A person wearing a plaid shirt

Description automatically generated with low confidence

Percobaan II

Pointer, Structure, Array, dan Operasi dalam Level Bit

**Ahmad Aziz (13220034)**Asisten: Oktavian Putra M (13219039)  
Tanggal Percobaan : 23/09/2022  
EL3111 Praktikum Arsitektur Sistem Komputer  
Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung

*Abstrak*— Pada praktikum modul kedua ini yaitu modul Pointer, Structure, Array, dan Operasi dalam Level Bit akan dilakukan beberapa percobaan dengan topik mengenai tipe data, operator bitwise dalam bahasa C, structure, array, dan juga pointer. Pada praktikum ini ada 13 percobaan yang akan dilakukan. Semua percobaan yang pada praktikum ini akan dilakukan pada computer dengan sistem operasi windows 11, dengan compiler program bahasa C menggunakan GCC serta teks editor yang digunakan adalah Visual Studio Code dan notepad++. Percobaan pertama yang dilakukan adalah fungsi XOR, pada percobaan ini akan digunakan operasi pade level bitwise dalam bahasa C yaitu operator XOR dimana untuk membuat fungsi ini hanya dengan operator AND dan operator NOT. Percobaan kedua adalah membuat fungsi enkstraksi byte, pada percobaan ini akan membuat sebuah fungsi yang mengebalikan nilai byte ke-n pada suatu value. Percobaan senlanjutnya masih berkaitan dengan byte yaitu membuat fungsi masking byte. Fungsi ini menghasilkan mask pada bit diantara rentang atas dan bawah yang dimasukkan pada parameter fungsi. Percobaan kelima adalah fungsi pengurangan dimana percobaan ini masih berkaitan dengan bitwise operator dimana untuk melakukan pengurangan hanya dilakukan dengan operator penjumlahan dan invers saja. Percobaan ke-6 sampai percobaan ke-7 masih berkaitan dengan bitwise operator yaitu membuat fungsi shift register dan enkripsi dalam level bitwise. Dari percobaan ke-8 hingga percobaan terakhir ke-13 dilakukan percobaan mengenai pointer dan juga mengenai array. Tujuan praktikum ini adalah membuat semua fungsi dan kode yang berfungsi sesuai dengan spesifikasi agar praktikan dapat memahami materi pada percobaan dengan baik.

*Kata Kunci*—*Struct*, *pointer*, *array*, bitwise.

# Pendahuluan

P

ada praktikum modul 1 yaitu Compiler Bahasa C dan Bahasa Assembly Intel® x86 dilakukan sebanyak sembilan percobaan yang bertujuan diantaranya sebagai berikut:

* Praktikan memahami representasi informasi dalam level bit yang disimpan pada memory.
* Praktikan mampu menggunakan operator-operator bitwise dalam bahasa C untuk mengolah informasi yang tersimpan dalam memory.
* Praktikan memahami fungsi pointer dan mampu menggunakan pointer untuk melakukan pengolahan data di dalam memory.
* Praktikan memahami array beserta representasinya dalam memory dan pengolahan informasinya dalam bahasa C.
* Praktikan memahami structure beserta representasinya dalam memory dan pengolahannya dalam bahasa C.

Praktikum pada modul ini menggunakan bahasa pemrograman C dengan compiler GCC untuk melakukan percobaan dan juga bahasa assembly sebagai hasil compiling dari bahasa C yang akan dianalisis. Ada beberapa topik yang dibahas pada modul praktikum ini yaitu sebagai berikut:

1. Kompilasi dengan GCC.
2. *Disassembly* dengan GCC.
3. Optimisasi program pada proses kompilasi.
4. Makefile dan Batch file.
5. Instruksi dan Bahasa Assembly Intel® x86.
6. *Stack* dan *procedure call.*

Dalam melakukan percobaan dan analisis pada praktikum modul ini, perangkat lunak dan alat yang digunakan adalah sebagai berikut:

1. Compiler GCC
2. CodeBlocks
3. Code editor Visual Studio Code
4. HexEdit

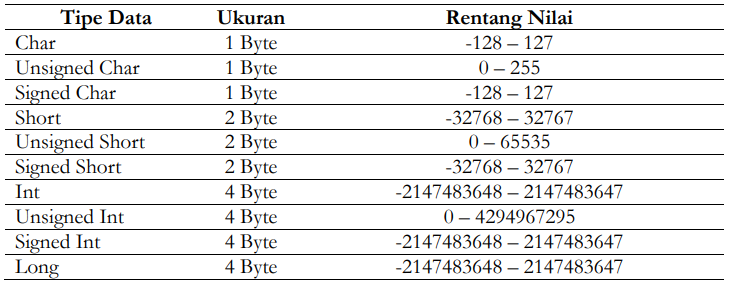
# Landasan Teoretis

Bahasa pemrograman C merupakan bahasa pemrograman yang bersifat processor independent. Artinya bahasa pemrograman C dapat berjalan tanpa bergantung pada jenis ataupun arsitektur processor dimana program tersebut dijalankan. Karena itu, bahasa C dapat dijalankan pada berbagai mesin tanpa mengubah kodenya. Karena flexibilitas dan portabilitasnya inilah bahasa pemrograman C banyak dipakai dalam pemrograman sistem ebeded. Bahasa pemrograman C dapat berjalan pada berbagai mesin karena compiler yang mendukung pada setiap mesin yang menjalankannya. Mesin tidak dapat membaca bahasa C, mesin hanya dapat membaca bahasa mesin (binary). Oleh karena itu, dibutuhkan sebuah proses agar bahasa C dapat dibaca oleh mesin, proses ini disebut proses kompilasi.

