# MODUL 3: POINTER, ARRAY

## 1.1. Deskripsi Singkat

Pada praktikum sebelumnya, kita telah mehami dan mempraktikkan penggunaan beberapa tipe data dasar Bahasa Pemrograman C. Jika dilihat dari sudut pandang struktur data, tipe data dasar termasuk ke dalam *Primitive Data Structure*. Pada praktikum kali ini, kita akan mencoba memahami dan mempraktikkan penggunaan salah satu *primitive data structure* lainnya yaitu pointer.

Berikutnya, kita akan mulai mencoba mehami dan mempraktikkan penggunaan *non primitive data structure,* dimulai dari Array. Pada array, kita dapat menampung kumpulan data yang sejenis. Sebagai contoh, jika sebuah array kita deklarasikan bertipe integer, maka semua data yang ada di dalam array tersebut semuanya harus bertipe integer.

## 1.2. Tujuan Praktikum

1) Mahasiswa mampu mendeklarasikan dan menggunakan pointer dalam bahasa C.
2) Menggunakan array, khususnya array multidimensi, dalam penyimpanan data

## 1.3. Material Praktikum

Kegiatan pada modul ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

## 1.4. Kegiatan Praktikum

### A. 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. Untuk melihat alamat memori yang digunakan pada variabel, kita bisa pakai simbol **&** (referensi). Tak perlu jauh-jauh, kita sudah tahu tentang fungsi **`scanf()`**, yaitu fungsi yang meminta input dari pengguna. Contoh:

```
scanf("%d", &a);
```

Pada contoh tersebut, terdapat **'&a'** yang artinya program akan menyimpan nilai input ke dalam alamat memori **a**. Alamat memori memiliki format heksadesimal (hex), yaitu sistem bilangan yang memiliki 16 simbol dalam penomoran. Contoh untuk mencetak alamat dari memori **a**:

```
#include <stdio.h>
int main(){
    int a;
    printf("%p\n", &a);
```

```
}
```

Hasil keluaran (Hasil bisa berbeda setiap saat dan berbeda antara perangkat satu dengan lainnya):

```
0x7ffef6dec9ec
```

Lalu apa hubungannya alamat memori dengan pointer? Pointer adalah sebuah **variabel yang berisi alamat memori** dari variabel yang lain. Setelah mengetahui cara mengambil alamat sebuah variabel, sekarang kita harus menggunakan pointer untuk menyimpannya. Pointer juga bisa mengakses data yang ada di suatu alamat memori.

1. Meskipun berbeda dari variabel umumnya, namun cara mendeklarasikan pointer hampir sama dengan yang lain. Deklarasi pointer:

   ```
   tipe_data *nama_variabel;
   ```

   Karena pointer menampung alamat memori, maka dalam melakukan inisialisasi juga harus memberikan alamat memori. Untuk memahami deklarasi dan inisialisasi pointer, ketik ulang potongan program di bawah ini.

   ```
   #include <stdio.h>
   int main(){
       int a = 5;
       int *p;
       p = &a;
       printf("%p\n", p);
   }
   ```

   Simpan program dengan nama **praktikum3A.c.** Setelah program dijalankan, apakah output yang didapatkan? Pada contoh tersebut, saat memberikan nilai pada variabel pointer **p**, tak perlu menambahkan *****. Kemudian, untuk mencetaknya tak perlu menggunakan &, karena **p** merupakan variabel pointer.

   Operator dereferensi (*****) merupakan kebalikan dari operator referensi (**&**) karena operator ini akan menunjuk nilai yang berada di alamat memori. Modifikasi baris sebelum kurung penutup pada **praktikum3A.c**, menjadi seperti berikut:

   ```
   printf("%d\n", *p);
   ```

   Simpan ulang program, kemudian jalankan. Maka output yang didapatkan akan berbeda dengan sebelumnya. **p** merupakan pointer yang menyimpan alamat dari **a**, sehingga ***p** berisi nilai dari **a**.

