TIF1101 – Dasar-Dasar Pemrograman

HO 13 - Kelas Penyimpanan Variabel

Opim Salim Sitompul

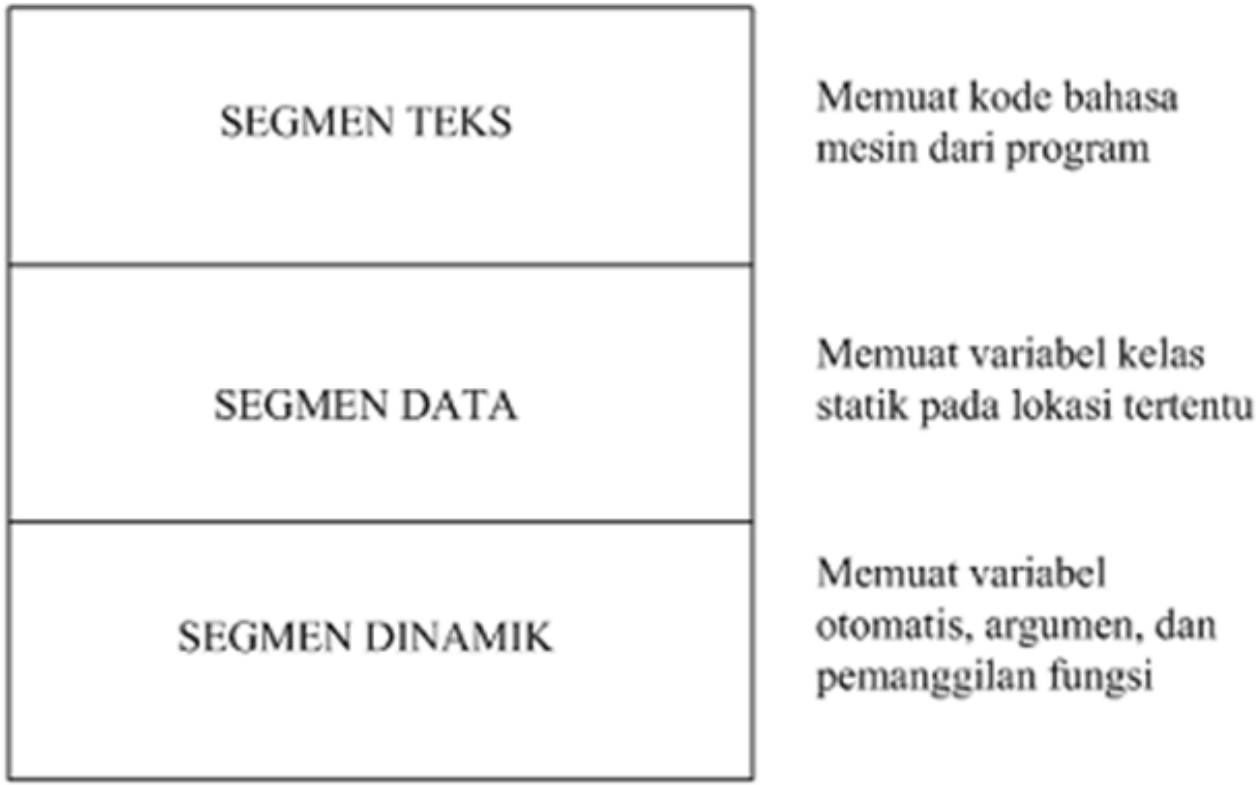
Department of Information Technology
Universitas Sumatera Utara

Outline

Pendahuluan

- Memori komputer disusun oleh C menjadi 3 bagian.
- Masing-masing disebut segmen teks, segmen data, dan segmen dinamik.
- Lokasi relatif ketiga segmen ini tergantung komputer.
- Kompiler C mengatur penggunaan segmen-segmen memori tersebut untuk menyimpan kode-kode program hasil kompilasi sesuai dengan kelompoknya.

Pendahuluan



Gambar 1: Pembagian segmen memori

Pendahuluan

- Segmen teks adalah bagian memori yang memuat kode-kode program berupa instruksi mesin hasil kompilasi.
- Segmen data adalah bagian memori yang memuat variabel-variabel yang menetap pada lokasi tertentu, yaitu variabel-variabel dari kelas statik.
- Segmen dinamik adalah bagian memori yang memuat variabel-variabel otomatis, argumen fungsi, dan kode program yang memuat badan fungsi.
  - Bagian memori dinamik ukurannya berubah-ubah tergantung dari pemanggilan fungsi.
  - Segmen dinamik akan berisi variabel, dan setelah kembali ke fungsi pemanggil, segmen memori ini akan bebas kembali.