## Tipe Data

Tipe data merupakan representasi data yang disimpan dalam memory. Tipe data menentukan operasi yang dapat dilakukan pada suatu data bertipe tertentu, rentang nilai yang mungkin dimiliki oleh data bertipe tertentu, arti dari data, dan cara menyimpan data tersebut dalam memory. Terdapat beberapa tipe data yang telah tersedia dalam bahasa C. Tipe data yang tersedia ini disebut simple data type.

Masing-masing tipe data memiliki ukuran yang berbeda-beda untuk disimpan di dalam memory. Dalam bahasa C, kita dapat menggunakan sintaks sizeof untuk mengetahui besar tipe data tersebut di dalam memory. Contohnya, untuk mengetahui ukuran tipe data integer, kita cukup menggunakan perintah sizeof(int) untuk mengetahui ukurannya. Kita juga dapat melihat rentang nilai yang direpresentasikan oleh masing-masing tipe data.



Table

Description automatically generated

Data pada tabel berikut dapat diketahui dengan program sederhana sebagai berikut. Perhatikan bahwa mesin berbeda dapat memberikan hasil yang berbeda berkaitan dengan besar tipe data. Dengan adanya perbedaan ukuran masing-masing tipe data, diperlukan sebuah mekanisme alignment pada memory agar setiap data tersusun dengan baik di dalam memory dan dapat diproses oleh mikroprosesor. Dengan alignment, data-data variabel disimpan dalam lokasi memory yang memiliki address offset yang berupa kelipatan dari ukuran word. Hal ini akan menambah performance karena data disusun sesuai cara mikroprosesor menggunakan memory.

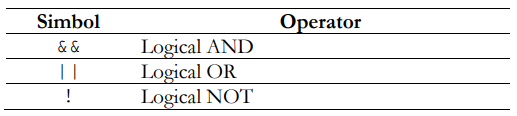
## Operator Bitwise dalam Bahasa C

Bahasa C mendukung pengolahan informasi dalam level bit menggunakan operator bitwise. Berbeda dengan operator level byte, operator bitwise akan mengoperasikan data untuk setiap bit. Sedangkan operator level byte, data akan diolah dalam bentuk 1 byte (1 byte = 8 bit). Operator bitwise dapat digunakan pada berbagai tipe data seperti char, int, short, long, atau unsigned. Operator-operator bitwise dalam bahasa C didefinisikan sebagai berikut.

Table

Description automatically generated

Bahasa C juga memiliki operator logika AND, inclusive OR, dan NOT. Operator ini sering tertukar dengan operator bitwise. Operator logika tersebut diberikan sebagai berikut. Pada operasi logika, setiap argumen bukan nol merepresentasikan TRUE, sedangkan argumen nol merepresentasikan FALSE. Ekspresi logika akan mengembalikan nilai 1 untuk TRUE dan nilai 0 untuk FALSE.



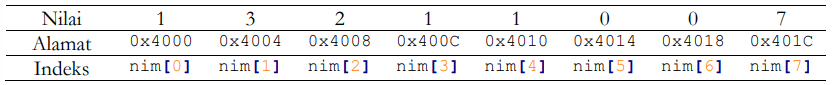
Khusus untuk operator Right Shift, terdapat dua jenis operator Right Shift, yaitu Logical Right Shift dan Arithmetic Right Shift. Logical Right Shift akan mengisi bit MSB dengan nilai 0 sementara Arithmetic Right Shift akan mengisi bit MSB sesuai dengan tanda (sign) variabel tersebut menurut aturan two’s complement. Untuk Left Shift, tidak ada perbedaan antara Logical Left Shift dan Arithmetic Left Shift. Pada umumnya, seluruh mesin dapat mendukung dua jenis operator Right Shift dan Left Shift ini.

## Structure

Structure (struct) merupakan complex data type yang mendefinisikan daftar variabel yang akan ditempatkan dalam blok memory menggunakan satu nama. Dengan demikian, setiap variabel berbeda pada structure dapat diakses menggunakan sebuah single pointer atau dengan menggunakan nama structure itu sendiri. Pada structure, masing-masing variabel disimpan dalam blok memory yang kontigu yang biasanya memiliki delimiter berupa panjang word. Kita juga dapat menggunakan sintaks sizeof untuk memeriksa ukuran structure yang kita definisikan. Perlu diingat bahwa bahasa C tidak mengizinkan kita melakukan deklarasi rekursif terhadap structure (sebuah structure tidak boleh berisi structure bertipe yang sama dengan structure tersebut). Kita dapat menggunakan pointer untuk melakukannya. Beberapa mesin 33 juga membutuhkan alignment data pada memory secara spesifik sehingga ukuran structure dapat berbeda karena compiler secara otomatis melakukan alignment data-data pada structure, contohnya dengan padding.

## Array

Array merupakan kumpulan lokasi penyimpanan data yang kontigu (berurutan) dengan tipe data yang sama. Setiap lokasi penyimpanan dalam sebuah array disebut elemen array. Array dialokasikan secara sekaligus dalam memory sesuai dengan ukurannya. Karena letak elemen yang berurutan, akses elemen array pada memory relatif lebih mudah dan cepat dibandingkan dengan struktur data LinkedList. Setiap elemen dalam array dapat diakses menggunakan indeks yang biasanya berupa bilangan bulat skalar bukan negatif. Dalam bahasa C, elemen pertama dalam array diberi indeks 0. Representasi array dalam memory dengan deklarasi int nim[8] adalah sebagai berikut (asumsikan address elemen ke-0 adalah 0x4000 dengan nilai elemen pertama adalah 1, nilai elemen kedua adalah 3, nilai elemen ketiga adalah 2, nilai elemen keempat adalah 1, nilai elemen kelima adalah 1, nilai elemen keenam adalah 0, nilai elemen ketujuh adalah 0, dan nilai elemen kedelapan adalah 7).



