MAKALAH PIONTER DALAM BAHASA

PEMROGRAMAN C



Oleh :

Diki Candra

NIM 2022903430010

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEMAWE

2022/2023

DAFTAR ISI

[BAB 1 3](#_Toc117197480)

[PENDAHULUAN 3](#_Toc117197481)

[1.1 Latar Belakang 3](#_Toc117197482)

[1.2 Rumusan masalah 3](#_Toc117197483)

[1.3 Manfaat 3](#_Toc117197484)

[1.4 Tujuan 4](#_Toc117197485)

[BAB II 5](#_Toc117197486)

[PEMBAHASAN 5](#_Toc117197487)

[2.1 PENGENALAN 5](#_Toc117197488)

[2.2 Pengertian Pointer 5](#_Toc117197489)

[2.3 Cara Membuat Pointer 7](#_Toc117197490)

[2.4 Single Pointer 27](#_Toc117197491)

[2.5 Double Pointer 28](#_Toc117197492)

[2.6 Jenis – jenis pointer pada bahasa C. 30](#_Toc117197493)

[1. Null pointer 30](#_Toc117197494)

[2. Void pointer 30](#_Toc117197495)

[3. Wild pointer 32](#_Toc117197496)

[4. Dangling pointer 32](#_Toc117197497)

[5. Complex pointer 36](#_Toc117197498)

[6. Near pointer 36](#_Toc117197499)

[7. Far pointer 36](#_Toc117197500)

[8. Huge pointer 36](#_Toc117197501)

[BAB III 37](#_Toc117197502)

[PENUTUP 37](#_Toc117197503)

[DAFTAR PUSTAKA 38](#_Toc117197504)

# 

# BAB 1

# PENDAHULUAN

## 1.1 Latar Belakang

Apa itu Pointer dalam Bahasa C. Pointer (dalam pemrograman C) adalah suatu variable yang berisikan suatu alamat lokasi tertentu. Yang disimpan oleh variabel yang didefinisikan sebagai pointer adalah ‘alamat’ variabel tersebut. Seperti yang kita ketahui, variabel pointer berfungsi sebagai penunjuk alamat lokasi variabel lain, dan mengganti nilai variable lyang ditunjuk tersebut Pada operasi pointer, kita menggunakan 2 buah operator dalam bahasa C yaitu “\*” dan “&”. Dalam pembuatan program menggunakan pointer, tanda asterisk “\*” digunakan untuk menentukan variable yang menjadi suatu pointer.

## 1.2 Rumusan masalah

Berdasarkan judul dari makalah ini, Saya merumuskan masalah yaitu Pointer dalam bahasa pemrograman bahasa C!

## 1.3 Manfaat

Manfaat yang diperoleh dari makalah tersebut adalah sebagai berikut:

1.**Pembaca**

A. Menjadikan pembaca makalah semata-mata agar pembaca memahami dasar-dasar pemrograman.

B. Untuk memotivasi pembaca untuk membuat makalah yang lebih baik dari Ini.

2.**Pengaramg**

C. Untuk memberikan semangat pada mata kuliah yang sedang diambil saat ini.

D. Sebagai titik awal untuk melakukan pembelajaran agar pembelajaran selanjutnya dapat dilakukan dengan lebih baik lagi.

## 1.4 Tujuan

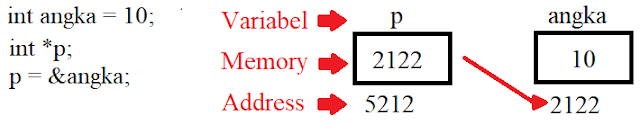
1. Mengetahui Pointer.

2. Dapat menerapkan Bahasa.

# BAB II

# PEMBAHASAN

Pointer adalah tipe data yang nilainya mengacu pada nilai lain yang disimpan di tempat lain di memori komputer menggunakan alamatnya. Pointer dilambangkan dengan simbol ‘\*’. Fungsi utama pointer yaitu memungkinkan suatu program agar melaksanakan kerja lebih efisien dan praktis. Pada intinya pointer mengambil address dari suatu variabel. Address merupakan alamat lokasi suatu memori dari sebuah variabel. Address berupa angka random atau angka acak dari sebuah variabel.



* Pada gambar diatas dideklarasikan variabel angka dengan nilai 10
* Lalu dideklarasikan int \*p;
* P = &angka; berfungsi untuk variabel p mengambil address dari variabel nilai. Sehingga nilai dari variabel p = 10.

## 2.2 Pengertian Pointer

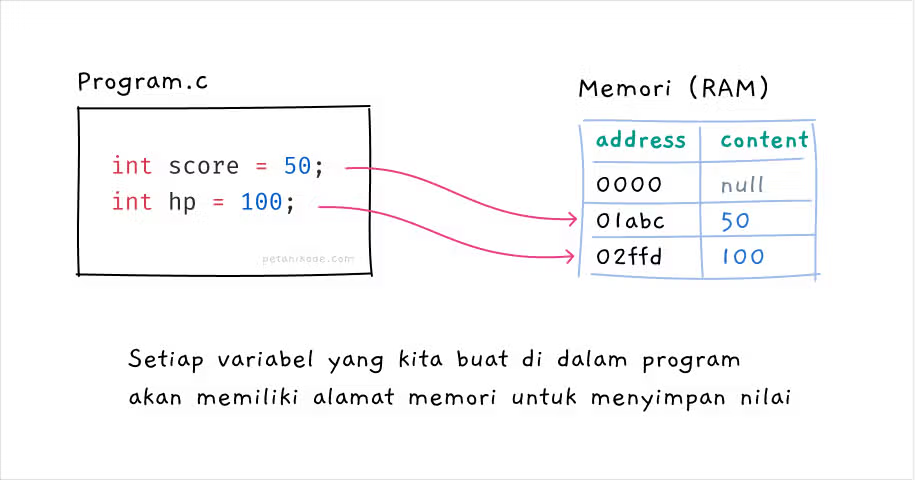
**Apa itu Pointer?**

Setiap variabel yang kita buat pada program akan memiliki alamat memori.

