:                              outfile << "J ";                              break;                          case 12:                              outfile << "Q ";                              break;                          case 13:                              outfile << "K ";                              break;                          default:                              outfile << *cardNums*[i] << " ";                              break;                      }                  }                  outfile << endl << endl;                  outfile << *solutions*.size() << " solutions found" << endl << endl;                  for (string solution : *solutions*) {                      outfile << solution << endl;                  }                  outfile << "\nSolution algorithm elapsed for " << std::fixed << std::setprecision(3)                          << *nanoseconds* \* 1e-6 << " miliseconds" << endl;                  outfile.close();                  break;              } catch (const ofstream::failure& e) {                  outfile.close();                  cout << "Filepath not valid" << endl;              }          } while (true);          cout << "\nFile successfully written!" << endl;      }  } |

|  |
| --- |
| evaluate\_exp.h |
| #include <string>  #include <vector>  using std::string;  using std::vector;  *// Evaluate expressions based off generated card permutations and possible prefix expressions*  vector<string> evaluateExpressions(int *permutations*[][4], int *nPerms*, char *possibleExpressions*[][7],                                     int *nExps*); |
| evaluate\_exp.cpp |
| #include "evaluate\_exp.h"  #include <stack>  #include <string>  #include <vector>  using std::stack;  using std::string;  using std::to\_string;  using std::vector;  *// Return value of prefix equation*  double evaluatePrefix(char *mappedExp*[]) {      stack<double> Stack;  *// Iterate from back to front*      for (int i = 6; i >= 0; i--) {          if (*mappedExp*[i] > '0' &&  *mappedExp*[i] <= '9') { *// Push numericals into stack & cast to double*              Stack.push((double)*mappedExp*[i] - '0');          } else if (*mappedExp*[i] >= 'a' &&  *mappedExp*[i] < 'w') { *// Push numericals into stack & cast to double*              Stack.push((double)*mappedExp*[i] - 'a' + 10);          } else { *// Pop & calculate top two numericals and push result into stack when operator*  *// found in*              double y = Stack.top();              Stack.pop();              double x = Stack.top();              Stack.pop();              if (*mappedExp*[i] == '\*') {                  Stack.push(x \* y);              } else if (*mappedExp*[i] == '/') {                  Stack.push(x / y);              } else if (*mappedExp*[i] == '+') {                  Stack.push(x + y);              } else if (*mappedExp*[i] == '-') {                  Stack.push(x - y);              }          }      }      return Stack.top();  }  *// Convert prefix array of chars into infix string*  string convertToInfix(char *prefix*[]) {      stack<string> Stack;  *// Iterate from back to front*      for (int i = 6; i >= 0; i--) {          if (*prefix*[i] > '0' && *prefix*[i] <= '9') { *// Push numericals into stack & cast to string*              string s(1, *prefix*[i]);              Stack.push(s);          } else if (*prefix*[i] >= 'a' &&  *prefix*[i] < 'w') { *// Push numericals into stack & cast to string*              Stack.push(to\_string(*prefix*[i] - 'a' + 10));          } else { *// Pop top two numericals and push infix expressions into stack when operator*  *// found in prefix*              string y = Stack.top();              Stack.pop();              string x = Stack.top();              Stack.pop();              string z;              if (i != 0) {                  z = "(" + x + " " + *prefix*[i] + " " + y + ")";              } else {                  z = x + " " + *prefix*[i] + " " + y;              }              Stack.push(z);          }      }      return Stack.top();  }  *// Evaluate expressions based off generated card permutations and possible prefix expressions*  vector<string> evaluateExpressions(int *permutations*[][4], int *nPerms*, char *possibleExpressions*[][7],                                     int *nExps*) {      vector<string> infixes;  *// Iterate over permutations of cards and expressions*      for (int currPerm = 0; currPerm < *nPerms*; currPerm++) {          for (int currExp = 0; currExp < *nExps*; currExp++) {              char mappedExp[7];  *// Cast permutations of type int to characters representing hexadecimals*              for (int k = 0; k < 7; k++) {                  char mappedOp = *possibleExpressions*[currExp][k];                  if (mappedOp >= 'w' && mappedOp <= 'z') {                      mappedExp[k] = *permutations*[currPerm][mappedOp - 'w'] + '0';                      if (mappedExp[k] > '9') {                          mappedExp[k] = mappedExp[k] - '9' - 1 + 'a';                      }                  } else {                      mappedExp[k] = mappedOp;                  }              }  *// Evaluate permutation and append to solutions list if equals 24*              if (evaluatePrefix(mappedExp) == 24) {                  infixes.push\_back(convertToInfix(mappedExp));              }          }      }      return infixes;  } |

