TIF1101 – Dasar-Dasar Pemrograman

HO 09 - Pointer

Opim Salim Sitompul

Department of Information Technology
Universitas Sumatera Utara

Outline

1. Pendahuluan
2. Mendeklarasikan Variabel Pointer
3. Inisialisasi Variabel Pointer
4. Mengakses Nilai yang Ditunjuk oleh Pointer
5. Mengganti Nilai yang Ditunjuk oleh Pointer
6. Operator Indireksi dan Alamat
7. Alokasi Memori Dinamik

Pendahuluan

- Konsep pointer merupakan konsep tentang variabel yang paling penting dalam C.
- Berbeda dari konsep variabel biasa yang merupakan objek data tempat menyimpan suatu nilai di dalam memori, pointer adalah sebuah objek data berisi alamat yang menunjukkan lokasi memori di mana suatu nilai data tersimpan.
- Melalui konsep pointer, C memberikan cara yang sangat efisien dalam mengakses dan mengubah data.

Pendahuluan

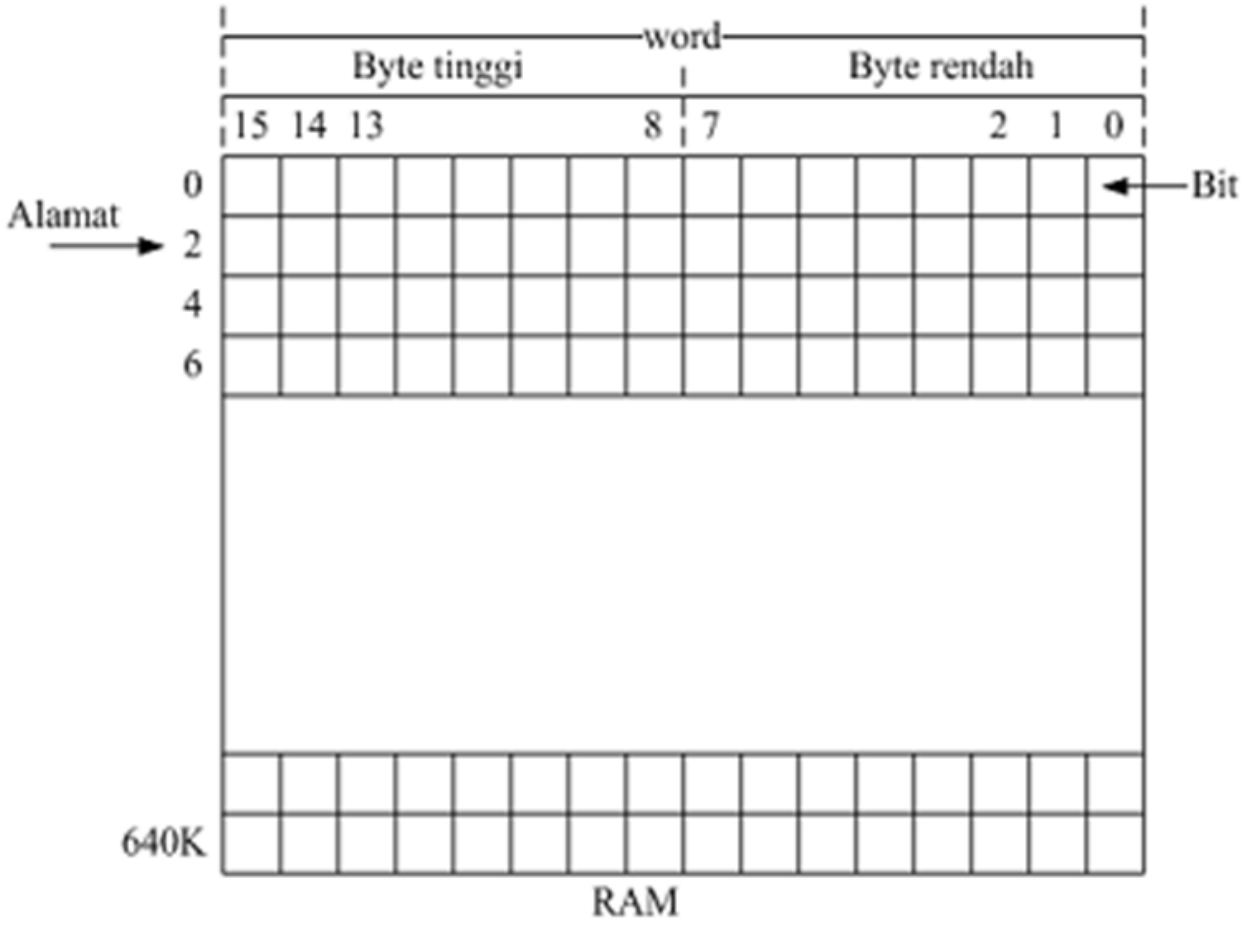
- Alamat variabel dalam C