Ruang Lingkup dan Masa Hidup

- Variabel-variabel yang dideklarasikan di dalam program C memiliki ruang lingkup (*scope*) dikenalnya variabel tersebut.
- *Scope* variabel ini menentukan apakah variabel dikenal oleh semua bagian program atau hanya dikenal oleh bagian-bagian program tertentu saja.
  - Pada sebuah fungsi,
  - Pada sekumpulan fungsi di dalam berkas yang sama,
  - Pada semua fungsi di dalam seluruh program.

Ruang Lingkup dan Masa Hidup

- Selain *scope*, variabel juga memiliki masa hidup (*lifetime*).
- *Lifetime* variabel menunjukkan apakah tempat penyimpanan variabel di dalam memori masih tersedia atau tidak.
- Beberapa jenis variabel berada di dalam memori selama program dijalankan, sebagian lain hanya ada di dalam memori selama pemanggilan fungsi.

Ruang Lingkup dan Masa Hidup

- Berdasarkan *scope* dan *lifetime*, variabel dapat digolongkan atas beberapa kelas penyimpanan.

Kelas Penyimpanan	Ruang Lingkup	Masa Hidup	Harga Awal
Otomatis	Fungsi tempat dideklarasikan	Selama pemanggilan fungsi	-
Register	Fungsi tempat dideklarasikan	Selama pemang-gilan fungsi	-
Eksternal	Seluruh program	Seluruh program	0 (Nol)
Statik	Fungsi tempat dideklarasikan	Seluruh program (menetap di memori selama program dijalankan)	0 (Nol)

Gambar 2: Kelas penyimpanan variabel

Ruang Lingkup dan Masa Hidup

- Kelas penyimpanan otomatis dan register tidak memiliki harga awal.
  - Pada saat variabel dideklarasikan, kompiler hanya menyediakan tempat untuk variabel tersebut, tanpa menentukan nilai apa yang tersimpan di dalamnya untuk pertama sekali.
- Untuk kelas penyimpanan eksternal dan statik kompiler memberikan harga awal 0 pada saat pertama sekali variabel tersebut dideklarasikan.
- Pemberian harga awal hanya sekali dilakukan oleh kompiler, yaitu pada saat fungsi atau bagian program yang memuat deklarasi itu dieksekusi pertama kalinya.

Kelas Penyimpanan Otomatis

- Kelas penyimpanan otomatis dikenal dengan kata kunci **auto**.
- Variabel dalam kelas ini disebut juga variabel lokal.
- Karena variabel-variabel ini secara *default* adalah otomatis jika dideklarasikan di dalam sebuah fungsi, maka kata kunci **auto** jarang digunakan.
- Variabel otomatis adalah semua variabel yang dideklarasikan di dalam badan sebuah fungsi.

Kelas Penyimpanan Otomatis

- Lifetime variabel otomatis adalah selama fungsi dieksekusi.
- Memori tempat penyimpanan variabel hanya tersedia pada saat fungsi dipanggil, dan akan musnah pada saat kendali dikembalikan ke fungsi pemanggil.
- Pada saat dideklarasikan, variabel-variabel otomatis punya harga awal sembarang bilangan yg terdapat pada lokasi memori saat itu (disebut *garbage*).
  - Oleh karena itu pemrogram bertanggung jawab untuk memberikan harga awalnya agar tidak terjadi kesalahan pada saat menggunakannya.

Kelas Penyimpanan Otomatis

```
1  /* Nama file: poskar.c
2     Menentukan karakter huruf keberapa */
3  #include <stdio.h>
4  int periksaKarakter(char);
5  int tentukanLokasi(char);
6
7  int main()
8  {
9    char ch;
10
11   printf("Tuliskan sembarang kalimat:\n");
12   while((ch = getchar()) != 10)
13   {
```

Kelas Penyimpanan Otomatis

```
14    if (ch != 32)
15      printf("\nkarakter %c", ch);
16    if (periksaKarakter(ch) == 1)
17      printf(", nomor %2d dalam alphabet.",
18        tentukanLokasi(ch));
19    }
20    printf("\n");
21
22    return 0;
23  }
```

Kelas Penyimpanan Otomatis

```
25 int periksaKarakter(char ch)
26 {
27   if(((ch > 64) && (ch < 91)) || (((ch > 96)
         && (ch < 123))))
28     return 1;
29   return -1;
30 }
31
32 int tentukanLokasi(char ch)
33 {
34   char kar;
35
36   kar = ch^32;
37   return (kar - 64);
38 }
```

Kelas Penyimpanan Otomatis

- Selain kelas penyimpanan otomatis yang dideklarasi pada baris-baris pertama badan fungsi, terdapat satu kelas penyimpanan lain, yaitu kelas penyimpanan blok.
- Berbeda dengan variabel otomatis biasa, variabel kelas penyimpanan ini hanya memiliki *scope* sebatas blok di mana dideklarasikan, sedangkan *lifetime*nya sama seperti *lifetime* variabel otomatis biasa.
- Variabel-variabel ini hanya dapat diakses dari dalam blok, bagian kode program yang terletak di luar blok tidak mengenal adanya variabel tersebut.

