**LAPORAN PRATIKUM STRUKTUR DATA**

**JOBSHEET 4**



**Disusun Oleh :**

**NAJWA ELDIARA OWILIA TIKSA**

**BP/NIM : 2023/23343079**

**Dosen Pengampu :**

**Randi Proska Sandra, S.Pd, M.Sc**

**Kode Kelas : 202323430157**

**PROGRAM STUDI S1 INFORMATIKA**

**DEPARTEMEN ELEKTRONIKA**

**FAKULITAS TEKNIK**

**UNIVERSITAS NEGERI PADANG**

**2023**

**1.**

1. **Source Code**

//created by 23343079\_Najwa Eldiara Owilia Tiksa

#include <stdio.h>

#include <stdlib.h>

void generateFibonacci(int n, int\* fibonacciArray) {

if (n <= 0) {

printf("Masukkan n harus lebih besar dari 0.\n");

return;

}

fibonacciArray[0] = 0;

if (n > 1) {

fibonacciArray[1] = 1;

for (int i = 2; i < n; i++) {

fibonacciArray[i] = fibonacciArray[i - 1] + fibonacciArray[i - 2];

}

}

}

int main() {

int n;

printf("Masukkan nilai n untuk deret Fibonacci: ");

scanf("%d", &n);

if (n <= 0) {

printf("Masukkan n harus lebih besar dari 0.\n");

return 1;

}

// Menggunakan malloc untuk mengalokasikan memori

int\* fibonacciArray = (int\*)malloc(n \* sizeof(int));

// Pengecekan apakah alokasi memori berhasil

if (fibonacciArray == NULL) {

printf("Gagal melakukan alokasi memori.\n");

return 1;

}

// Memanggil fungsi untuk mengisi nilai deret Fibonacci

generateFibonacci(n, fibonacciArray);

// Menampilkan deret Fibonacci

printf("Deret Fibonacci pertama %d angka:\n", n);

for (int i = 0; i < n; i++) {

printf("%d ", fibonacciArray[i]);

}

printf("\n");

// Menggunakan free untuk membebaskan memori yang dialokasikan

free(fibonacciArray);

return 0;

}

1. **Penjelasan Program**

Program ini ditulis dalam bahasa C dan bertujuan untuk menghasilkan deret Fibonacci sebanyak \( n \) elemen, di mana \( n \) adalah input dari pengguna. Berikut adalah penjelasan rinci dari program ini