Bahasa C mendukung deklarasi array secara statis maupun secara dinamis. Array statis memiliki ukuran yang tidak bisa diubah-ubah sedangkan array dinamis memiliki ukuran yang dapat ditentukan saat program sedang berjalan. Array dinamis dapat dideklarasikan dengan contoh int pusheen[]. Dengan demikian pusheen merupakan array yang memiliki elemen bertipe integer namun dengan banyak elemen yang belum didefinisikan. Untuk melakukan alokasi terhadap array pusheen, kita dapat menggunakan perintah malloc atau calloc dalam bahasa C. Untuk mengubah ukuran array dinamis yang telah dialokasikan, kita dapat menggunakan perintah realloc. Untuk melakukan dealokasi array dinamis, kita dapat menggunakan perintah free. Perhatikan bahwa pada beberapa kasus, perintah realloc dapat menyebabkan program tidak efisien contohnya saat array diubah ukurannya menjadi lebih besar dan sistem harus melakukan pemindahan elemen-elemen array ke posisi memory yang baru agar perbesaran ukuran array dapat dilakukan.

Array juga dapat memiliki elemen array berupa array. Dengan demikian, kita dapat mendefinisikan array n-dimensi. Contohnya, sebuah matriks merupakan array dengan dua dimensi. Array tersebut memiliki elemen array berupa array, contohnya int pusheen[][5] atau int pusheen[4][5]. Namun, karena memory komputer bersifat linear, komputer akan menyimpan array n-dimensi dalam bentuk linear juga. Hal ini menyebabkan kita harus memperhatikan urutan indeks untuk mengakses setiap elemen array n-dimensi karena hal ini akan berpengaruh terhadap performance dari program yang kita buat terlebih data array yang diolah cukup besar, contohnya seberapa baikkah program yang kita buat dalam memanfaatkan cache memory (cache-friendly).

## Pointer

Pointer merupakan variabel yang menyimpan alamat memory. Dengan kata lain, pointer memiliki nilai alamat memory untuk melakukan referensi terhadap suatu objek yang tersimpan dalam memory komputer. Dengan menggunakan pointer, kita dapat memiliki akses terhadap memory secara langsung. Untuk setiap tipe data T, terdapat pointer ke T. Deklarasi pointer dalam bahasa C dapat dilakukan dengan mudah, contohnya int \*ptr yang merupakan pointer yang melakukan referensi terhadap objek bertipe integer.

Pointer juga dapat digunakan untuk menunjukkan structure berdasarkan alamatnya di memory. Hal ini sangat berguna untuk melakukan passing structure ke atau dari sebuah fungsi hanya dengan memberikan alamat structure tersebut di memory. Pointer ini juga dapat di-dereferensi seperti halnya pointer lain dalam bahasa C, yaitu menggunakan operator dereference (\*). Selain itu, juga terdapat operator yang sangat berguna yaitu struct\_name -> member untuk melakukan dereferensi pointer-to-struct lalu mengakses nilai dari anggota structure tersebut. Operator tersebut memiliki ekuivalensi dengan (\*struct\_name).member. Sebetulnya, sebuah fungsi dapat langsung mengembalikan sebuah structure walaupun hal ini terkadang tidak efisien saat dijalankan.

Dalam array, indeks biasanya didefinisikan sebagai perhitungan matematika terhadap alamat pointer. Dengan demikian penulisan array[i] memiliki ekuivalensi dengan \*(array + i). Perhitungan matematika terhadap pointer untuk mengakses setiap elemen array dapat dilakukan karena array memiliki elemen yang tersusun secara kontigu (berurutan tanpa jeda).

# Hasil dan Analisis

Setelah melakukan percobaan pada semua tugas didapatkan hasil sebagai brerikut:

## Tugas I : Proses Kompilasi Bahasa C Menggunakan GCC

Pada tugas 1 dilakukan proses kompilasi pada program Bahasa C. Kode program bahasa C yang digunakan dan hasil kompilasi pada percobaan ini dapat dilihat pada [lapiran tugas 1](#_Source_code_untuk). Proses kompilasi yang dilakukan secara berurutan mulai dari proses preprocessor, compiler, assembler, dan linker. Semua proses kompilasi pada percobaan 1 ini dilakukan dengan perintah command line satu per satu pada terminal powershell windows dengan GCC.

Proses kompilasi pertama yang dilakukan adalah proses preprosessing. Proses preprocessing yang bertujuan untuk mempersiapkan kode dalam bahasa C untuk dilakukan proses compiling. Pada hasil preprocessing pada kode program code.c terlampir, maka didapatkan hasil dalam bentuk file .i sebagai berikut:

# 0 "code.c"

# 0 "<built-in>"

# 0 "<command-line>"

# 1 "code.c"

# 13 "code.c"

int main(void){

    int indeks;

    int accumulator;

    indeks = 0;

    accumulator = 0;

    while(indeks<500){

        accumulator = accumulator + indeks;

        indeks = indeks + 1;

    }

    return accumulator;

}

Dapat dilihat pada kode hasil preprocessing diatas masih seperti bahasa program C yang dapat dipahami seperti biasa. Namun, terdapat perbedaan pada bagian header baris kode dimana tidak terdapat lagi komentar pada headernya. Hal ini menunjukkan pada proses preprocessing ini terjadi penyederhanaan baris kode program C dimana semua komentar pada baris program dihilangkan sehingga terjadi pengurangan jumlah baris yang sebelumnya berjumlah 23 baris termasuk komentar menjadi hanya 16 baris. Dapat diperhatikan juga pada bagian header program terdapat baris baru yang sebelumnya tidak ada pada program bahasa C yang ditulis.

