Latihan Soal POINTER

(Soal 1)

```c
int f (void) {
    int s = 1;
    int t = 1;
    int *ps = &s;
    int **pps = &ps; // atau: int **pps; pps = &ps;
    int *pt = &t;

    **pps = 2;
    pt = ps;
    *pt = 3;
    t = s;
}
```

Penjelasan Rinci

1. **Inisialisasi Variabel:**
   - `s = 1` dan `t = 1`.
   - `ps` adalah **pointer** yang menunjuk ke `s`.
   - `pps` adalah **pointer** ke **pointer**, dan diset agar menunjuk ke `ps`.
   - `pt` adalah **pointer** yang awalnya menunjuk ke `t`.

2. **Eksekusi Statement `**pps = 2;`.**
   - `pps` menunjuk ke `ps`, sehingga `*pps` adalah `ps`.
   - Dengan `**pps` berarti mengakses nilai yang ditunjuk oleh `ps`, yaitu `s`.
   - Maka, pernyataan ini mengubah nilai `s` dari **1** menjadi **2**.

3. **Eksekusi Statement `pt = ps;`.**
   - Sekarang `pt` dialihkan agar menunjuk ke alamat yang sama dengan yang ditunjuk oleh `ps` (menunjuk ke `s`).
   - Jadi, setelah ini, `pt` tidak lagi menunjuk ke `t`, melainkan ke `s`.

4. **Eksekusi Statement `*pt = 3;`.**
   - Karena `pt` sekarang menunjuk ke `s`, statement ini mengubah nilai yang ditunjuk oleh `pt` (nilai `s`) menjadi **3**.
   - Nilai `s` dari sebelumnya yang sudah diubah menjadi **2**, sekarang menjadi **3**.

5. **Eksekusi Statement `t = s;`.**
   - Menetapkan nilai `t` sama dengan nilai `s`.
   - Karena `s` kini bernilai **3**, maka `t` juga di-set menjadi **3**.

Latihan Soal POINTER

(Soal 2)

```c
int *value(void)
{
    int i = 3;
    return &i;
}
void callme(void)
{
    int x = 35;
}

int main(void)
{
    int *ip;
    ip = value();
    printf("*ip == %d\n", *ip);
    callme();
    printf("*ip == %d\n", *ip);
}
```

Penjelasan Rinci

Hal yang terjadi pada kode diatas (pada bagian fungsi `main()`) adalah sebagai berikut:

1. Sebuah variabel dengan nama `ip` dengan tipe data pointer of integer dibuat
2. Variabel `ip` diisi dengan hasil kembalian suatu fungsi `value()`.
3. Didalam fungsi `value()` diinisiasikan sebuah variabel baru, yaitu `i` dengan nilai **3**, kemudian fungsi akan mengembalikan alamat memori dari variabel `i`.
4. Proses print informasi sebuah nilai yang ditunjuk oleh `ip` (yaitu `i`) tidak dapat dilakukan, karena setelah keluar dari fungsi `value()`, alamat memori untuk variabel `i` dihapus, sehingga `ip` menunjuk pada alamat memori yang tidak valid (tidak ada nilai yang bisa diambil/mengakses memori sampah).
5. Kemudian fungsi `callme()` dipanggil, di dalam fungsi tersebut di inisiasikan variabel baru, yaitu `x` dengan nilai **35** (pada bagian ini, ada kemungkinan memori yang tadinya digunakan oleh variabel `i` ditimpa oleh variabel `x`, sehingga ada kemungkinan nilai yang ditunjuk oleh variabel `ip` juga menjadi **35**)
6. Proses print informasi sebuah nilai yang ditunjuk oleh `ip` (alamat yang tidak valid) tidak dapat dilakukan, karena `ip` menunjuk pada alamat memori yang tidak valid (ada kemungkinan mengambil nilai 35/mengakses memori sampah).

Pointer `ip` menjadi **dangling** karena menunjuk ke alamat variabel lokal yang sudah keluar dari *scope*. ***Undefined Behavior*** terjadi, sehingga hasil cetak dari `*ip` tidak dapat dipastikan nilainya.

Latihan Soal POINTER

(Soal 3)

```c
int main()
{
    // misal, alamat array blocks adalah 4434
    char blocks[3] = {'I', 'T', 'B'};
    char *ptr = &blocks[0];
    char temp;

    temp = blocks[0];
    temp = *(blocks + 2);
    temp = *(ptr + 1);
    temp = *ptr;

    ptr = blocks + 1;
    temp = *ptr;
    temp = *(ptr + 1);

    ptr = blocks;
    temp = *++ptr;
    temp = ++*ptr;
    temp = *ptr++;
    temp = *ptr;

    return 0;
}
```

Penjelasan Rinci

1. **Inisialisasi Variabel:**
   - `blocks` adalah array dengan elemen: `blocks[0] = 'I'`, `blocks[1] = 'T'`, `blocks[2] = 'B'`.
   - Pointer `ptr` awalnya menunjuk ke alamat memori `blocks[0]`.

2. **Analisis Setiap Baris**

   Baris 1: `temp = blocks[0];`
   - Mengambil elemen pertama dari array.
   - **temp = 'I'**

   Baris 2: `temp = *(blocks + 2);`
   - Mengakses elemen di indeks 2 (karakter ketiga).
   - **temp = 'B'**

Baris 3: `temp = *(ptr + 1);`

- `ptr` masih menunjuk ke `blocks[0]`, jadi `ptr + 1` menunjuk ke alamat memori `blocks[1]`.
- **temp = 'T'**

Baris 4: `temp = *ptr;`

- Mengambil nilai yang ditunjuk oleh `ptr` (masih `blocks[0]`).
- **temp = 'I'**

Perubahan Pointer: `ptr = blocks + 1;`

- Sekarang `ptr` menunjuk ke `blocks[1]`.

Baris 5: `temp = *ptr;`

- Dengan `ptr` menunjuk ke `blocks[1]`, maka
- **temp = 'T'**

Baris 6: `temp = *(ptr + 1);`

- `ptr + 1` menunjuk ke `blocks[2]`.
- **temp = 'B'**

Reset Pointer: `ptr = blocks;`

- `ptr` kembali menunjuk ke `ptr`.

Baris 7: `temp = *++ptr;`

- Operator increment `*++ptr` terlebih dahulu meningkatkan `ptr` sehingga sekarang menunjuk ke `blocks[1]`,
- Kemudian dereference: **temp = 'T'**

Baris 8: `temp = ++*ptr;`

- Di sini, `*ptr` adalah karakter yang saat ini ditunjuk (di `blocks[1]`), yaitu 'T'.
- Operator increment `++*ptr` menaikkan nilai karakter 'T' ke karakter berikutnya dalam tabel ASCII, yaitu 'U'.
- Jadi, **temp = 'U'** (dan blocks[1] berubah menjadi 'U').

Baris 9: `temp = *ptr++;`

- Operator post-increment berarti nilai di `*ptr` (saat ini 'U') diambil dulu, kemudian `ptr` ditingkatkan sehingga menunjuk ke alamat memori dari elemen berikutnya (`blocks[2]`).
- **temp = 'U'**
- Setelah ini, `ptr` menunjuk ke `blocks[2]`.

Baris 10: `temp = *ptr;`

- Mengakses nilai yang ditunjuk oleh `ptr` yang sekarang menunjuk ke alamat memori `blocks[2]`.
- **temp = 'B'**