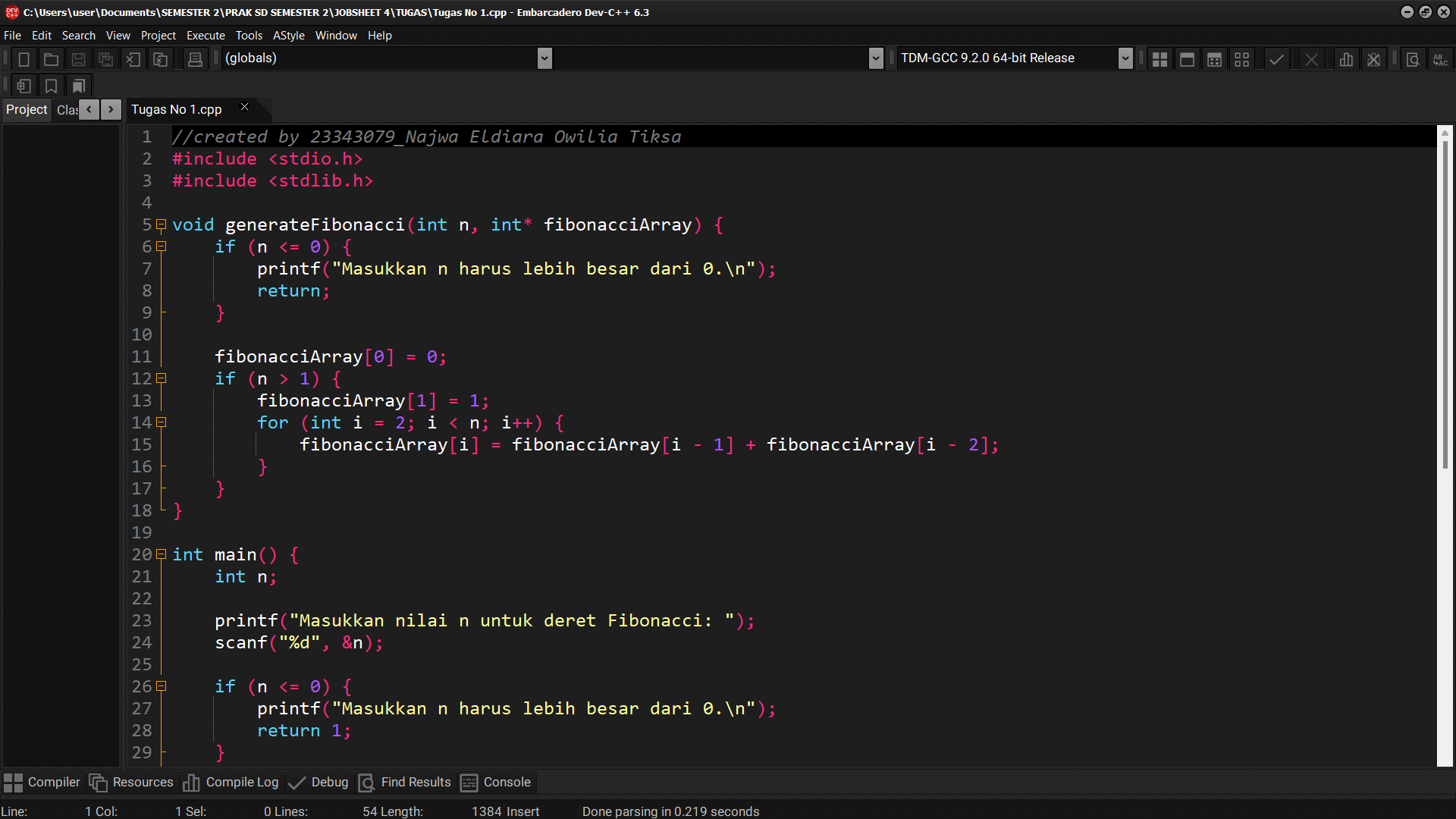
Algoritma yang Digunakan:

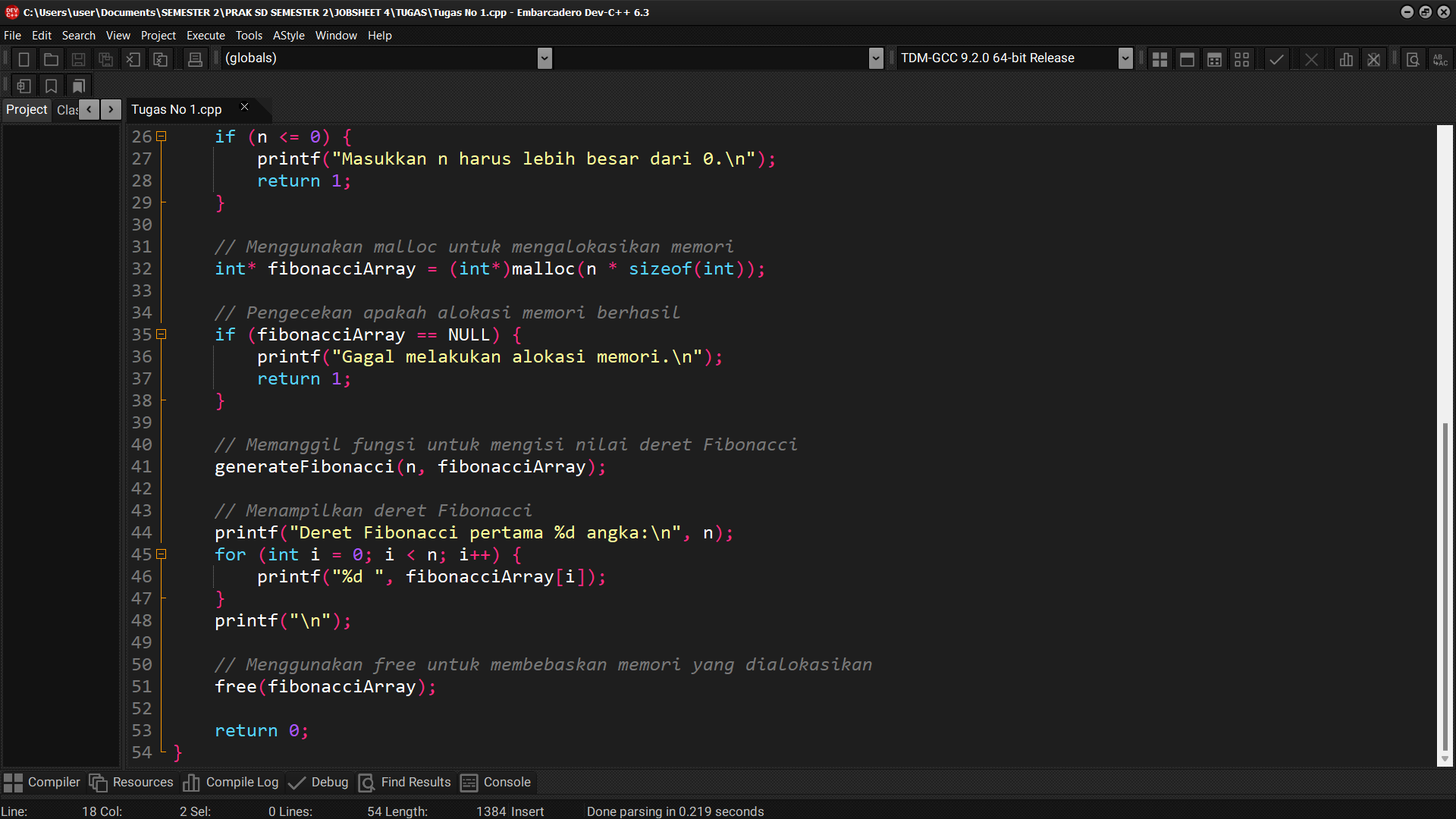
Program ini menggunakan algoritma iteratif untuk menghasilkan deret Fibonacci. Secara khusus, algoritma ini bekerja dengan cara berikut:

1. Menetapkan dua nilai pertama dari deret Fibonacci (0 dan 1).
2. Menggunakan perulangan `for` untuk menghitung setiap elemen berikutnya sebagai jumlah dari dua elemen sebelumnya.

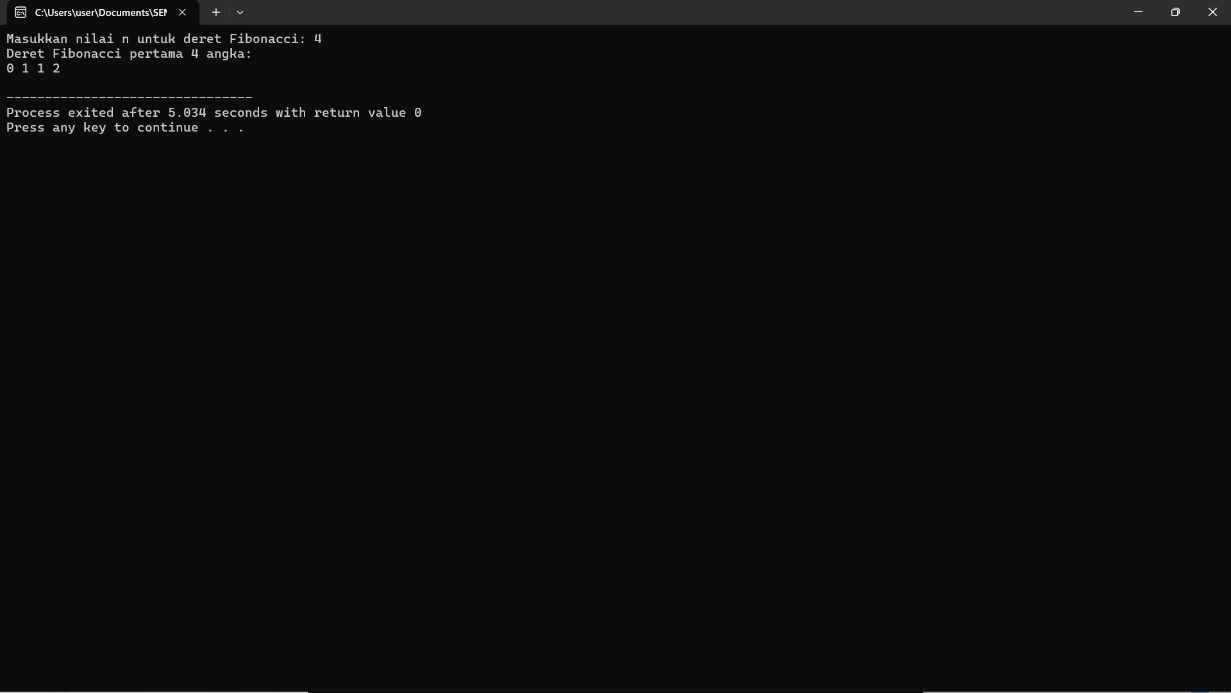
Algoritma iteratif ini lebih efisien dalam hal penggunaan memori dan waktu eksekusi dibandingkan dengan algoritma rekursif, terutama untuk nilai \( n \) yang besar. Hal ini karena algoritma iteratif hanya membutuhkan ruang memori yang sebanding dengan jumlah elemen yang dihasilkan dan tidak memiliki overhead dari pemanggilan fungsi rekursif berulang-ulang.