Alamat memori berfungsi untuk menentukan lokasi penyimpanan data pada memori (RAM).

Kadang alamat memori ini disebut reference atau referensi.

Coba perhatikan gambar ini:



Pada gambar ini, kita membuat dua variabel.. yakni score dan hp.

Kedua variabel ini punya alamat memori masing-masing.

Variabel score alamat memorinya adalah 01abc, sedangkan hp alamat memorinya 02ffd.

Selama sebuah alamat masih kosong.. maka alamat itu yang akan dipilih.

Oya, pemilihan alamat memori ini, dilakukan secara acak. Inilah mengapa memori ini di sebut RAM (Random Access Memory).

Intinya, setiap kita membuat variabel pasti akan punya alamat memori.

Kalau tidak percaya, kamu bisa buktikan sendiri dengan menggunakan simbol & (ampersand).

Contoh:

#include <stdio.h>

void main () {

int a;

char b[10];

printf("Alamat memori variabel a: %x\n", &a);

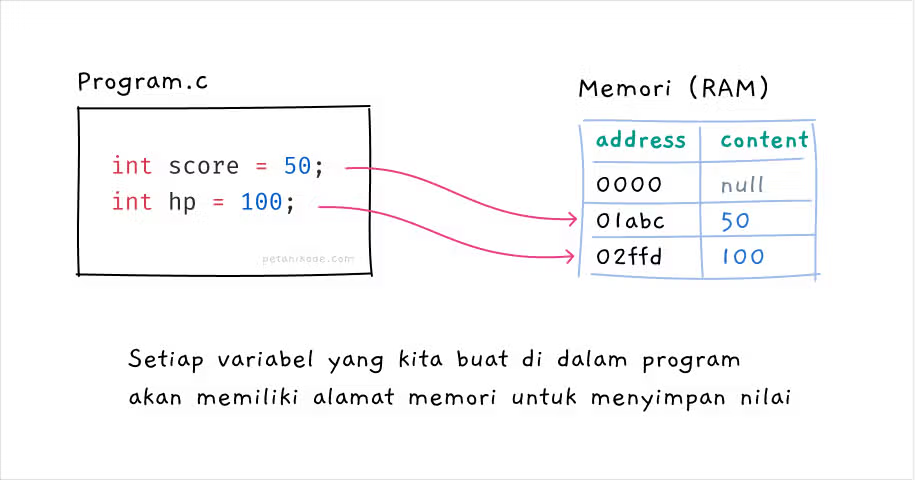
printf("Alamat memori variabel b: %x\n", &b);

}

Pada program ini, kita menggunakan simbol & untuk mengambil alamat memori dari variabel a dan b.

Lalu menggunakan format specifier %x untuk menampilkannya dalam bilangan heksadesimal.

Hasilnya:



Lalu apa hubungannya alamat memori dengan pointer?

Mari kita bahas…

Pointer adalah sebuah variabel khusus yang berisi alamat memori. Pointer nantinya akan bisa mengakses data yang ada di suatu alamat memori.1

Kata kunci yang perlu kamu ingat:

“Pointer berisi alamat memori”

## 2.3 Cara Membuat Pointer

Pointer dibuat dengan menambahkan simbol \* (asterik) di depan namanya, kemudian diisi dengan alamat memori yang akan digunakan sebagai referensi.

Contoh:

int \*pointer1 = 00001;

Maka \*pointer1 akan bisa mengakses data yang ada pada alamat memori 00001. Dengan kata lain, si \*pointer1 akan menggunakan alamat 00001 sebagai referensinya.

Kita juga bisa membuat pointer tanpa harus mengisinya langsung dengan alamat memori.

Contoh:

int \*pointer\_ku;

// atau bisa juga

int \*pointer\_ku = NULL;

Maka \*pointer\_ku akan menggunakan alamat memori 00000, alamat memori ini khusus untuk menyimpan data null atau data kosong.

Sekarang masalahnya:

Karena kita tidak bisa lihat daftar alamat memori secara langsung, kita akan kesulitan memberikan referensi alamat memori untuk pointer.

Belum lagi.. beda komputer beda juga alamat memorinya. Ada yang 8 bit, ada juga yang 16, 32, dan sebagainya.

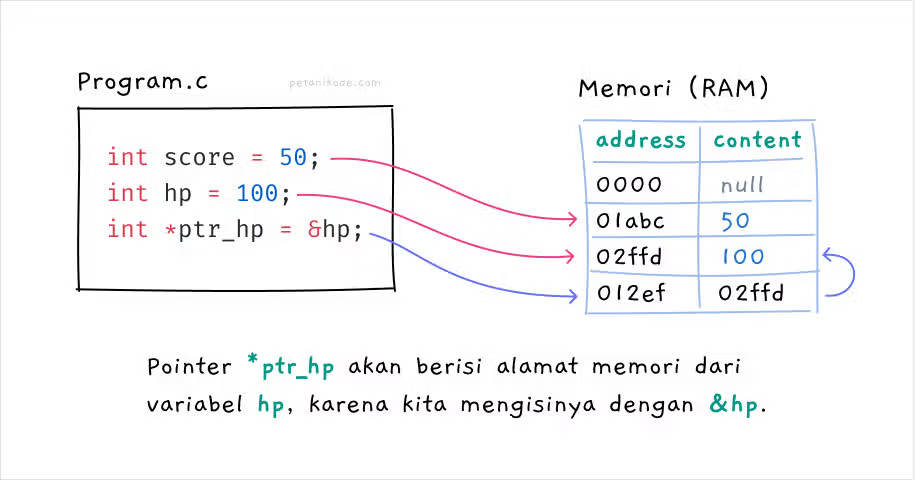
Solusinya:

Kita harus mengambil alamat memori dari variabel yang lain.