Setelah dilakukan preprocessing, selanjutnya proses dilanjutkan dengan proses compiling. Proses compiling ini akan memproses output dari proses preprocesesing yang telah dilakukan, Proses compiling menghasilkan kode assembly. Pada percobaan ini masih menggunakan kode program yang sama yaiu code.c , setelah dilakukan compiling menghasilkan kode assembly dalam format file .s seperti yang terdapat pada lampiran code [lapiran tugas 1](#_Source_code_untuk).

Proses selanjutnya adalah assembly. Pada proses assembly, kode akan diterjemahkan ke bahasa assembly. Kode hasil proses assembly dapat di export kedalam bentuk file .s.

Preses terakhir dalam rangkaian proses compiling adalah proses linking. Proses lingking dilakukan untuk mengubah bahasa assembly menjadi bahasa mesin (binary code) yang dapat diexport dalam format .o atau .exe yang dapat dijalankan/dieksekusi oleh mesin.

## Tugas 2 : Proses Kompilasi Bahasa C Menggunakan GCC dengan Bantuan Batch File

Pada percobaan tugas 2 ini akan dilakukan proses compilasi dengan GCC menggunakan batch file. Kode yang digunakan masih sama dengan tugas yaitu code.c yang terdapat pada lampiran. Berikut adalah baris kode dalam batch file yang digunakan:

%~d0

cd "%~dp0"

gcc -E code.c > code.i

gcc -S code.c

gcc -c code.c

gcc -o code.exe code.c

code.exe

pause

Pada batch file tersebut dilakukan berbagai proses complilasi dari preprocessing hingga linking. Batchfile akan mengeksekusi baris-baris perintah dalam batchfile yang sudah kita tulis untuk melakukan proses kompilasi. Hasil dari eksekusi file batch ini manghasilkan file output yang identik/sama persis dengan output hasil kompilasi melalui terinal command line yang ada pada tugas satu.

Dengan menggunakan batch file proses kompilasi yang dilakukan lebih mudah karena dapat dengan sekali eksekusi untuk menjalankan semua perintah kompilasi yang diinginkan.

File output hasil eksekusi dari batchfile ini terdapat pada folder tugas 2.

## Tugas 3 : Disassembly File Objek

GCC selain dapat melakukan assembly program juga dapat melakukan proses disassembly. Proses disassembly adalah proses untuk mengembalikan file program dalam bentuk binary code (.o atau .exe) menjadi kembali ke bentuk kode dalam bahasa assembly.

Pada percobaan ini dilakukan percobaan untuk mendisassembly kode program dalam bentuk binary code yang sudah decompile pada percobaan sebelumnya.

## Tugas 4: Optimisasi Kompilasi Program pada GCC

Proses compilasi yang dilakukan menggunakan GCC dilakukan secara otomatis menerjemahkan kode dari bahasa program C ke bahasa mesin. Proses kompilasi ini dapat kita kustomisasi untuk menghasilkan kode bahasa mesin untuk mendapatkan hasil yang terbaik sesuai dengan kebutuhan/keinginan kita. Proses ini disebut dengan optimisasi program.

Pada perobaan ini dilakukan berbagai jenis optimisasi pada proses kompilasi program bahasa C. Code program bahasa C yang digunakan pada proses optimisasi ini masih sama dengan percobaan sebelumnya. Dilakukan parameterisasi untuk setiap kode program yang dioptimisasi dengan cara tertentu dengan menabahkan jenis optimisasi pada akhir nama program sepert pada folder lab yang ada pada program.

Berdasarkan hasil optimisasi pada semua program untuk setiap jenis optimisasi tersebut didapatkan output yang ketika dijalankan secara fungsionalitasnya identik karena merupakan kode program yang sama.

Berikut adalah perbandingan ukuran file untuk semua output binary pada semua jenis optimisasi:

* O0 : 768 byte
* O1 : 732 byte
* O2 : 896 byte
* O3 : 896 byte
* Os : 896 byte
* Ofast : 896 byte

Dapat diperhatikan dari data hasil optimisasi tersebut file hasil compilasi terkecil adalah dengan optimisasi O1 dilanjutkan O0. Sedangkan jenis optimisasi lainnya menghasilkan ukuran file yang sama.

Hal ini berbanding lurus dengan jumlah baris kode pada bahasa assembly yang dihasilkan. Untuk optimisasi O2 sampai Ofast jumlah baris kode assemblynya sama yaitu 25 baris kode, dan yang terkecil yaitu O1 dengan jumlah baris kode paling sedikit yaitu hanya 18 baris. Sedangkan, pengecualian untuk optimisasi O0 yang tidak berbanding lurus, dimana besaran filenya lebih kecil namun jumlah baris assemblynya lebih besar dari keempat optmisasi yang sama lainnya yaitu sebanyak 30 baris.