  - Jika sebuah variabel *x* dideklarasikan di dalam sebuah program, &*x* akan memberikan alamatnya dalam memori.
  - Alamat sudah sangat sering digunakan ketika menggunakan fungsi *scanf()*.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int x = 5;
6    printf("x = %d\n", x);
7    // Perhatikan penggunaan & sebelum x
8    printf("address of x: %p", &x);
9
10   return 0;
11 }
```

Pendahuluan

- Setiap lokasi memori komputer (*Random Access Memory*, disingkat RAM) memiliki sebuah alamat tunggal yang dimulai dari 0 dan bertambah secara sekuensial hingga maksimum memori yang tersedia.
- Sebagai ilustrasi, perhatikan susunan alamat memori untuk komputer yang memiliki memori sebesar 640K seperti pada Gambar 1.

Pendahuluan

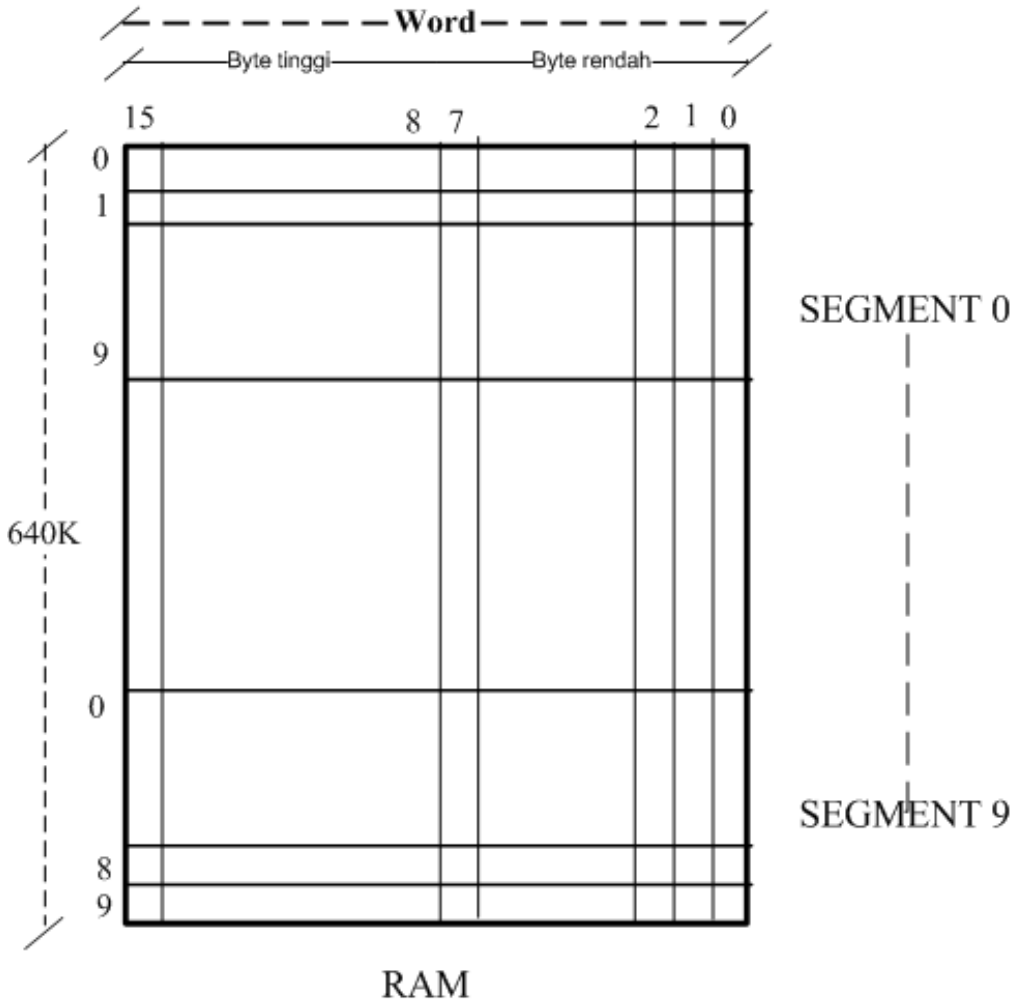


Gambar 1: Alamat Memori Komputer

Pendahuluan

- Akan tetapi dalam implementasinya alamat memori tersebut dibagi atas SEGMENT dan OFFSET.