Masih ingat caranya?

Ya, dengan menggunakan simbol &.

Coba perhatikan gambar ini:



Pada gambar ini, kita membuat pointer dengan nama \*ptr\_hp\* dengan isi alamat memori dari variabel hp.

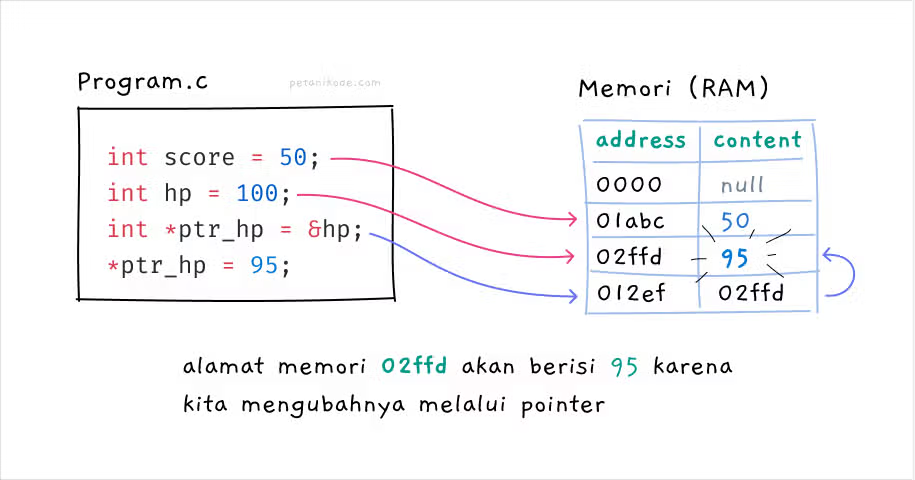
Dengan begini..

Pointer \*ptr\_hp akan bisa mengakses nilai pada alamat memori 02ffd yang mana ini adalah alamat memori dari variabel hp.

Jika kita ingin mengubah nilai pada alamat memori tersebut, maka kita bisa gunakan pointer \*ptr\_hp seperti ini:

\*ptr\_hp = 95;

Maka, sekarang alamat memori 02ffd akan berisi 95 begitu juga dengan variabel hp.



Biar lebih jelas, mari kita coba latihan dalam program.

**Latihan: Mengakses data dengan Pointer**

Buatlah program baru dengan nama contoh\_pointer.c, kemudian isi dengan kode berikut:

#include <stdio.h>

void main(){

int score = 50;

int hp = 100;

// membuat pointer dengan isi alamat memori dari hp

int \*ptr\_hp = &hp;

// print isi variabel dan alama memori

printf("Nama Variabel \t Alamat \t Konten\n");

printf("score \t\t %x \t %d \n", &score, score);

printf("hp \t\t %x \t %d \n", &hp, hp);

printf("ptr\_hp \t %x \t %x \n", &ptr\_hp, ptr\_hp);

printf("\*ptr\_hp \t %x \t %d \n", &ptr\_hp, \*ptr\_hp);

// mengubah data pada alamat memori dengan pointer

\*ptr\_hp = 95;

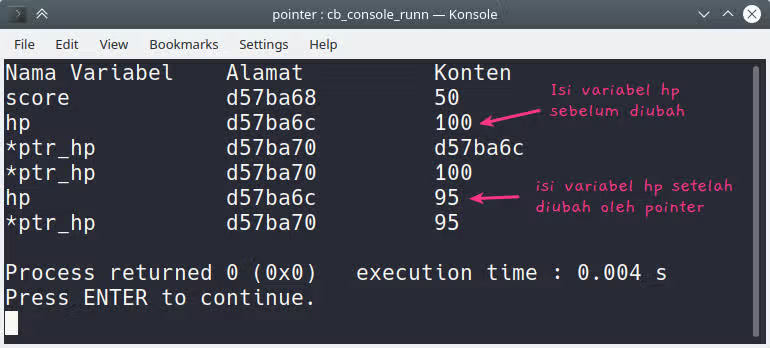
printf("hp \t\t %x \t %d \n", &hp, hp);

printf("\*ptr\_hp \t %x \t %d \n", &ptr\_hp, \*ptr\_hp);

}

Setelah itu, coba compile dan jalankan.

Maka hasil:



Pointer \*ptr\_hp berhasil mengubah nilai pada alamat d57ba6c menjadi 95.

Saat menggunakan pointer, kita menggunakan tanda \* di depan nama pointer untuk mengakses nilai pada alamat memori. Jika tidak menggunakan tanda ini, maka kita akan mendapatkan alamat memori yang di-pointing.

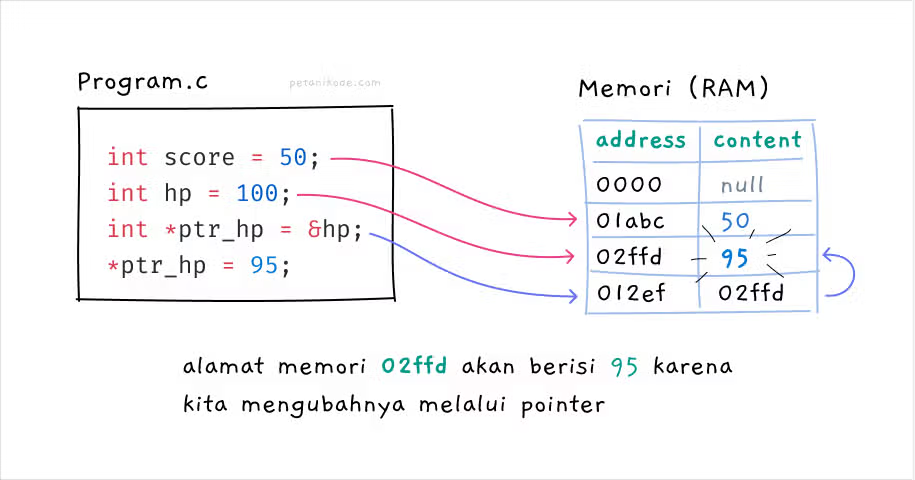
\*ptr // ini akan berisi 95 (nilai dari alamat)

ptr // ini akan berisi d57ba6c (alamat memori dari variabel hp)

Oh iya, pointer juga punya alamat memorinya sendiri.