## Tugas 5: Kompilasi Beberapa File Kode dengan GCC

Kompilasi beberapa file yaitu mengkompilasi file yang saling terhubung pada rangkaian program C yang biasa disebut dengan file header. File header ini dapat menggabungkan beberapa file program C. Ini biasanya digunakan pada program yang kompleks untuk mempermudah dalam debugging dan pembacaan kode. Penggabungan file ini juga banyak ditemukan dalam library kode dalam program terintegrasi.

Pada percobaan ini dilakukan kompilasi beberapa program yang dihubungkan dengan file header .h. program yang digunakan sebagai main program adalah sebagai berikut:

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 5

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : main\_text.c

// Deskripsi : Demonstrasi MakeFile

// Memanggil prosedur test pada text.c

#include "text.h"

void main(void){

    test();

}

Program main\_text.c ini akan memanggil file headernya dengan #include. Dan pada fungsi main akan memanggil fungsi yang ada dalam fil text.c yang dhibungkan oleh file header tersebut.

Berikut adalah kode yang ada pada file header:

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 5

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : text.h

// Deskripsi : Demonstrasi MakeFile, Mencetak string ke layar

#ifndef TES\_H

    #define TES\_H 100

    void test(void);

#endif

Sedangkan untuk file c nya sendiri adalah sebagai berikut:

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 5

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : text.c

// Deskripsi : Demonstrasi MakeFile, Mencetak string ke layar

#include <stdio.h>

#include "text.h"

void test(void){

    printf("Arsitektur Sistem Komputer sangat menyenangkan!\n");

}

GCC dapat melakukan kompilasi pada semua file tersebut dan menjadikannya satu kesatuan yang nantinya dalam bentuk binary code yang dapat di eksekusi.

Perintah kompilasi untuk melakukan compilasi bersama adalah sebagai berikut:

gcc -o main\_text.exe text.c main\_text.c

Perintah tersebut akan mengkompilasi file dan akan menghasilkan satu binary code yaitu main\_text.exe. Setelah selesai kompilasi dan file binary dieksekusi program yang didapatkan berjalan dengan baik.

## Tugas 6: Penggunaan Makefile pada GCC

Kompilasi dapat dilakukan beberapa perintah secara bersamaan. Hal ini juga dapat dilakukan dengan menggunakan batch file seperti yang dilakukan pada percobaan 2. Pada percobaan ke 5 ini akan dilakukan proses kompilasi beberapa perintah kompilasi dengan menggunakan makefile pada GCC.

Berikut ini adalah perintah kompilasi yang digunakan pada makefile:

all: main\_text.exe

main\_text.exe: main\_text.o text.o

    gcc main\_text.o text.o -o main\_text.exe

main\_text.o: main\_text.c

    gcc -c main\_text.c

text.o: text.c

    gcc -c text.c

Program yang digunakan pada percobaan ini adalah program yang sama dengan program percobaan sebelumnya. Pada percobaan ini setelah dilakukan kompilasi dengan makefile, hasil output binary filenya memiliki ukuran yang sama dan ketika dijalankan juga menghasilkan output dan fungsionalitas yang sama sesuai dengan program. Artinya kedua cara kompilasi ini menghasilkan output yang identik.

Yang membedakan kedua cara ini adalah dengan makefile perintah kompilasi dijalankan satu kali saja ketika kita mengeksekusi makefile tersebut yang hampir mirip dengan batchfile pada percobaan sebelumnya. Sedangkan, dengan menggunakan perintah terminal kita mengetikkan baris perintah dan mengeksekusinya melalaui terminal. Ini menjadi kelebihan pernggunaan makefile untuk meminimalisir terjadinya kesalah serta mempermudah dalam melakukan kompilasi karena kita dapat menyimpan dan mengedit kembali makefile yang sudah dibuat dan mengonfigurasi ulangnya jika dibutuhkan tanpa menulis ulang.

Perbedaan antara makefile dan juga batch file ada pada cara kerjanya. Pada prinsinya batchfile seperti terminal windows yang dibuat dalam bentuk file, dia akan mengeksekusi baris per baris perintah yang ada di dalamnya. Jadi bentuknya akan sama saja dengan terminal windows. Sedangkan makefile merupakan tools eksternal yang dapat menjalankan perintah yang sesuai dengan cara penulisan makefile itu sendiri.

Hasil kompilasi semua cara ini sama karena proses kompilas ini dilakukan pada GCC, makefile, terminal, dan juga batcfile hanya menjalankan perintah GCC untuk menjalankan kompilasi.

## Tugas 7: Header File

Header file merupakan file yang digunakan untuk menghubungkan file program C lain yang akan digunakan dan digabungkan pada suatu program yang sama. Percobaan headerfile ini menggunakan 2 program C yaitu program main dan program fungsi add yang dihubungkan oleh file header .h.

Berikut ini adalah kode fungsi add yang digunakan pada percobaan:

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 7

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : add.c

// Deskripsi : Demonstrasi header file

// Menjumlahkan dua bilangan

#define START\_VAL 0

int accum = START\_VAL;

int sum(int x, int y){

    int t = x + y;

    accum += t;

    return t;

}

Dan berikut ini adalah kode main yang digunakan yang akan memanggil fungsi sum pada kode yang ada di file add.c

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 7

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : main.c

// Deskripsi : Demonstrasi header file

// Main program

#include <stdio.h>

#include "add.h"

int main(){

    int a;

    int b;

    scanf("%d %d", &a, &b);

    int c = sum(a,b);

    printf("Hasil penjumlahan %d + %d = %d",a,b,c);

    return 0;

}

Kedua file ini dihubungkan oleh file header atau main akan menginclude file add.h melalui headernya untuk memanggil file .c nya. Berikut adalah file header dari add

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 7

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : add.h

// Deskripsi : Demonstrasi header file

// Header file fungsi add

#ifndef ADD\_H

    int sum(int x, int y);

#endif

Percobaan ini hampir sama dengan percobaan sebelumya, pada percobaan ini akan dikompilasi dengan perintah berikut melalui terminal:

gcc -o main.exe add.c main.c

Setelah dilakukan eksekusi program pada file output binary didapatkan hasil yang berjalan sesuai dengan program C yang telah dibuat.