1. **Screenshot program**





1. **Output**



**2.**

1. **Source Code**

//created by 23343079\_Najwa Eldiara Owilia Tiksa

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Fungsi untuk mengecek apakah suatu bilangan adalah prima

bool isPrime(int num) {

if (num < 2) {

return false;

}

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

// Fungsi untuk menghasilkan bilangan prima pertama sampai ke-n

void generatePrimes(int n, int\*\* primeArray, int\* size) {

int count = 0;

int\* primes = \*primeArray;

for (int num = 2; count < n; num++) {

if (isPrime(num)) {

// Gunakan realloc untuk mengubah ukuran memori yang dialokasikan

\*primeArray = (int\*)realloc(\*primeArray, (\*size + 1) \* sizeof(int));

if (\*primeArray == NULL) {

printf("Gagal melakukan realokasi memori.\n");

exit(1);

}

primes = \*primeArray;

primes[\*size] = num;

(\*size)++;

count++;

}

}

}

int main() {

int n, m;

int\* primeArray = NULL;

int size = 0;

printf("Masukkan nilai n untuk bilangan prima: ");

scanf("%d", &n);

if (n <= 0) {

printf("Masukkan n harus lebih besar dari 0.\n");

return 1;

}

printf("Masukkan nilai m untuk alokasi awal memori: ");

scanf("%d", &m);

if (m <= 0) {

printf("Masukkan m harus lebih besar dari 0.\n");

return 1;

}

// Menggunakan malloc untuk mengalokasikan memori awal

primeArray = (int\*)malloc(m \* sizeof(int));

// Pengecekan apakah alokasi memori berhasil

if (primeArray == NULL) {

printf("Gagal melakukan alokasi memori.\n");

return 1;

}

// Memanggil fungsi untuk mengisi nilai bilangan prima

generatePrimes(n, &primeArray, &size);

// Menampilkan bilangan prima

printf("Bilangan prima pertama %d:\n", n);

for (int i = 0; i < size; i++) {

printf("%d ", primeArray[i]);

}

printf("\n");

// Menggunakan free untuk membebaskan memori yang dialokasikan

free(primeArray);

return 0;

}

1. **Penjelasan Program**

Program ini berfungsi untuk menghasilkan bilangan prima pertama sebanyak `n` buah, dengan menggunakan alokasi memori dinamis. Berikut adalah penjelasan langkah demi langkah dari program ini beserta algoritma yang digunakan:

Algoritma yang Digunakan:

1. Algoritma Pengecekan Bilangan Prima:

- Digunakan di dalam fungsi `isPrime`.

- Menggunakan loop dari 2 hingga `sqrt(num)` untuk memeriksa apakah bilangan tersebut memiliki pembagi selain 1 dan dirinya sendiri. Ini adalah algoritma klasik untuk pengecekan bilangan prima.