Kelas Penyimpanan Otomatis

```
1  /* Namafile: scope.c */
2  #include <stdio.h>
3  int main()
4  {
5    int x = 1;
6    int y = 2;
7    printf("x=%d\n", x);
8    printf("y=%d\n", y);
9    {
10     double x = 3.141592;
11     y  += (int) x;
12     printf("y=%d\n", y);
13   }
14   y  += x;
15   printf("y=%d\n", y);
16   return 0;
```

Kelas Penyimpanan Register

- Kelas penyimpanan register dikenal dengan kata kunci **register** dan digunakan pada jenis data integer.
- Kelas penyimpanan ini memberitahu kompiler bahwa akses terhadap objek data harus secepat mungkin asal saja hal itu secara fisik dapat dilakukan.
- Pada dasarnya, penggunaan kelas penyimpanan register merupakan upaya untuk menambah kecepatan eksekusi.

Kelas Penyimpanan Register

- Bila soal kecepatan ini menjadi perhatian, pemrogram dapat memilih beberapa variabel yang paling sering diakses, misalnya variabel pengulangan dan argumen-argumen fungsi untuk dideklarasikan sebagai variabel-variabel register.
- Bila kompiler tidak dapat mengalokasikan sebuah register, kelas penyimpanan ini secara default menjadi otomatis.

Kelas Penyimpanan Otomatis

```
1  /* Nama file: balikbil.c
2     Program membalik bilangan */
3  #include <stdio.h>
4  int balikBilangan(register);
5
6  int main()
7  {
8    int n;
9
10   printf("Berikan sebuah bilangan bulat: ");
11   scanf("%d", &n);
12   printf("Bilangan saudara sekarang: %d\n",
        balikBilangan(n));
13   return 0;
14 }
```

Kelas Penyimpanan Otomatis

```
15 int balikBilangan(register m)
16 {
17       register rev = 0;
18
19       while (m > 0)
20       {
21      rev = rev * 10 + m % 10;
22      m = m / 10;
23       }
24       return rev;
25 }
```

Kelas Penyimpanan Eksternal

- Variabel eksternal adalah variabel-variabel yang dideklarasikan di luar fungsi dan dikenal dengan kata kunci **extern**.
- Sifatnya adalah global terhadap semua fungsi yang didefinisikan setelah pendeklarasian variabel ini.
- Semua fungsi pada program dapat mengakses variabel tersebut dan akan tetap berada di dalam memori selama program dieksekusi.

Kelas Penyimpanan Eksternal

```
1  /* Nama file: fibo1.c
2      Deret n bilangan fibonacci */
3
4  #include <stdio.h>
5
6  int fibonacci(int); /* prototipe fungsi */
7  int f1 = 1, f2 = 1;    /* variabel-variabel
       global */
8
9  int main()
10 {
11   int i, n;
12
13   printf("Banyaknya bilangan fibonacci: ");
14   scanf("%d", &n);
```

Kelas Penyimpanan Eksternal

```
15    printf("Deret %d bilangan fibonacci:\n", n);
16    for(i=1; i <= n; ++i)
17      printf("%d ", fibonacci(i));
18    printf("\n");
19
20    return 0;
21  }
```

Kelas Penyimpanan Eksternal

```
22 int fibonacci(int f)
23 {
24   f = (f <= 2) ? 1 : f1 + f2;
25   f2 = f1;
26   f1 = f;
27   return f;
28 }
```

Kelas Penyimpanan Eksternal

- Selain dapat dideklarasikan pada sebuah berkas program, juga dapat dideklarasikan pada berkas lain apabila program yang disusun terdiri dari beberapa berkas.
- Fungsi yang terletak pada sebuah berkas dapat mengakses variabel-variabel eksternal pada berkas lain dengan memberi kata kunci **extern** pada variabel-variabel yang diinginkan.
  - Kata kunci extern ini akan memberitahu kompiler bahwa variabel dimaksud telah dideklarasikan di tempat lain.
  - Harga awal yang diberikan oleh kompiler pada variabel-variabel eksternal adalah nol.
  - Harga awal ini dapat diubah sesuai kebutuhan.