- Jadi jika besar memori ada 640K kemudian dibagi atas 10 Segment, maka lokasi memori tersebut dapat diilustrasikan seperti pada Gambar 2.

Pendahuluan



Gambar 2: Alamat SEGMENT:OFFSET Memori Komputer

Pendahuluan

- Karena pointer menunjuk ke alamat memori, kompiler dapat secara langsung mencari data yang diinginkan pada lokasi memori di mana data tersimpan.
- Alamat lokasi penyimpanan data diketahui oleh kompiler pada saat variabel dideklarasi.
- Dengan menggunakan alamat tunggal ini, komputer dapat dengan mudah menelusuri memori komputer.

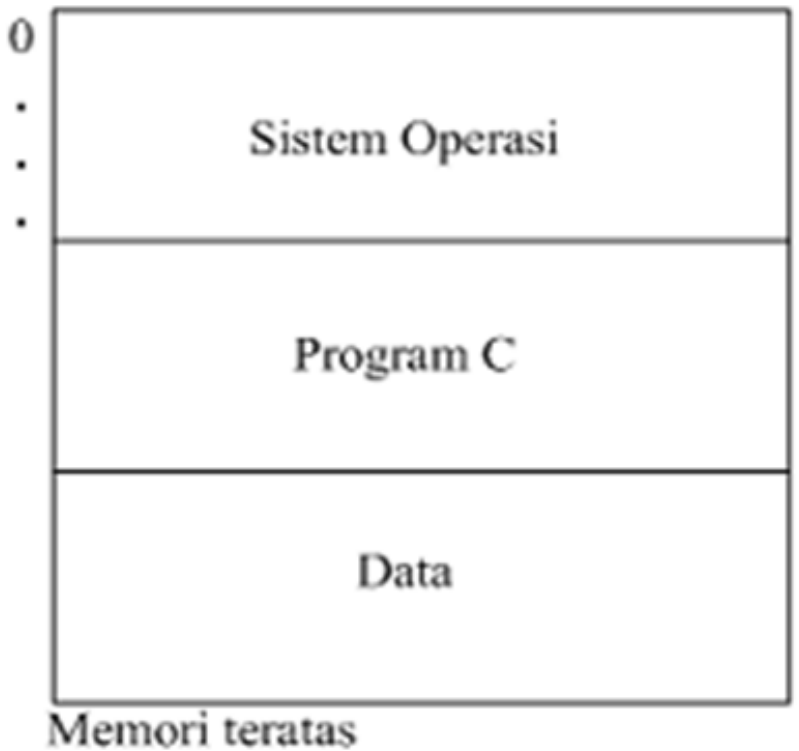
Pendahuluan

- Apabila kompiler akan menyimpan hasil perhitungan di dalam memori, kompiler mencari alamat lokasi variabel tempat hasil perhitungan itu disimpan, dan menyimpannya pada alamat tersebut.
- Program C, data, dan sistem operasi berbagi memori yang sama.
- Besarnya memori yang ditempati oleh program dan data tergantung dari besarnya kode program yang diperoleh setelah dikompilasi, dan data yang digunakan di dalam program.