## Tugas 8: Pemanggilan Prosedur dan Stack Memory

Pemanggilan prosedur atau fungsi berkaitan dengan register dan memori yang ada pada computer.

## Tugas 9: Program Fibonacci

Berikut ini adalah kode main dari program fibbonaci

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 9

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : fibo\_main.c

// Deskripsi : Main file untuk fibonacci function

// Menghitung deret fibonacci sebanyak input user

#include "inputn.h"

#include "fibo.h"

#include <stdio.h>

int main(){

    int i;

    int N = inputn();

    for(i = 0; i < N; i++){

        printf("%d ", fibonacci(i));

    }

    return 0;

}

Pada program ini memanggil dua file lainnya yang berisi fungsi input dan fibbonacinya sendiri dengan masing masing-masing headerfile sebagai berikut:

Fibo.h

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 9

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : inputn.h

// Deskripsi : Deklarasi fungsi fibonacci

// Header file library fibonacci

int fibonacci(int n);

fibo.c

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 9

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : fibo.h

// Deskripsi : Library fibonacci function

// Menghitung deret fibbonaci

#include "fibo.h"

int fibonacci(int n){

    if(n <= 1){

        return 1;

    }

    else{

        return fibonacci(n-1) + fibonacci(n-2);

    }

}

Kemudian fungsi untuk input yang juga memvalidasi input:

Inputn.h

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 9

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : inputn.h

// Deskripsi : Deklarasi fungsi inputn

// Header file library inputn

int inputn();

dan file c nya:

// Praktikum EL3111 Arsitektur Sistem Komputer

// Modul : 1

// Percobaan : 9

// Tanggal : 23 September 2022

// Kelompok : 10

// Rombongan : B

// Nama (NIM) 1 : Ahmad Aziz (13220034)

// Nama (NIM) 2 : Gilbert Ng (13220032)

// Nama File : inputn.c

// Deskripsi : Fungsi input dan validasi input user

// Fungsi input dan validasi input user

#include "inputn.h"

#include <stdio.h>

int inputn(){

    int n;

    printf("masukkan n: ");

    scanf("%d", &n);

    while (n<2){

        printf("masukkan n: ");

        scanf("%d", &n);

    }

    return n;

}

# Simpulan

* Bahasa C merupakan bahasa yang processor atau OS independent yang dapat berjalan pada platform yang berbeda karena adanya proses ompiling untuk menerjemahkan kode ke platformnya.
* Bahasa program C dapat dikompile dan dikostomisasi jenis kompilasinya dengan optimisasi dan akan mendapatkan hasil yang sesuai dengan kebutuhan dengan jenis optimisasi yang sesuai.
* Kompilasi dapat dilakukan bersamaan untuk banyak perintah dengan menggunakan batfile atau makefile.
* Bahasa C dapat menggabungkan beberapa file yang terpisah dengan tujuan untuk kerapihan kode dan kemudahan pembacaan kode dengan header file.

Referensi

*Basic format for books:*

1. Bryant, Randal, dan David O’Hallaron. *Computer Systems: A Programmer’s Perspective 2nd Edition*. 2011. Massachusetts: Pearson Education Inc.
2. Patterson, David, dan John Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. 2012. Waltham: Elsevier Inc*.*

**Lampiran**

### Source code untuk tugas 1 dan 2

code.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 1  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : code.c  // Deskripsi : Demonstrasi proses kompilasi C  // Menjumlahkan deret bilangan sebanyak N\_LOOP  #define N\_LOOP 500  int main(void){      int indeks;      int accumulator;      indeks = 0;      accumulator = 0;      while(indeks<N\_LOOP){          accumulator = accumulator + indeks;          indeks = indeks + 1;      }      return accumulator;  } |

code.i

|  |
| --- |
| # 0 "code.c"  # 0 "<built-in>"  # 0 "<command-line>"  # 1 "code.c"  # 13 "code.c"  int main(void){      int indeks;      int accumulator;      indeks = 0;      accumulator = 0;      while(indeks<500){          accumulator = accumulator + indeks;          indeks = indeks + 1;      }      return accumulator;  } |

code.s

|  |
| --- |
| .file   "code.c"      .text      .def    \_\_main; .scl    2;  .type   32; .endef      .globl  main      .def    main;   .scl    2;  .type   32; .endef      .seh\_proc   main  main:      pushq   %rbp      .seh\_pushreg    %rbp      movq    %rsp, %rbp      .seh\_setframe   %rbp, 0      subq    $48, %rsp      .seh\_stackalloc 48      .seh\_endprologue      call    \_\_main      movl    $0, -4(%rbp)      movl    $0, -8(%rbp)      jmp .L2  .L3:      movl    -4(%rbp), %eax      addl    %eax, -8(%rbp)      addl    $1, -4(%rbp)  .L2:      cmpl    $499, -4(%rbp)      jle .L3      movl    -8(%rbp), %eax      addq    $48, %rsp      popq    %rbp      ret      .seh\_endproc      .ident  "GCC: (Rev9, Built by MSYS2 project) 11.2.0" |