Pada contoh di atas, alamat memori dari pointer \*ptr\_hp adalah d57ba70. Mungkin di komputermu akan berbeda, silahkan di cek sendiri.

Jika kamu perhatikan gambar ini:



Alamat memori yang dipakai \*ptr\_hp adalah 012ef dengan isi alamat memori 02ffd.

Sapai di sini sudah paham?

Sekarang pertanyaanya:

Kalau kita menggunakan pointer, bukankah ini akan boros memori? Krena kita harus mengalokasikan alamat memori untuk si pointernya juga.

Jika kita bisa menggunakan variabel biasa, ngapain pakai pointer?

Penggunaan pointer sebenarnya opsional, kamu boleh pakai.. boleh juga tidak.

Namun..

Pada kondisi tertentu, penggunaan pointer lebih optimal.

Nanti kita akan bahas ini lagi.

**Kapan Kita Harus Pakai Pointer?**

Seperti yang saya katakan tadi, kita tidak harus selalu menggunakan pointer dalam program.

Namun, ada beberapa kasus tertentu yang menyarankan menggunakan pointer daripada cara biasa.

Biar jelas..

Kita bahas dulu, mengapa pointer diciptakan?

Jadi zaman dulu, memori komputer itu sangat terbatas. Tidak seperti sekarang yang kapasitasnya sampai giga-gigaan.

Saat kita melakukan operasi iterasi pada tipe data seperti array, string, tree, linked list, graph, dan sebagainya.. sering kali memakan banyak memori dan membuat programnya lambat.

Penggunaan Pointer pada operasi semacam akan meningkatkan performa secara signifikan dibandingkan tanpa menggunakan pointer. 1

Latihan: Pointer untuk Pass by Reference pada Fungsi

Petama kita akan coba menggunakan pointer untuk melakukan passing argumen berdasaran referensinya (pass by reference).

Contoh:

#include <stdio.h>

void add\_score(int score){

score = score + 5;

}

void main(){

int score = 0;

printf("score sebelum diubah: %d\n", score);

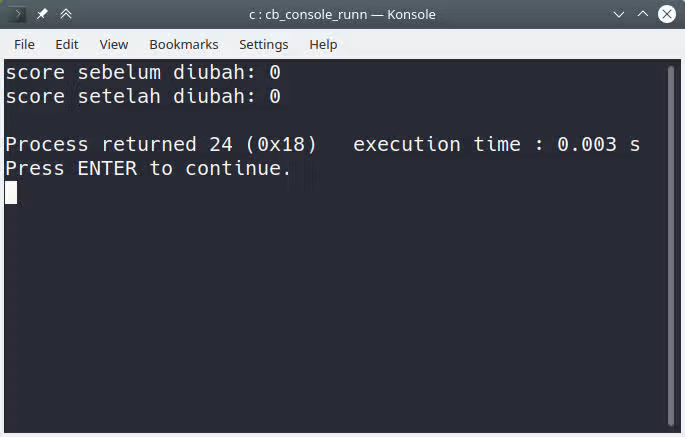
add\_score(score);

printf("score setelah diubah: %d\n", score);

}

Pada program ini, kita membuat fungsi dengan nama add\_score() untuk menambahkan nilai score sebanyak 5.

Tapi ketika dijalankan:



Nilai variabel score tidak berubah, ia tetap bernilai 0.

Mengapa?

Ini karena kita melakukan pass by value, bukan pass by reference.

Variabel score kan dibuat di dalam fungsi main(), lalu ketika fungsi add\_score() mencoba mengubah nilainya..

…maka perubahan hanya terjadi secara lokal di dalam fungsi add\_score() saja.

Nggak percaya?

Coba buktikan dengan mengubah fungsi add\_score() menjadi seperti ini:

#include <stdio.h>

void add\_score(int score){

score = score + 5;

printf("Score dibuah ke %d\n", score);

}

void main(){

int score = 0;

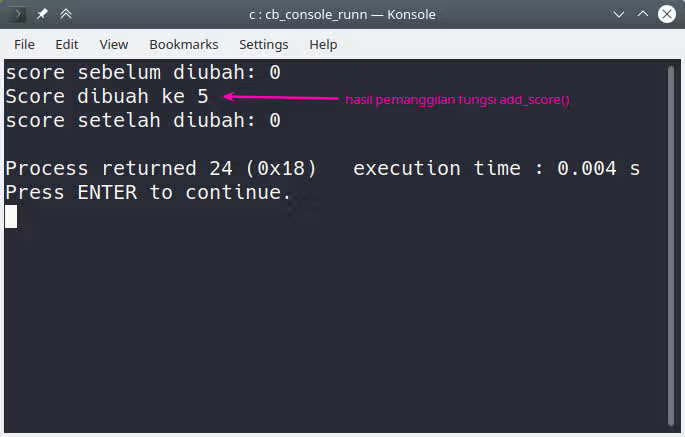
printf("score sebelum diubah: %d\n", score);

add\_score(score);

printf("score setelah diubah: %d\n", score);

}

Hasilnya:



Benar kan yang saya katakan..

Nilai score pada fungsi add\_score() sudah berubah menjadi 5, namun variabel score pada fungsi main() akan tetap bernilai 0.

Nah..