2. Agar lebih mengerti mengenai penggunaan pointer, cobalah untuk menjalankan dan memodifikasi contoh yang menggunakan pointer berikut ini. Buatlah file baru dengan nama **praktikum3B.c**, kemudian isi dengan kode berikut:

```
#include <stdio.h>

void main(){
    // membuat variabel
    int umur = 19;
    float tinggi = 175.6;

    // membuat pointer
    int *pointer_umur = &umur;
    int *pointer2 = &umur;
    float *p_tinggi = &tinggi;

    // mecetak alamat memori variabel
    printf("alamat memori variabel 'umur' = %d\n", &umur);
    printf("alamat memori variabel 'tinggi' = %d\n",
&tinggi);
    // mencetak referensi alamat memori pointer
    printf("referensi alamat memori *pointer_umur = %d\n",
pointer_umur);
    printf("referensi alamat memori *pointer2 = %d\n",
pointer2);
    printf("referensi alamat memori *p_tinggi = %d\n",
p_tinggi);
}
```

Setelah itu, jalankan program. Perhatikan hasilnya. Alamat memori yang digunakan sebagai referensi pada pointer akan sama dengan alamat memori dari variabel yang kita gunakan sebagai referensi.

Pointer juga memiliki alamat memorinya sendiri. Sama seperti variabel biasa, jika ingin melihat alamat memori dari pointer, maka kita harus menggunakan simbol **&**. Sebagai contoh, kita masih menggunakan program yang tersimpan pada **praktikum3B.c**. Modifikasi **praktikum3B.c** dengan menambahkan potongan program untuk mencetak alamat pointer-pointer yang ada pada program tersebut. Jalankan kembali **praktikum3B.c** dan pastikan alamat memori dari pointer-pointer yang ada pada program tersebut berhasil ditampilkan pada layar.

3. Kita sudah tahu bahwa pointer menyimpan alamat memori atau ”menunjuk ” sebuah variabel. Pointer juga dapat menyimpan alamat memori atau menunjuk pointer lain (*pointer to pointer*). Jadi, ketika kita mendefinisikan pointer ke pointer, pointer pertama

digunakan untuk menyimpan alamat variabel, dan pointer kedua digunakan untuk menyimpan alamat pointer pertama. Itu sebabnya mereka juga dikenal sebagai pointer ganda. Bagaimana cara mendeklarasikan pointer ke pointer di C? Mari kita memahami dengan bantuan program di bawah ini. Simpan program dengan nama **praktikum3C.c**.

```
#include <stdio.h>
int main()
{
  int var = 789;
  int *ptr2; // pointer for var
  int **ptr1; // double pointer for ptr2
  ptr2 = &var; // storing address of var in ptr2
  ptr1 = &ptr2; // Storing address of ptr2 in ptr1

// Displaying value of var using single and double pointers
printf("Value of var = %d\n", var );
printf("Value of var using single pointer = %d\n", *ptr2 );
printf("Value of var using double pointer = %d\n", **ptr1);

return 0;
}
```

Coba jalankan **praktikum3C.c**. Dari program tersebut dapat kita pahami bahwa mendeklarasikan Pointer ke Pointer mirip dengan mendeklarasikan pointer di C. Perbedaannya adalah kita harus menempatkan tambahan '*' sebelum nama pointer. Berapa banyak level pointer yang dapat kita buat? Bisakah lebih dari **(double)? Cobalah modifikasi program **praktikum3C.c**. untuk mengetahui jawabannya.