batch.bat

|  |
| --- |
| %~d0  cd "%~dp0"  gcc -E code.c > code.i  gcc -S code.c  gcc -c code.c  gcc -o code.exe code.c  code.exe  pause |

code.o

A picture containing chart

Description automatically generated

1. Source code untuk tugas 3

disassembly\_code\_o.asm

|  |
| --- |
| code.o:     file format pe-x86-64  Disassembly of section .text:  0000000000000000 <main>:     0: 55                      push   %rbp     1: 48 89 e5                mov    %rsp,%rbp     4: 48 83 ec 30             sub    $0x30,%rsp     8: e8 00 00 00 00          call   d <main+0xd>     d: c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)    14: c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)    1b: eb 0a                   jmp    27 <main+0x27>    1d: 8b 45 fc                mov    -0x4(%rbp),%eax    20: 01 45 f8                add    %eax,-0x8(%rbp)    23: 83 45 fc 01             addl   $0x1,-0x4(%rbp)    27: 81 7d fc f3 01 00 00    cmpl   $0x1f3,-0x4(%rbp)    2e: 7e ed                   jle    1d <main+0x1d>    30: 8b 45 f8                mov    -0x8(%rbp),%eax    33: 48 83 c4 30             add    $0x30,%rsp    37: 5d                      pop    %rbp    38: c3                      ret    39: 90                      nop    3a: 90                      nop    3b: 90                      nop    3c: 90                      nop    3d: 90                      nop    3e: 90                      nop    3f: 90                      nop |

1. Tugas 4

batch.bat

|  |
| --- |
| %~d0  cd "%~dp0"  gcc -O0 -c code\_O0.c  gcc -O1 -c code\_O1.c  gcc -O2 -c code\_O2.c  gcc -O3 -c code\_O3.c  gcc -Os -c code\_Os.c  gcc -Ofast -c code\_Ofast.c  objdump -d code\_O0.o > code\_O0.s  objdump -d code\_O1.o > code\_O1.s  objdump -d code\_O2.o > code\_O2.s  objdump -d code\_O3.o > code\_O3.s  objdump -d code\_Os.o > code\_Os.s  objdump -d code\_Ofast.o > code\_Ofast.s  pause |

code-O0.s

|  |
| --- |
| code\_O0.o:     file format pe-x86-64  Disassembly of section .text:  0000000000000000 <main>:     0: 55                      push   %rbp     1: 48 89 e5                mov    %rsp,%rbp     4: 48 83 ec 30             sub    $0x30,%rsp     8: e8 00 00 00 00          call   d <main+0xd>     d: c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)    14: c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)    1b: eb 0a                   jmp    27 <main+0x27>    1d: 8b 45 fc                mov    -0x4(%rbp),%eax    20: 01 45 f8                add    %eax,-0x8(%rbp)    23: 83 45 fc 01             addl   $0x1,-0x4(%rbp)    27: 81 7d fc f3 01 00 00    cmpl   $0x1f3,-0x4(%rbp)    2e: 7e ed                   jle    1d <main+0x1d>    30: 8b 45 f8                mov    -0x8(%rbp),%eax    33: 48 83 c4 30             add    $0x30,%rsp    37: 5d                      pop    %rbp    38: c3                      ret    39: 90                      nop    3a: 90                      nop    3b: 90                      nop    3c: 90                      nop    3d: 90                      nop    3e: 90                      nop    3f: 90                      nop |

code-O1.s

|  |
| --- |
| code\_O1.o:     file format pe-x86-64  Disassembly of section .text:  0000000000000000 <main>:     0: 48 83 ec 28             sub    $0x28,%rsp     4: e8 00 00 00 00          call   9 <main+0x9>     9: b8 f4 01 00 00          mov    $0x1f4,%eax     e: 83 e8 01                sub    $0x1,%eax    11: 75 fb                   jne    e <main+0xe>    13: b8 4e e7 01 00          mov    $0x1e74e,%eax    18: 48 83 c4 28             add    $0x28,%rsp    1c: c3                      ret    1d: 90                      nop    1e: 90                      nop    1f: 90                      nop |

code-O2.s

|  |
| --- |
| code\_O2.o:     file format pe-x86-64  Disassembly of section .text.startup:  0000000000000000 <main>:     0: 48 83 ec 28             sub    $0x28,%rsp     4: e8 00 00 00 00          call   9 <main+0x9>     9: b8 4e e7 01 00          mov    $0x1e74e,%eax     e: 48 83 c4 28             add    $0x28,%rsp    12: c3                      ret    13: 90                      nop    14: 90                      nop    15: 90                      nop    16: 90                      nop    17: 90                      nop    18: 90                      nop    19: 90                      nop    1a: 90                      nop    1b: 90                      nop    1c: 90                      nop    1d: 90                      nop    1e: 90                      nop    1f: 90                      nop |

code-O3.s

|  |
| --- |
| code\_O3.o:     file format pe-x86-64  Disassembly of section .text.startup:  0000000000000000 <main>:     0: 48 83 ec 28             sub    $0x28,%rsp     4: e8 00 00 00 00          call   9 <main+0x9>     9: b8 4e e7 01 00          mov    $0x1e74e,%eax     e: 48 83 c4 28             add    $0x28,%rsp    12: c3                      ret    13: 90                      nop    14: 90                      nop    15: 90                      nop    16: 90                      nop    17: 90                      nop    18: 90                      nop    19: 90                      nop    1a: 90                      nop    1b: 90                      nop    1c: 90                      nop    1d: 90                      nop    1e: 90                      nop    1f: 90                      nop |