2. Algoritma Generasi Bilangan Prima:

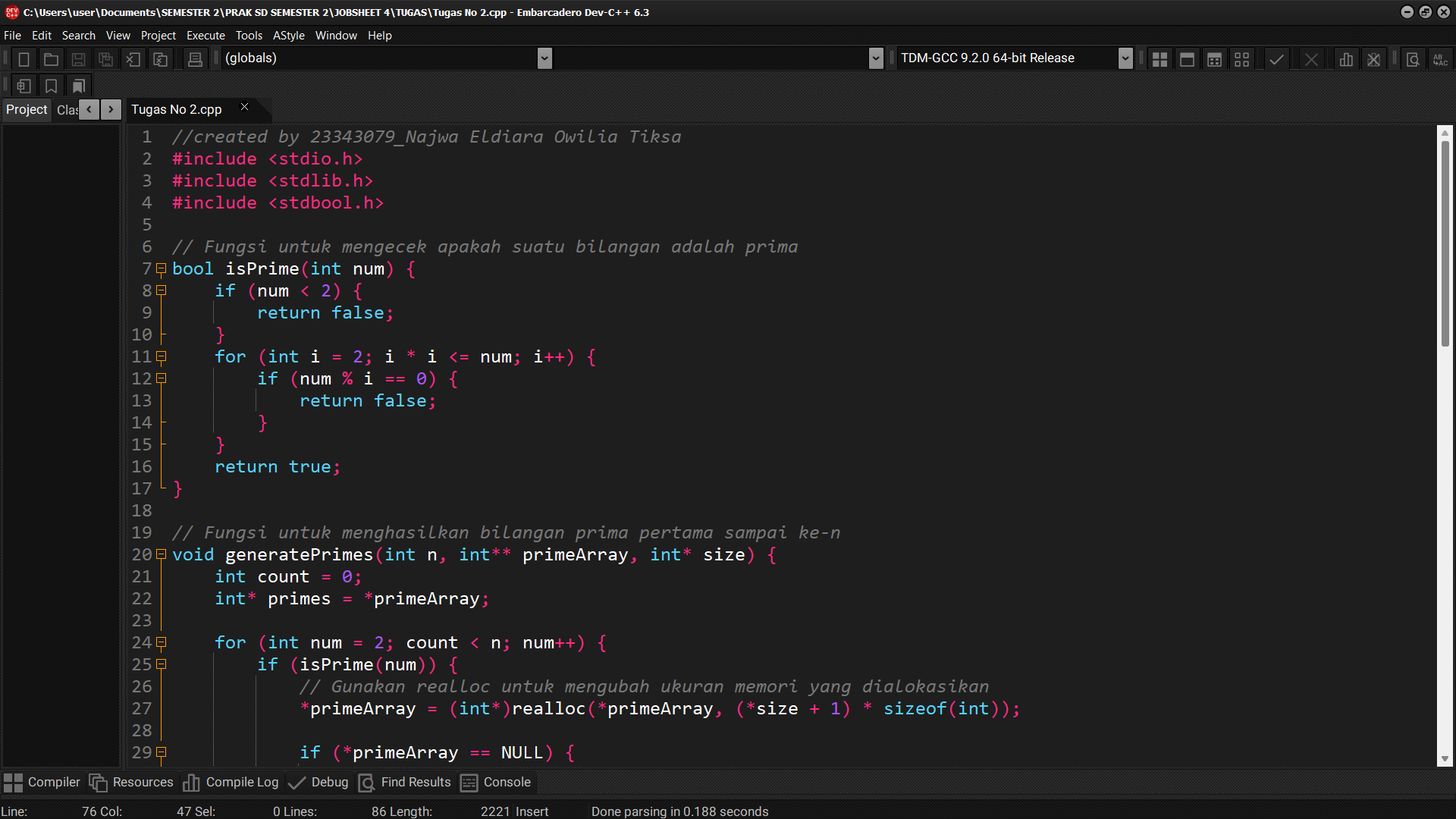
- Digunakan di dalam fungsi `generatePrimes`.