Di sinilah kita harus menggunakan pointer untuk melakukan pass-by-reference.

Sekarang, coba ubah kode programnya menjadi seperti ini:

#include <stdio.h>

void add\_score(int \*score){

\*score = \*score + 5;

printf("score dibuah ke: %d\n", \*score);

}

void main(){

int score = 0;

printf("score sebelum diubah: %d\n", score);

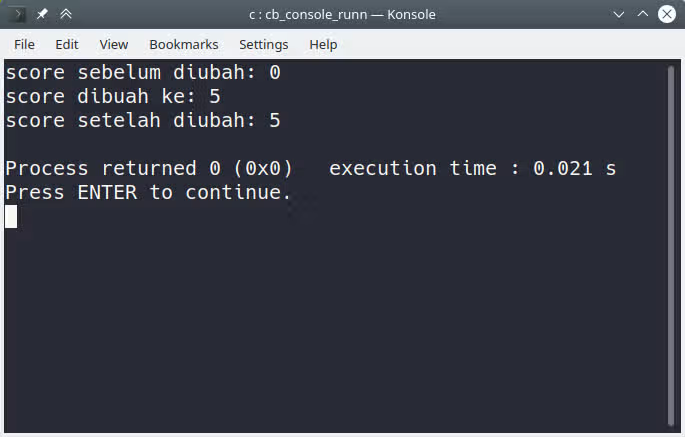
add\_score(&score);

printf("score setelah diubah: %d\n", score);

}

Karena argumen fugsi add\_score() kita ubah menjadi pointer, maka kita harus memberikan alamat memori saat memanggilnya.

Maka hasilnya:



Setiap fungsi add\_score() dipanggil atau dieksekusi, maka nilai variabel score akan bertambah 5.

Coba kita ubah menjadi seperti ini:

#include <stdio.h>

void add\_score(int \*score){

\*score = \*score + 5;

printf("score dibuah ke: %d\n", \*score);

}

void main(){

int score = 0;

printf("score sebelum diubah: %d\n", score);

add\_score(&score);

add\_score(&score);

add\_score(&score);

add\_score(&score);

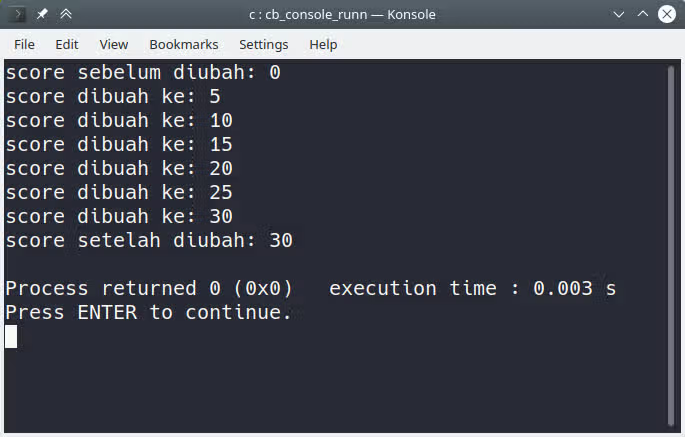
add\_score(&score);

add\_score(&score);

printf("score setelah diubah: %d\n", score);

}

Hasilnya:



**Latihan: Pointer untuk Mengakses Data pada Array**

Pointer juga sering digunakan untuk mengakses data pada array.

Contoh: pointer\_array.c

#include <stdio.h>

void main(){

printf("## Program Antrian CS ##\n");

char no\_antrian[5] = {'A', 'B', 'C', 'D', 'E'};

// menggunakan pointer

char \*ptr\_current = &no\_antrian;

for(int i = 0; i < 5; i++){

printf("📢 Pelanggan dengan no antrian %c silahkan ke loket!\n", \*ptr\_current);

printf("Saat ini CS sedang melayani: %c\n", \*ptr\_current);

printf("-------- Tekan Enter untuk Next --------");

getchar();

ptr\_current++;