Pendahuluan

- Sedangkan besarnya memori yang ditempati oleh sistem operasi tergantung dari sistem operasi yang digunakan.
- Semakin tinggi versi sistem operasi semakin besar memori yang digunakan karena semakin besar kemampuan yang disediakannya.
- Susunan penempatan program, data, dan sistem operasi di memori komputer dapat dilihat pada Gambar 12.
- Sistem operasi menempati blok memori pertama, program C menempati blok kedua, dan terakhir blok untuk data.

Pendahuluan



Gambar 3: Alokasi Memori

Mendeklarasikan Variabel Pointer

- Bentuk umum deklarasi variabel pointer adalah:
  jenis_data $*$*nama_variabel*;
- Jenis_data adalah jenis data yang disimpan pada alamat yang ditunjuk oleh nama_variabel.
- Contoh:
  int $*$*intptr*1, $*$*intptr*2;
  float $*$*flptr*;

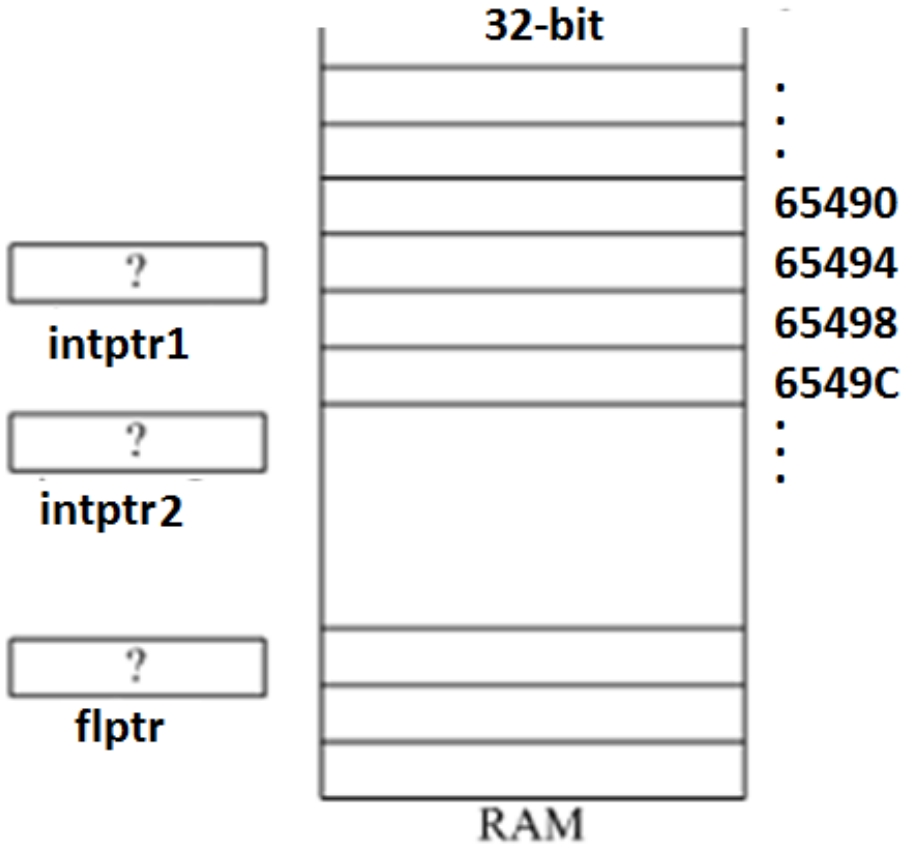
Mendeklarasikan Variabel Pointer