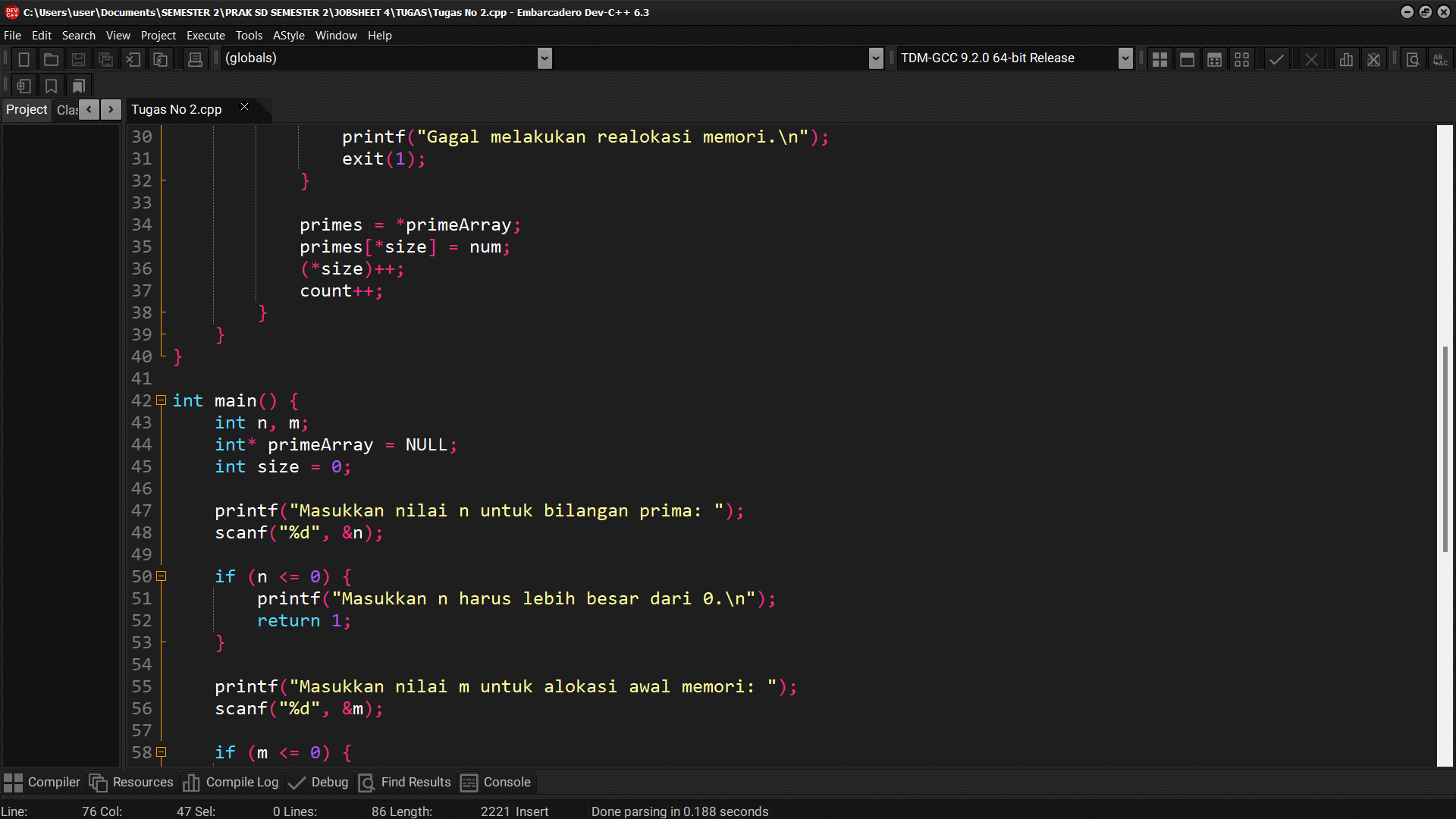
- Menggunakan loop mulai dari 2 dan terus meningkat sampai ditemukan `n` bilangan prima.