}

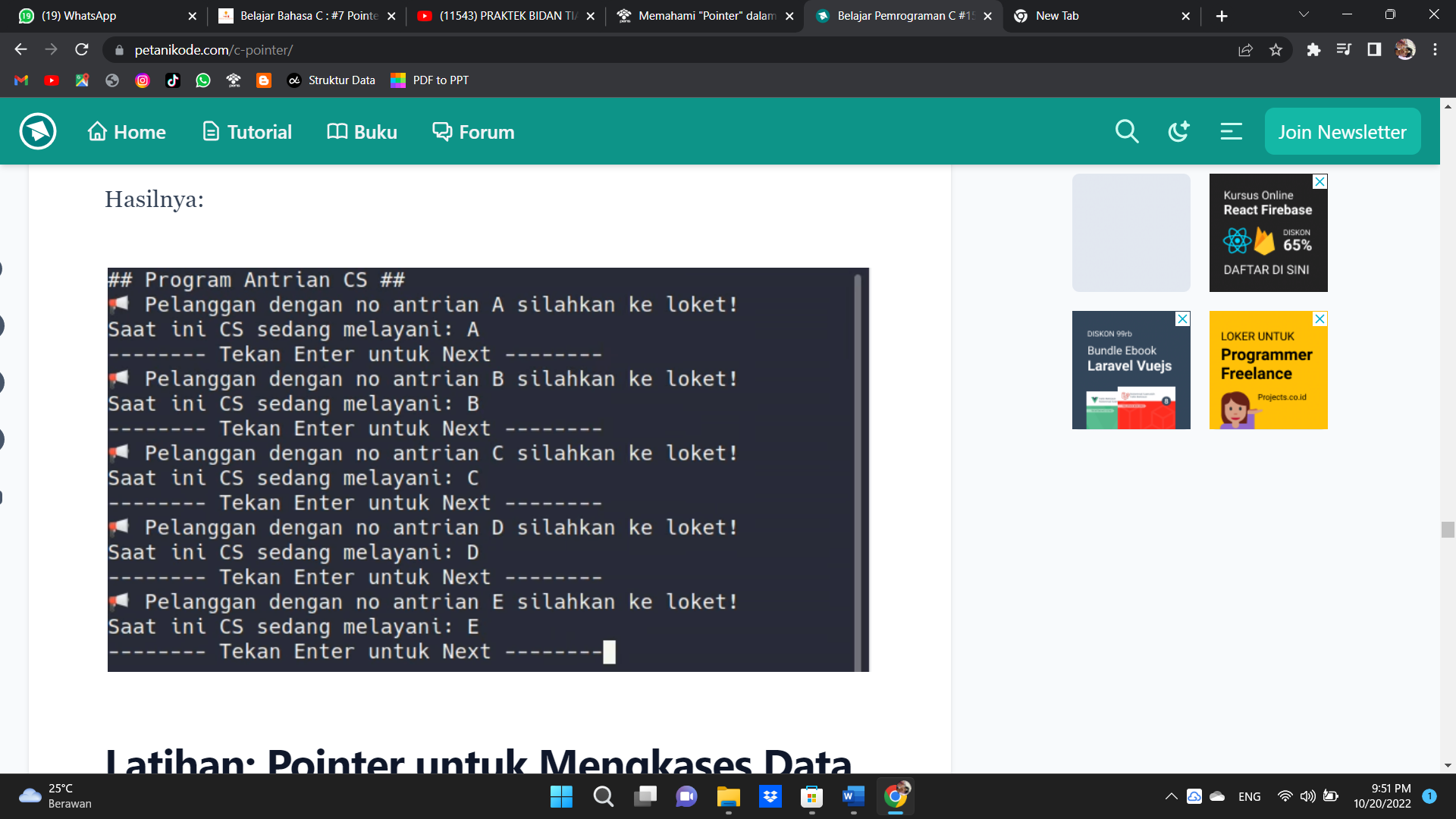
printf("✅ Selesai");

}

Pada program ini, kita menggunakan ptr\_current untuk mengakses elemen array. Saat pertama kali dibuat, pointer ptr\_current akan mereferensi pada elemen pertama array.

Lalu pada perulangan dilakukan increment ptr\_current++, maka pointer ini akan mereferensi ke elemen array sealnjutnya.

Hasilnya:



Latihan: Pointer untuk Mengkases Data pada Struct

Menggunakan pointer pada struct akan membantu kita membuat kode menjadi lebih mudah dan gampang dibaca dibandingkan tanpa pointer.

Sebagai contoh..

Misalnya kita punya struct seperti ini:

struct Player {

char \*name;

int score;

int hp;

struct Weapon \*weapon;

};

struct Weapon {

char \*name;

int attack;

int guard;

};

Struct Player di dalamnya ada struct lagi, yakni struct Weapon. Nah di sini kita menggunakan pointer untuk struct Weapon.

struct Weapon \*weapon;

Pertanyaanya:

Gimana cara mengakses data di struct weapon?

Ada dua cara, kita bisa pakai operator . (dot/titik) dan -> (operator pointer).

Pertama kita coba dulu pakai titik.

Silahkan buat program baru dengan nama pointer\_struct.c, kemudian isi dengan kode berikut:

#include <stdio.h>

void main(){

struct Player {

char \*name;

int score;

int hp;

struct Weapon \*weapon;

};

struct Weapon {

char \*name;

int attack;

int guard;

};

struct Player player1;

player1.name = "Petani Kode";

player1.score = 0;

player1.hp = 100;

(\*player1.weapon).name = "Katana";

(\*player1.weapon).attack = 16;

(\*player1.weapon).guard = 10;

// cetak status player

printf("PLAYER STATUS\n");

printf("Name: %s\n", player1.name);

printf("Score: %d\n", player1.score);

printf("HP: %d\n", player1.hp);

printf("Weapon\n");

printf(" name: %s\n", (\*player1.weapon).name);

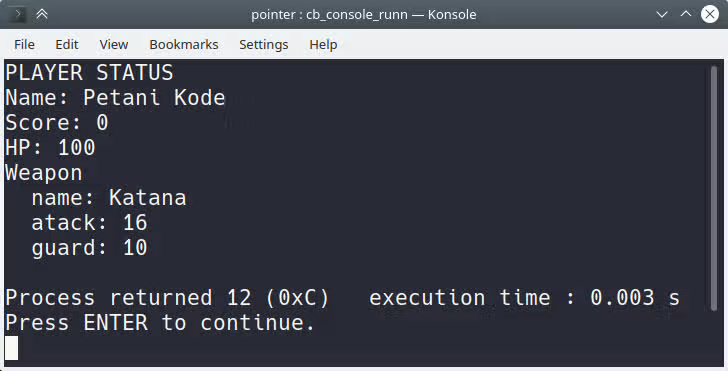
printf(" atack: %d\n", (\*player1.weapon).attack);

printf(" guard: %d\n", (\*player1.weapon).guard);

}

Setelah itu, compile dan jalankan.

Maka hasilnya:



Perhatikan kodenya!

Saat kita mengakses data member pada struct yang bentuknya pointer dengan operator dot atau titik, maka kita harus menggunakan tanda kurung dan \* untuk menyatakan itu adalah pointer.

(\*player1.weapon).name = "Katana";

(\*player1.weapon).attack = 16;

(\*player1.weapon).guard = 10;

Jika hanya satu member yang berbentuk pointer struct, ini oke-oke aja.

Tapi..

Kalau misalnya ada member struct pointer dan di dalamnya lagi ada struct pointer dan di dalamnya ada struct pointer, pasti kita akan bingung.

Bisa saja kodenya seperti ini:

(\*(\*(\*player.weapon).katana).type).name = "Wakizashi";

Duh! saya aja bingung bacanya.