Kelas Penyimpanan Eksternal

```
1  /* Nama File: fibo2a.c
2   Menampilkan deret n bilangan fibonacci */
3
4  #include <stdio.h>
5
6  int f1 = 1, f2 = 1; /* variabel eksternal */
7  int fibonacci(int); /* prototipe fungsi */
8
9  int main()
10 {
11   int i, n;
12
13   printf("Banyak bilangan fibonacci: ");
14   scanf("%d", &n);
```

Kelas Penyimpanan Eksternal

```
15
16    printf("Deret %d bilangan fibonacci:\n", n);
17    for(i=1; i <= n; ++i)
18      printf("%d ", fibonacci(i));
19    printf("\n");
20
21    return 0;
22  }
```

Kelas Penyimpanan Eksternal

```
1  /* Nama file: fibo2b.c */
2
3  extern int f1, f2;  /* variabel eksternal */
4
5  int fibonacci(int f)
6  {
7    f = (f <= 2) ? 1 : f1 + f2;
8    f2 = f1;
9    f1 = f;
10
11   return f;
12 }
```

Kelas Penyimpanan Static

- Variabel dari kelas penyimpanan statik dikenal dengan kata kunci **static**.
- Variabel **static** bersifat lokal terhadap fungsi tempat dideklarasikan, tetapi nilainya akan tetap tersimpan di memori walaupun fungsi itu selesai dieksekusi dan kendali dikembalikan ke fungsi pemanggil.
- Apabila fungsi tersebut kembali dipanggil, variabel tadi akan tetap memiliki nilai yang terakhir kali disimpan.
- Harga awal yang diberikan kompiler pada variabel-variabel statik adalah 0.

Kelas Penyimpanan Static

```
1   /* filename: simple_static.c
2   Demonstrasi variabel static */
3
4   #include<stdio.h>
5
6   void simpel(); /* prototype */
7
8   int main()
9   {
10    simpel(); /* Call 1 */
11    simpel(); /* Call 2 */
12    simpel(); /* Call 3 */
13
14    return 0;
15  }
```

Kelas Penyimpanan Static

```
16 /* Function definition */
17 void simpel()
18 {
19   int x = 0;
20   static int y = 1;
21
22   printf("a = %d\n", x);
23   printf("b = %d\n", y);
24
25   x++;
26   y++;
27 }
```

Kelas Penyimpanan Static

```
1  /*Nama file: fibo3.c
2     Menampilkan deret n bilangan fibonacci */
3
4  #include <stdio.h>
5
6  int fibonacci(int); /* Prototipe fungsi */
7
8  int main()
9  {
10   int i, n;
11
12   printf("Berikan banyaknya bil fibonacci: ");
13   scanf("%d", &n);
14   printf("Deret %d bilangan fibonacci:\n", n);
```

Kelas Penyimpanan Static

```
15    for(i=1; i <= n; ++i)
16      printf("%d ", fibonacci(i));
17    printf("\n");
18    return 0;
19  }
20
21  int fibonacci(int f)
22  {
23    /* f1 dan f2 bersifat statik */
24    static int f1 = 1, f2 = 1;
25
26    f = (f <= 2) ? 1 : f1 + f2;
27    f2 = f1;
28    f1 = f;
29    return f;
30  }
```

Kelas Penyimpanan Static

- Selain dapat digunakan secara internal di dalam sebuah fungsi, kelas penyimpanan statik juga dapat digunakan secara eksternal pada program-program multi berkas.
- Dengan pendeklarasian eksternal statik, berkas yang satu dapat kembali mengakses nilai yang tersimpan pada berkas yang lain.
- Masa hidup variabel dari kelas eksternal statik adalah selama program dijalankan, dan ruang lingkupnya meliputi berkas-berkas di mana variabel itu dideklarasikan.

Kelas Penyimpanan Static

```
1  /* Program 10.6
2  Nama file: fibo3a.c
3  Program deret n bilangan fibonacci */
4  #include <stdio.h>
5  extern int fibonacci(int);
6  int main()
7  {
8    int i, n;
9    printf("Berikan banyaknya bil fibonacci: ");
10   scanf("%d", &n);
11   printf("Deret %d bilangan fibonacci:\n", n);
12   for(i=1; i <= n; ++i)
13   printf("%d ", fibonacci(i));
14   printf("\n");
15   return 0;
16 }
```

Kelas Penyimpanan Static

```
17 /* Program 10.6
18 Nama file: fibo3b.c
19 Program deret n bilangan fibonacci */
20
21 int fibonacci(int f)
22 {
23   /* f1 dan f2 bersifat statik */
24   static int f1 = 1, f2 = 1;
25
26   f = (f <= 2) ? 1 : f1 + f2;
27   f2 = f1;
28   f1 = f;
29
30   return f;
31 }
```