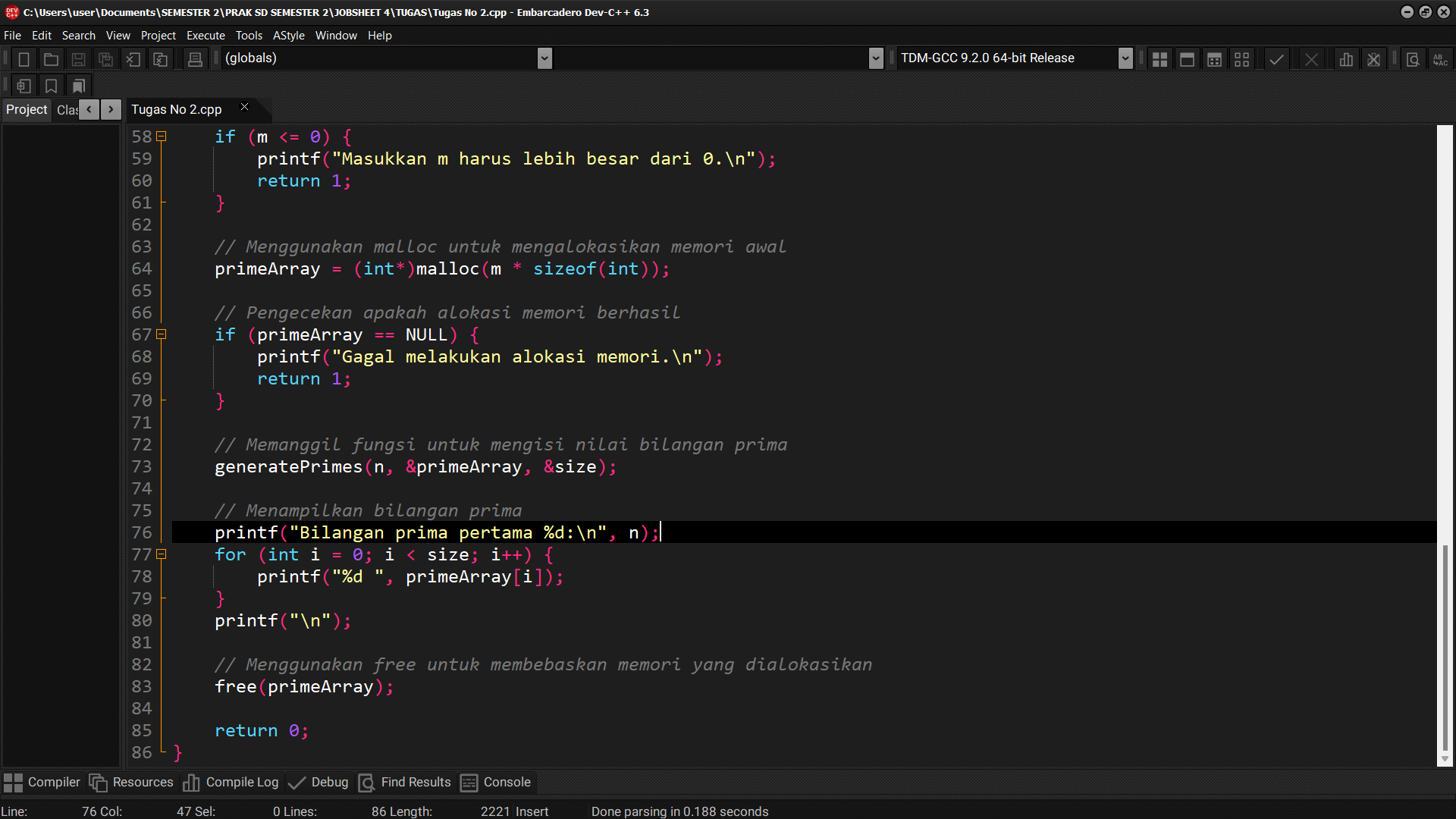
- Setiap kali bilangan prima ditemukan, ukuran array yang menampung bilangan prima diperbesar menggunakan `realloc`.

Program ini menggunakan algoritma dasar pengecekan bilangan prima dan teknik alokasi memori dinamis untuk menghasilkan dan menyimpan bilangan prima.

1. **Screenshot program**







1. **Output**