|  |
| --- |
| generate\_exp.h |
| *// Generate possible expressions*  *// with w, x, y, and z as placeholders for numerical values*  void generateExpressions(char *possibleExpressions*[][7], int \**nExpressions*); |
| Generate\_exp.cpp |
| #include "generate\_exp.h"  *// Generate prefix with 4 numbers and 3 operators, following the rule that up until the last*  *// operand, there are more or equal operators than operands*  void generatePrefix(char *possibleExpressions*[][7], char *toInsert*, char *unusedVar*, char *generated*[],                      int *generatedIdx*, int *nOperators*, int *nNums*, int \**nExpressions*) {  *generated*[*generatedIdx*] = *toInsert*;  *// Prefix expression is complete*      if (*generatedIdx* == 6) {  *// Insert into array*          for (int i = 0; i < 7; i++) {  *possibleExpressions*[\**nExpressions*][i] = *generated*[i];          }          \**nExpressions* += 1;      }      if (*nOperators* < 3) { *// Generate operators*          generatePrefix(*possibleExpressions*, '\*', *unusedVar*, *generated*, *generatedIdx* + 1,  *nOperators* + 1, *nNums*, *nExpressions*);          generatePrefix(*possibleExpressions*, '/', *unusedVar*, *generated*, *generatedIdx* + 1,  *nOperators* + 1, *nNums*, *nExpressions*);          generatePrefix(*possibleExpressions*, '+', *unusedVar*, *generated*, *generatedIdx* + 1,  *nOperators* + 1, *nNums*, *nExpressions*);          generatePrefix(*possibleExpressions*, '-', *unusedVar*, *generated*, *generatedIdx* + 1,  *nOperators* + 1, *nNums*, *nExpressions*);      }  *// Generate numbers when possible or when last operand*      if (*nNums* < *nOperators* || *generatedIdx* == 5)          generatePrefix(*possibleExpressions*, *unusedVar*, *unusedVar* + 1, *generated*, *generatedIdx* + 1,  *nOperators*, *nNums* + 1, *nExpressions*);  }  *// Generate possible expressions with w, x, y, and z as placeholders for numerical values*  void generateExpressions(char *possibleExpressions*[][7], int \**nExpressions*) {      char generated[7];      \**nExpressions* = 0;      generatePrefix(*possibleExpressions*, '\*', 'w', generated, 0, 1, 0, *nExpressions*);      generatePrefix(*possibleExpressions*, '/', 'w', generated, 0, 1, 0, *nExpressions*);      generatePrefix(*possibleExpressions*, '+', 'w', generated, 0, 1, 0, *nExpressions*);      generatePrefix(*possibleExpressions*, '-', 'w', generated, 0, 1, 0, *nExpressions*);  } |

|  |
| --- |
| generate\_perms.h |
| *// Generate permutations of the recieved cards*  void generatePermutations(int *cardNums*[], int *permutations*[][4], int \**nPermutations*); |
| generate\_perms.cpp |
| #include "generate\_perms.h"  *// Check if permutation is duplicate*  bool isInPermutations(int *permutation*[], int *permutations*[][4], int *last\_empty\_idx*) {      int i = 0;  *// Iterate over generated permutations*      while (i < *last\_empty\_idx*) {          if (*permutation*[0] == *permutations*[i][0] && *permutation*[1] == *permutations*[i][1] &&  *permutation*[2] == *permutations*[i][2] && *permutation*[3] == *permutations*[i][3]) {              return true;          }          i++;      }      return false;  }  *// Generate permutations of the recieved cards*  void generatePermutations(int *cardNums*[], int *permutations*[][4], int \**nPermutations*) {      \**nPermutations* = 0;      for (int a = 0; a < 4; a++) {  *// Initialize generated to store generated permutation &*  *// Initialize used to keep track of used cards*          int generated[4];          bool used[4] = {0, 0, 0, 0};          generated[0] = *cardNums*[a];          used[a] = 1;          for (int b = 0; b < 4; b++) {              if (used[b])                  continue;              generated[1] = *cardNums*[b];              used[b] = 1;              for (int c = 0; c < 4; c++) {                  if (used[c])                      continue;                  generated[2] = *cardNums*[c];                  used[c] = 1;                  for (int d = 0; d < 4; d++) {                      if (used[d])                          continue;                      generated[3] = *cardNums*[d];                      used[d] = 1;  *// Check if permutation is duplicate*                      if (!isInPermutations(generated, *permutations*, \**nPermutations*)) {  *// Insert to array if permutation is not duplicate*                          for (int i = 0; i < 4; i++) {  *permutations*[\**nPermutations*][i] = generated[i];                          }                          \**nPermutations* += 1;                      }                      used[d] = 0;                  }                  used[c] = 0;              }              used[b] = 0;          }      }  } |

## BAB III Hasil Pengujian

|  |
| --- |
| **Test Case #1** |
|  |

|  |
| --- |
| **Test Case #2** |
|  |

|  |
| --- |
| **Test Case #3** |
|  |

|  |
| --- |
| **Test Case #4** |
|  |

|  |
| --- |
| **Test Case #5** |
|  |

|  |
| --- |
| **Test Case #6** |
|  |

## BAB IV Cek List Program

|  |  |  |  |
| --- | --- | --- | --- |
| No | Poin | Ya | Tidak |
| 1 | Program berhasil dikompilasi tanpa  Kesalahan | ✓ |  |
| 2 | Program berhasil *running* | ✓ |  |
| 3 | Program dapat membaca input / generate  sendiri dan memberikan luaran | ✓ |  |
| 4 | Solusi yang diberikan program memenuhi  (berhasil mencapai 24) | ✓ |  |
| 5 | Program dapat menyimpan solusi dalam file  teks | ✓ |  |

## LAMPIRAN

**Repositori Github :**

<https://github.com/Rinaldy-Adin/Tucil1_13521134>