Nah, biar gak seperti ini.. maka sebaiknya pakai Pointer. Sehingga akan bisa seperti ini:

player->weapon->katana->type->name = "Wakizashi";

Ini lebih mudah dibaca dibandingkan yang tadi.

Jadi, jika ada member pointer di dalam Struct, maka sebaiknya pakai operator -> untuk mengakses membernya.

Mari kita ubah kode yang tadi (pointer\_struct.c) menjadi seperti ini:

#include <stdio.h>

void main(){

struct Player {

char \*name;

int score;

int hp;

struct Weapon \*weapon;

};

struct Weapon {

char \*name;

int attack;

int guard;

};

// membuat struct player

struct Player player;

// membuat pointer untuk player

struct Player \*player1;

player1 = &player;

player1->name = "Petani Kode";

player1->score = 0;

player1->hp = 100;

player1->weapon->name = "Katana";

player1->weapon->attack = 16;

player1->weapon->guard = 10;

// cetak status player

printf("PLAYER STATUS\n");

printf("Name: %s\n", player1->name);

printf("Score: %d\n", player1->score);

printf("HP: %d\n", player1->hp);

printf("Weapon\n");

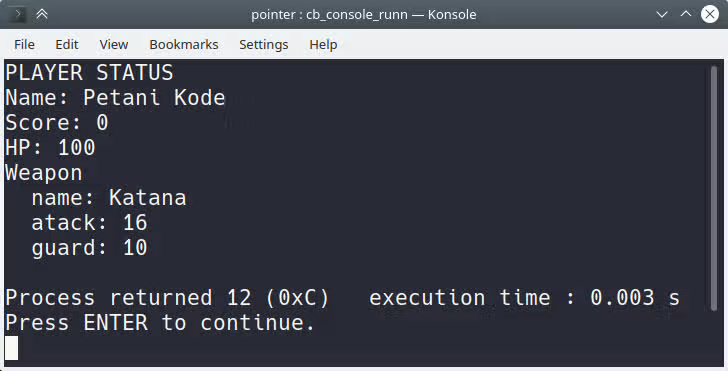
printf(" name: %s\n", player1->weapon->name);

printf(" atack: %d\n", player1->weapon->attack);

printf(" guard: %d\n", player1->weapon->guard);

}

Hasilnya:



Pada program ini kita membuat dua variabel untuk struct Player, yakni player dan player1. Variabel player1 akan menjadi pointer untuk mengakses data pada player.

// membuat struct player

struct Player player;

// membuat pointer untuk player

struct Player \*player1;

player1 = &player;

Selain cara ini, kita juga bisa pakai fungsi malloc() seperti ini:

// alokasi memori untuk player1

struct Player \*player1 = (struct Player\*) malloc(sizeof(struct Player));

// alokasi memori untk player->weapon

player1->weapon = (struct Weapon\*) malloc(sizeof(struct Weapon));

## 2.4 Single Pointer

Single pointer merupakan penggunaan pointer tunggal untuk menunjuk address dari variabel lain.

Contoh program single pointer :

#include <stdio.h>

int main(){

int angka = 10;

int \*p = &angka;

printf("Nilai dari variabel angka : %d\n", angka);

printf("Nilai dari variabel \*p : %d\n", \*p);

printf("Address dari variabel angka : %d\n", &angka);

printf("Address dari variabel \*p : %d\n", p);

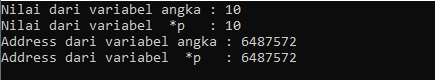
return 0;

}

Penjelasan :

* Variabel angka memiliki nilai 10. Sedangkan pada awalnya variabel p tidak memiliki sebuah nilai.
* Variabel \*p mengambil address dari variabel angka dengan cara \*p = &angka;
* Ketika alamat address telah diambil maka variabel \*p memiliki nilai yang sama dengan variabel angka.

Output



## 2.5 Double Pointer

Double pointer digunakan untuk menunjuk address dari single pointer.

Contoh program double pointer :

#include <stdio.h>

int main(){

int angka = 10;

int \*p = &angka;

printf("Nilai dari variabel angka : %d\n", angka);

printf("Nilai dari variabel \*p : %d\n", \*p);

printf("Address dari variabel angka : %d\n", &angka);

printf("Address dari variabel \*p : %d\n", p);

int \*\*p2 = &p;

printf("Nilai dari variabel \*\*p2 : %d\n", \*\*p2);

printf("Address dari variabel \*p2 : %d\n", \*p2);

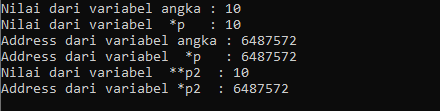
return 0;

}

Penjelasan :

Penggunaan double pointer tidak beda jauh dengan single pointer. Double pointer untuk menunjuk address dari single pointer yang sebelumnya single pointer menunjuk ke sebuah variabel. Sehingga pada akhirnya keduanya memiliki nilai yang sama.

Output



## 2.6 Jenis – jenis pointer pada bahasa C.

### 1. Null pointer

Null pointer berguna ketika kamu tidak menetapkan alamat apa pun ke penunjuk. Pointer null selalu berisi nilai 0.

Contoh program NULL pointer :

#include <stdio.h>

int main(){

int \*ptr;

ptr = NULL;

printf("%d\n", ptr);

return 0;

}

Output



### 2. Void pointer

pointer yang tidak memiliki tipe data terkait dengannya. Sebuah pointer void dapat menyimpan alamat dari jenis apapun dan dapat typecast untuk jenis apapun.