```
1  /* Nama file: lokasiptr.c
2     Lokasi memori variabel pointer */
3  #include <stdio.h>
4  int main()
5  {
6    int *intptr1, *intptr2;
7    float *flptr;
8
9    printf("Alamat intptr1: %p\n", &intptr1);
10   printf("Nilai tersimpan = %p\n", intptr1);
11   printf("Alamat intptr2: %p\n", &intptr2);
12   printf("Nilai tersimpan = %p\n", intptr2);
13   printf("Alamat flptr: %p.\n", &flptr);
14   printf("Nilai tersimpan = %p\n", flptr);
15
16   return 0;
```

Mendeklarasikan Variabel Pointer

- Variabel *intptr1* dan *intptr2* adalah dua buah variabel pointer ke jenis integer, sedangkan *flptr* adalah sebuah pointer yang menunjuk ke jenis data float.
- Keadaan ketiga variabel pointer di atas pada saat dideklarasikan dapat diilustrasikan seperti pada Gambar 16.

Mendeklarasikan Variabel Pointer



Gambar 4: Alamat Memori Variabel Pointer

Mendeklarasikan Variabel Pointer

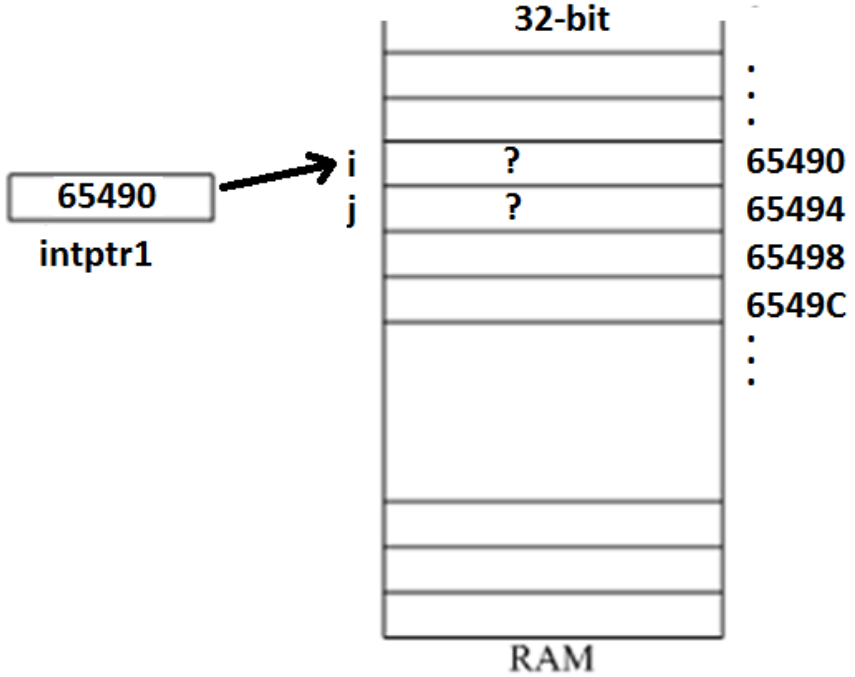
- Ketiga variabel pointer yang dideklarasikan di atas, belum menunjuk ke alamat memori tertentu, dinyatakan dengan tanda tanya.
- Ketiga variabel tersebut sebenarnya sudah memuat alamat memori, tetapi alamat ini adalah sembarang alamat memori pada saat itu.
- Karena kita tidak mempunyai pengetahuan tentang alamat ini, maka dalam dunia pemrograman nilai ini disebut sampah (*garbage*).

Inisialisasi Variabel Pointer

- Pemberian harga awal pada variabel-variabel pointer dapat sekaligus dilakukan pada saat variabel-variabel tersebut dideklarasikan.
- Harga awal yang diberikan adalah alamat lokasi memori.
- Contoh:
  int i, j, *intptr1 = &i;
- Variabel *i*, dan *j* adalah variabel-variabel berjenis integer, sementara *intptr1* adalah variabel berjenis pointer ke jenis data integer.

Inisialisasi Variabel Pointer

- Misalkan *i*, dan *j* berturut-turut menempati lokasi memori 65490 dan 65494.



Gambar 5: Pemberian alamat *i* ke pointer *intptr1*.

Inisialisasi Variabel Pointer

- Dalam memberikan harga awal variabel pointer pada saat pendeklarasian, hendaklah diperhatikan urutan penulisannya.
- Pemberian harga awal pada variabel *intptr1* berikut:
  int **intptr1* = &*i*, *i*;

  akan mengakibatkan terjadinya kesalahan, karena pada saat memberikan harga awal pada *intptr1*, kompiler belum mengetahui alamat variabel *i*.

Inisialisasi Variabel Pointer