4. Berikutnya kita akan mencoba menggunakan pointer untuk melakukan passing argumen berdasaran referensinya (*pass by reference*). Pertama-tama ketik ulang program di bawah ini, lalu simpan dengan nama **praktikum3D.c.**

```
#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. Hal ini karena variabel **score** 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. Hal ini dapat dibuktikan dengan menambahkan potongan program berikut ke dalam fungsi **`add_score()`:**

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

Jalankan ulang **praktikum3D.c** Dapat dilihat bahwa nilai **score** pada fungsi **`add_score()`** sudah berubah menjadi 5, namun variabel **score** pada fungsi **`main()`** akan tetap bernilai 0. Permasalah ini salah satunya dapat diselesaikan dengan menggunakan pointer untuk melakukan *pass-by-reference*.

Sekarang, coba lakukan modifikasi berikut pada kode **praktikum3D.c**:

- Ubah argumen fungsi **`add_score()`** menjadi pointer. Sesuaikan pula operasi-operasi yang menggunakan argumen tersebut.
- Saat pemanggilan fungsi **`add_score()`**, kita harus memberikan alamat memori.

Jalankan ulang **praktikum3D.c**. Sekarang pastikan setiap fungsi **`add_score()`** dipanggil atau dieksekusi, nilai variabel score bertambah 5.

## B. Array

Array merupakan suatu tipe data yang terstruktur dan dapat digunakan untuk menyimpan data yang memiliki tipe data yang sama. Array biasa juga disebut larik. Penggunaan array dapat mengurangi kerumitan dalam proses penyimpanan data dalam jumlah yang besar. Jika kita hendak menyimpan variabel nama yang berjumlah sepuluh nama, kita tidak harus membuat 10 variabel untuk nama tersebut. Dengan menggunakan array kita tidak perlu membuat banyak variabel untuk data yang memiliki tipe yang sama. Selain itu, dalam alokasi memori penyimpanan, tipe data array melakukan pemesanan tempat terlebih dahulu sesuai dengan kebutuhan yang ada.

Terdapat 2 jenis array, yaitu array 1 dimensi dan array 2 dimensi. array dengan 1 dimensi merupakan array yang dapat digambarkan sebagai sebuah baris. Dalam array 1 dimensi, elemen yang ada di dalamnya dapat diakses hanya dengan menggunakan 1 indeks saja. Sedangkan array 2 dimensi merupakan array yang dapat digambarkan seperti sebuah matrik.

Selain itu elemen yang ada dalam array 2 dimensi dapat diakses dengan menggunakan 2 indeks, yaitu indeks kolom dan juga indeks baris. Berikut format pendeklarasian array:

```
tipe_data nama_array[jumlah_elemen];
```

Contoh:

```
int Number[10];
```

Eelemen pada array dapat diinisialisasi sebagai berikut:

```
int Number[10]={1,3,2,4,5,7,6,9,8,0};
```

5. Contoh dibawah ini adalah penggunaan array untuk menyimpan sejumlah angka dan menghitung rata-ratanya. Ketik program pada IDE C Anda, lalu simpan dengan nama **praktikum3E.c.**

```
#include <stdio.h>
int main(void)
{
int numbers[10];
int count = 10;
long sum = 0L;
float average = 0.0f;
printf("\n Masukkanlah 10 Angka:\n");
for(int i = 0; i < count; i ++)
{
printf("%2d> ",i+1);
scanf("%d", &numbers[i]); /* Read a number */
sum += numbers[i]; /* Jumlahkan setiap elemen */
}
average = (float)sum/count; /* Hitung rata-rata */
printf("\n Rata-rata dari sepuluh Angka yang dimasukkan:
%f\n", average);
return 0;
}
```

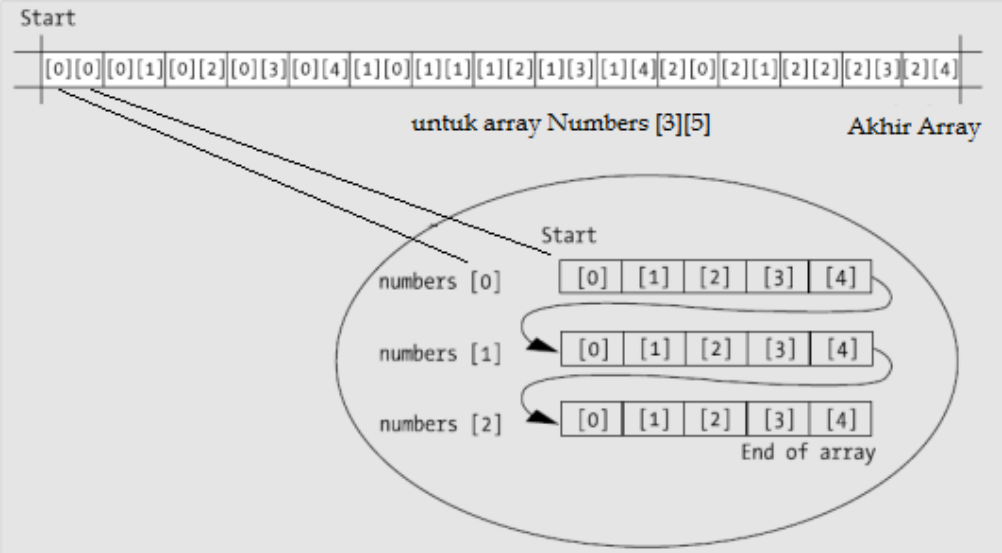
Jika program dijalankan, hasilnya adalah:

```
Masukkanlah 10 Angka:
1> 450
2> 765
```

```
3> 562
4> 700
5> 598
6> 635
7> 501
8> 720
9> 689
10> 527
Rata-rata dari sepuluh Angka yang dimasukkan: 614.700000
```

Pada contoh sebelumnya, array yang digunakan adalah array 1 dimensi, yang dicirikan dengan menggunakan 1 tanda "[ ]". Array multi dimensi akan menggunakan lebih dari satu "[ ]" untu kmenyatakan elemen yang ada pada variabel tersebut. Contoh:

```
float Numbers[3][5];
```




Contoh inisialisasi array multidimensi adalah seperti yang ditampilkan dibawah ini:

```
int numbers[3][4] = {
                      { 10, 20, 30, 40 },
                      { 15, 25, 35, 45 },
                      { 47, 48, 49, 50 }
                    };
```

```
int numbers[2][3][4] = {
                          {
                             { 10, 20, 30, 40 },
                             { 15, 25, 35, 45 },
                             { 47, 48, 49, 50 }
                          },
                          {
                             { 10, 20, 30, 40 },
                             { 15, 25, 35, 45 },
                             { 47, 48, 49, 50 }
                          }
                        };
```

6. Contoh di bawah ini menunjukkan penggunaan array 2 dimensi dalam hal penjumlahan dua matriks.

```
#include <stdio.h>
#define ROW 2
#define COL 3

int main(void)
{
int Matriks_A[ROW][COL]={{4,2,3},{5,7,6}};
int Matriks_B[ROW][COL]={{1,8,9},{3,5,4}};
int Matriks_C[ROW][COL];

printf("Matriks_C adalah : \n");
for(int i = 0; i < ROW; i++)
{
for(int j = 0; j < COL; j++)
{
Matriks_C[i][j] = Matriks_A[i][j]+ Matriks_B[i][j];
}
printf("%d %d %d\n", Matriks_C[i][0], Matriks_C[i][1],
Matriks_C[i][2]);
}
return 0;
}
```

Simpan program tersebut dengan nama **praktikum3F.c.** Jalankan program dan pastikan hasil penjumlahan matriks yang ditampilkan sudah benar. Selanjutnya, simpan ulang program dengan nama **praktikum3G.c.** Lakukan modifikasi pada program **praktikum3G.c** untuk penjumlahan matriks berukuran 3 X 3.

## C. Pointer dan Array

Pointer juga sering digunakan untuk mengakses elemen array, seperti yang ditunjukkan pada program di bawah ini. Ketik ulang program lalu simpan dengan nama **praktikum3H.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. Untuk memahami hal ini, jalankan program **praktikum3H.c**, dan pelajari hasil yang dikeluarkan oleh program.

## 1.5. Penugasan

1. Kerjakan latihan 3 sampai dengan 6 pada akhir bahan tayang (ppt) pertemuan 3.
2. Buatlah contoh program sederhana yang memuat sebuah structure dengan elemen pointer di dalamnya.
3. Buatlah contoh program sederhana yang memuat sebuah pointer yang menunjuk sebuah structure.
4. Buatlah contoh program untuk membuat array dinamis 2 dimensi menggunakan fungsi malloc().

Penugasan dikumpulkan dalam format pdf, yang memuat hasil pengerjaan latihan poin penugasan 1, dan tangkapan layar dari program serta outputnya dari pengerjaan latihan poin penugasan 2, 3, dan 4. Beri nama dengan format **Praktikum3_kelas_noabsen.pdf**. Unggah file zip/rar pada Google Classroom sesuai dengan batas waktu yang telah ditetapkan.