Contoh program void pointer :

#include <stdio.h>

int main(){

float desimal = 7.5;

int angka = 5;

void \*ptr1; // dapat mengakses dari type data manapun

void \*ptr2;

ptr1 = &desimal;

ptr2 = &angka;

printf("Nilai dari \*ptr1 %f\n", \*((float\*) ptr1));

printf("Nilai dari \*ptr2 %d\n", \*((int\*) ptr2));

return 0;

}

Penjelasan :

Void pointer bersifat fleksibel sehingga dia dapat mengakses tipe data apapun. Pada code diatas void pointer mengakses data dari tipe data float dan int.

Output



### 3. Wild pointer

Pointer yang tidak diinisialisasi. Pointer ini dapat diinisialisasi ke nilai sampah non-NULL yang mungkin bukan alamat yang valid.

Contoh program Wild pointer :

#include <stdio.h>

int main(){

int \*ptr;

printf("\n%d",\*ptr);

return 0;

}

Penjelasan :

Wild pointer tidak menghasilkan output karena program tersebut tidak menunjuk ke sebuah variabel dan pointer tersebut tidak diinisialisasi.

### 4. Dangling pointer

Sebuah dangling pointer menunjuk ke variabel non-eksistensial atau lokasi memori. Dangling pointer yang mungkin menunjuk ke variabel atau lokasi memori yang mungkin telah dibebaskan.

Terdapat tiga cara berbeda di mana Pointer disebut dangling pointer.

**a.) Variabel keluar dari ruang lingkup**

#include <stdio.h>

int main(void)

{

int \*ptr;

{

int angka = 17;

ptr = &angka;

}

//variabel angka adalah dangling pointer

printf("angka = %d\n", \*ptr);

return 0;

}

Penjelasan :

* Variabel lokal memiliki masa hidup lokal dan itu milik blok mereka di mana ia dideklarasikan. Setiap kali kontrol keluar ke blok maka memori yang dicadangkan untuk itu akan bebas secara otomatis.
* Pada code diatas int angka =17; dan ptr = &angka merupakan lokal variabel
* Jika variabel lokal dirujuk oleh pointer di luar masa pakainya, maka perilaku tidak akan terdefinisi dan nilai pointer menjadi tak tentu.
* Pada kode di atas, kita mencoba membaca nilai “angka” (variabel bilangan bulat) di luar bloknya (lingkup) melalui “ptr” (penunjuk bilangan bulat), sehingga perilaku penunjuk bilangan bulat (ptr) tidak akan terdefinisi.

Output



**b.) Dealokasi memori**

#include <stdio.h>

#include <stdlib.h>

int main(){

int \*ptr=(int \*)malloc(sizeof(int));

int x=10;

ptr=&x;

free(ptr);

return 0;

}

Penjelasan :

Ketika memori dalam program tidak dialokasikan, maka penunjuk menunjuk ke ruang yang dibebaskan atau dihapus yang akhirnya mengarah ke dangling pointer.

Output

Tidak ada output karena pointer tidak valid.

**c.) Memanggil function**

#include <stdio.h>

int \*function\_angka(){

int angka = 10;

return &angka;

}

int main(){

int \*ptr = function\_angka();

printf("%d\n", \*ptr);

return 0;

}

Penjelasan :

Pointer yang menunjuk ke suatu variabel menjadi dangling pointer ketika pointer yang menunjuk ke variabel lokal bukan merupakan variabel statis.

Output



### 5. Complex pointer

Sebelum mengetahui cara membaca pointer kompleks maka kamu harus terlebih dahulu mengetahui associativity dan prioritas.

Associativity: Operator urutan dengan prioritas yang sama dalam ekspresi digunakan.

Prioritas: Prioritas operator menjelaskan urutan C membaca ekspresi.

Contoh :

int (\*p)(int (\*)[20], int (\*)void))

### 6. Near pointer

Pointer yang hanya dapat menunjuk segmen data 64KB atau nomor segmen 8 dikenal sebagai pointer dekat. Itu hanya dapat mengakses data berukuran kecil sekitar 64 kb dalam jangka waktu tertentu, yang merupakan kelemahan utama dari jenis pointer ini.

### 7. Far pointer

Far Pointer yang dapat mengarahkan atau mengakses seluruh memori tempat tinggal RAM, yaitu, yang dapat mengakses semua 16 segmen dikenal sebagai far pointer. Ukuran far pointer adalah 4 byte atau 32 bit.

### 8. Huge pointer

Huge pointer memiliki ukuran 32-bit yang sama dengan Far pointer, dan juga dapat mengakses bit yang terletak di luar sektor. Far pointer yang diperbaiki dan karenanya bagian dari sektor di mana mereka berada tidak dapat dimodifikasi dengan cara apa pun, sedangkan huge pointer bisa.

# BAB III

# PENUTUP

Demikianlah Makalah ini saya buat dengan sebaik-baiknya. Terima kasih buat Orang tua yang selalu mendukung saya baik materil maupun moril,dan terima kasih untuk abang dan kakak assisten laboratorium paket applikasi, juga terima kasih buat kawan-kawan yang telah membantu dalam mengerjakan makalah ini.semoga makalah ini dapat berguna buat kita semua dan buat angkatan ke depannya kelak.

lebih dan kurangnya saya mohon maaf, kepada abang dan kakak saya ucapkan terima kasih

Wassalammu’alaikum Wr.Wb.

## DAFTAR PUSTAKA

1. <https://www.sobatambisius.com/2021/09/belajar-bahasa-c-7-pointer.html>
2. <https://www.belajarstatistik.com/blog/2021/12/04/pointer-dalam-bahasa-c/>
3. <https://www.petanikode.com/c-pointer/>
4. <https://redaksi.pens.ac.id/2019/11/21/memahami-pointer-dalam-bahasa-c/>
5. <https://cahinfor.com/penjelasan-mengenai-pointer-bahasa-c/>
6. <https://id.wikibooks.org/wiki/Pemrograman_C/Pointer>
7. https://www.mahirkoding.com/pointer-dan-address-dalam-bahasa-c/