```
1  /* Nama file: initptr.c
2     Lokasi memori pointer */
3  #include <stdio.h>
4
5  int main()
6  {
7    int x, y, *intptr1 = &x, *intptr2 = &y;
8    float f, *flptr = &f;
9
10   printf("Alamat x: %p.\n", &x);
11   printf("Alamat y: %p.\n", &y);
12   printf("Alamat f: %p.\n", &f);
```

Inisialisasi Variabel Pointer

```
13    printf("Alamat intptr1 %p.\n", &intptr1);
14    printf("Nilai tersimpan = %p\n", intptr1);
15    printf("Alamat intptr2 %p.\n", &intptr2);
16    printf("Nilai tersimpan = %p\n", intptr2);
17    printf("Alamat flptr: %p.\n", &flptr);
18    printf("Nilai tersimpan = %p\n", flptr);
19
20    return 0;
21 }
```

Mengakses Nilai yang Ditunjuk oleh Pointer

- Untuk mengakses nilai yang ditunjuk oleh sebuah pointer, digunakan operator *.
- Contoh:

```
int *intptr, i;
i = 5;
intptr = &i;
printf("%d", *intptr);
```

- Pada contoh ini, alamat variabel *i* diberikan kepada pointer *intptr*. Untuk mendapatkan nilai yang tersimpan dalam alamat itu, digunakan **intptr*.
- Output: 5

Mengganti Nilai yang Ditunjuk oleh Pointer

- Contoh:

```
int i, *intptr = &i;
i = 5;
printf("%d\n", *intptr);
i = 1;
printf("%d\n", *intptr);
```

- Alamat *i* diberikan ke pointer *intptr*.
- Setelah mencetak 5 berupa nilai *i* yang ditunjuk pointer *intptr*, kemudian nilai *i* diubah ke 1.
- Karena *intptr* dan alamat *i* adalah sama, **intptr* memberikan nilai 1.

Operator Indireksi dan Alamat

- Pada pembahasan sebelumnya, operator & memberikan alamat suatu variabel.
- Sementara operator * memberikan nilai yang terdapat pada suatu lokasi memori tertentu, disebut **dereference**.
- Dengan memanfaatkan kedua operator ini, kita dapat melakukan pemberian harga kepada suatu variabel secara tidak langsung.
- Contoh:

Operator Indireksi dan Alamat

```
1  /* Namafile: ptroper1.c
2     Ilustrasi indirection pointer */
3  #include <stdio.h>
4
5  int main()
6  {
7    int i, j, *intptr;
8
9    printf("Alamat i = %p\n", &i);
10   i = 890;
11   printf("i menerima nilai = %d\n", i);
```

Operator Indireksi dan Alamat

```
12    printf("Alamat intptr = %p\n", &intptr);
13    intptr = &i;
14    printf("intptr menerima alamat %p\n",
15        intptr);
16    printf("Alamat j = %p\n", &j);
17    j = *intptr;  /* dereference */
18    printf("j menerima nilai = %d\n", j);
19
20    return 0;
21 }
```

Operator Indireksi dan Alamat

- Baris 10: Variabel *i* menerima nilai 890.
- Baris 13: Variabel pointer *intptr1* menerima alamat variabel *i*, sehingga *intptr1* menunjuk ke alamat itu.
- Pemberian alamat ini dilakukan dengan menggunakan operator alamat & untuk memberikan alamat variabel *i* kepada variabel pointer *intptr1*.
- Baris 17: Variabel *j* menerima nilai yang ditunjuk oleh variabel *intptr1*, sehingga variabel *j* = 890 (disebut *dereference*)
- Secara tidak langsung, variabel *j* menerima nilai variabel *i*.

Operator Indireksi dan Alamat

- Sebenarnya, ketiga pernyataan di atas sama saja dengan pernyataan:

  ```
  j = i;
  ```

  akan tetapi pada kelompok pernyataan tersebut, pemberian nilai terhadap *j* tidak dilakukan secara langsung, melainkan dilakukan melalui variabel pointer *intptr1*.
- Oleh karena itu, * disebut juga operator tak langsung (*indirection operator* atau operator indireksi).

Alokasi Memori Dinamik

- Lokasi memori yang ditunjuk oleh sebuah variabel pointer dapat pula diberikan secara dinamik.
- Pengalokasian memori dilakukan menggunakan fungsi pustaka **malloc()** yang terdapat di berkas judul **stdlib.h**.
- Pengalokasian dengan cara ini dikatakan dinamik karena memori yang akan ditunjuk oleh sebuah pointer dapat diberikan pada saat memori itu diperlukan.

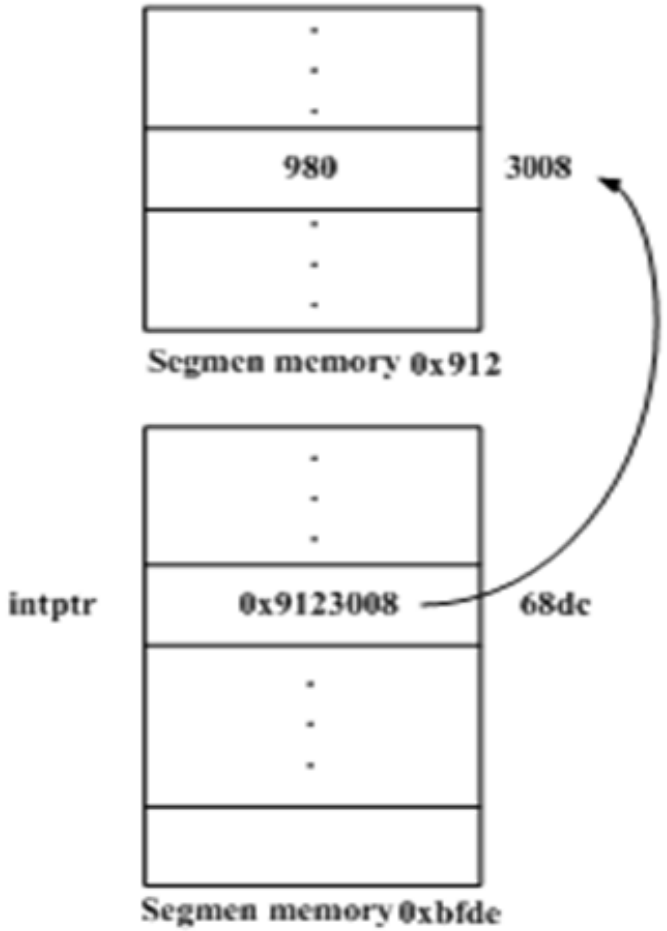
Alokasi Memori Dinamik

```
1  /* Namafile: memaloc.c
2     Ilustrasi dereference pointer */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8    int *intptr;
9
10   printf("Alamat intptr = %p\n", &intptr);
11   /* mengalokasikan memori */
12   intptr = (int *) malloc(sizeof(int));
```

Alokasi Memori Dinamik

```
13    printf("intptr menerima alamat %p\n",
14         intptr);
15
16    *intptr = 980;
17
18    printf("Isi memori yang ditunjuk intptr=
19         %d\n", *intptr);
20
21    return 0;
22  }
```

Alokasi Memori Dinamik



Gambar 6: Ilustrasi keadaan memori program

Alokasi Memori Dinamik

- Alokasi memori tersebut dapat dikembalikan ke pool memori apabila tidak diperlukan lagi menggunakan fungsi pustaka **free()**.
- Setelah alokasi memori tersebut dikembalikan ke *pool* memory, upaya mengakses kembali lokasi memori tersebut akan memberikan nilai *garbage*.
- Perhatikan contoh program berikut:

Alokasi Memori Dinamik

```
1  /* Namafile: freememory.c
2     Ilustrasi fungsi free() */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8    int *intptr;
9
10   printf("Alamat intptr = %p\n", &intptr);
11   /* mengalokasikan memori */
12   intptr = (int *) malloc(sizeof(int));
```

Alokasi Memori Dinamik

```
13    printf("intptr menerima alamat %p\n",
14        intptr);
15    *intptr = 980;
16    printf("Isi memori yang ditunjuk intptr=
17        %d\n", *intptr);
18
19    free(intptr);
20
21    printf("Isi memori yang ditunjuk intptr=
22        %d\n", *intptr);
23
24    return 0;
25 }
```