code-Ofast.s

|  |
| --- |
| code\_Ofast.o:     file format pe-x86-64  Disassembly of section .text.startup:  0000000000000000 <main>:     0: 48 83 ec 28             sub    $0x28,%rsp     4: e8 00 00 00 00          call   9 <main+0x9>     9: b8 4e e7 01 00          mov    $0x1e74e,%eax     e: 48 83 c4 28             add    $0x28,%rsp    12: c3                      ret    13: 90                      nop    14: 90                      nop    15: 90                      nop    16: 90                      nop    17: 90                      nop    18: 90                      nop    19: 90                      nop    1a: 90                      nop    1b: 90                      nop    1c: 90                      nop    1d: 90                      nop    1e: 90                      nop    1f: 90                      nop |

code-Os.s

|  |
| --- |
| code\_Os.o:     file format pe-x86-64  Disassembly of section .text.startup:  0000000000000000 <main>:     0: 48 83 ec 28             sub    $0x28,%rsp     4: e8 00 00 00 00          call   9 <main+0x9>     9: b8 4e e7 01 00          mov    $0x1e74e,%eax     e: 48 83 c4 28             add    $0x28,%rsp    12: c3                      ret    13: 90                      nop    14: 90                      nop    15: 90                      nop    16: 90                      nop    17: 90                      nop    18: 90                      nop    19: 90                      nop    1a: 90                      nop    1b: 90                      nop    1c: 90                      nop    1d: 90                      nop    1e: 90                      nop    1f: 90                      nop |

1. Tugas 5

main\_text.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 5  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : main\_text.c  // Deskripsi : Demonstrasi MakeFile  // Memanggil prosedur test pada text.c  #include "text.h"  void main(void){      test();  } |

text.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 5  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : text.c  // Deskripsi : Demonstrasi MakeFile, Mencetak string ke layar  #include <stdio.h>  #include "text.h"  void test(void){      printf("Arsitektur Sistem Komputer sangat menyenangkan!\n");  } |

text.h

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 5  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : text.h  // Deskripsi : Demonstrasi MakeFile, Mencetak string ke layar  #ifndef TES\_H      #define TES\_H 100      void test(void);  #endif |

1. Tugas 6

makefile

|  |
| --- |
| all: main\_text.exe  main\_text.exe: main\_text.o text.o      gcc main\_text.o text.o -o main\_text.exe  main\_text.o: main\_text.c      gcc -c main\_text.c  text.o: text.c      gcc -c text.c |

1. Tugas 7

add.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 7  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : add.c  // Deskripsi : Demonstrasi header file  // Menjumlahkan dua bilangan  #define START\_VAL 0  int accum = START\_VAL;  // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 7  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : add.h  // Deskripsi : Demonstrasi header file  // Header file fungsi add  #ifndef ADD\_H      int sum(int x, int y);  #endif |

add.h

|  |
| --- |
| #ifndef ADD\_H      int sum(int x, int y);  #endif |

main.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 7  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : main.c  // Deskripsi : Demonstrasi header file  // Main program  #include <stdio.h>  #include "add.h"  int main(){      int a;      int b;      scanf("%d %d", &a, &b);      int c = sum(a,b);      printf("Hasil penjumlahan %d + %d = %d",a,b,c);      return 0;  } |

1. Tugas 8

main.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 8  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : main.c  // Deskripsi : Demonstrasi procedure call dan stack  // Menghitung jumlah dari kuadrat bilangan  int square (int x){      return x\*x;  }  int squaresum (int y, int z){      int temp1 = square(y);      int temp2 = square(z);      return temp1+temp2;  }  int main (void){      int a = 5;      int b = 9;      return squaresum(a,b);  } |

1. Tugass 9

fibo\_main.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 9  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : fibo\_main.c  // Deskripsi : Main file untuk fibonacci function  // Menghitung deret fibonacci sebanyak input user  #include "inputn.h"  #include "fibo.h"  #include <stdio.h>  int main(){      int i;      int N = inputn();      for(i = 0; i < N; i++){          printf("%d ", fibonacci(i));      }      return 0;  } |

fibo.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 9  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : fibo.h  // Deskripsi : Library fibonacci function  // Menghitung deret fibbonaci  #include "fibo.h"  int fibonacci(int n){      if(n <= 1){          return 1;      }      else{          return fibonacci(n-1) + fibonacci(n-2);      }  } |

fibo.h

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 9  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : inputn.h  // Deskripsi : Deklarasi fungsi fibonacci  // Header file library fibonacci  int fibonacci(int n); |

inputn.c

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 9  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : inputn.c  // Deskripsi : Fungsi input dan validasi input user  // Fungsi input dan validasi input user  #include "inputn.h"  #include <stdio.h>  int inputn(){      int n;      printf("masukkan n: ");      scanf("%d", &n);      while (n<2){          printf("masukkan n: ");          scanf("%d", &n);      }      return n;  } |

inputn.h

|  |
| --- |
| // Praktikum EL3111 Arsitektur Sistem Komputer  // Modul : 1  // Percobaan : 9  // Tanggal : 23 September 2022  // Kelompok : 10  // Rombongan : B  // Nama (NIM) 1 : Ahmad Aziz (13220034)  // Nama (NIM) 2 : Gilbert Ng (13220032)  // Nama File : inputn.h  // Deskripsi : Deklarasi fungsi inputn  // Header file library inputn  int inputn(); |

makefile

|  |
| --- |
| all:      gcc -o main fibo\_main.c inputn.c fibo.c  clean:      rm main |