# Contoh dan Penjelasan BAHASA SINGKONG

## Mahir Bekerja dengan GUI

**Dilengkapi Pengantar:**

**❶Analisis Kebutuhan Perangkat Lunak**

**❷Pemodelan Matematika dalam Ilmu Komputer**

```
var p = [0, "Connecting..."]

var f = fn() {
    delay(random(1000, 1500))
    set(p, 0, random(30, 35))
    set(p, 1, "Downloading...")
    delay(random(2000, 2500))
    set(p, 0, random(70, 75))
    set(p, 1, "Processing...")
    delay(random(3000, 3500))
    set(p, 0, 100)
    set(p, 1, "Done")
    delay(1000)
}

var t = thread(f, p)

var s = number(@)

reset()
var x = component("progress", "")
add_s(x)

var c = fn() {
  if (thread_alive(t)) {
    var m = number(@) - s
    statusbar(0, m + " ms", true)
    statusbar(1, p[1], true)
    statusbar(2, p[0] + "%", true)
    config(x, "contents", p[0])
  } else {
    stop()
  }
}

timer(random(100, 500), c)
show()
```

**Dr. Noprianto**
**Dr. Wartika**
**Dr. Ford Lumban Gaol**

Penerbit PT. Stabil Standar Sinergi

**Contoh dan Penjelasan Bahasa Singkong: Mahir Bekerja dengan GUI**

Penulis: Dr. Noprianto, Dr. Wartika, Dr. Ford Lumban Gaol

ISBN: 978-602-52770-4-7

Penerbit:

PT. Stabil Standar Sinergi

| | |
|---|---|
| Alamat | Puri Indah Financial Tower<br>Lantai 6, Unit 0612<br>Jl. Puri Lingkar Dalam Blok T8, Puri Indah,<br>Kembangan, Jakarta Barat 11610 |
| Website | www.singkong.dev |
| Email | info@singkong.dev |

Cover buku:

- Masakan: singkong goreng oleh Meike Thedy, S.Kom.
- Desain cover oleh Noprianto, menggunakan GIMP. Filter yang digunakan adalah: Maze. Font yang digunakan adalah Calibri. Masing-masing teks tulisan (kecuali tulisan Penerbit) pada cover dibuat dengan beberapa layer untuk mendapatkan efek outline.
- Cuplikan kode adalah bagian dari contoh dalam buku.

# Daftar Isi

# Kata Pengantar

Buku ini berisi sejumlah contoh source code dan penjelasan langkah demi langkah yang mudah dipahami untuk membuat program komputer yang dilengkapi Graphical User Interface (GUI), dengan bahasa pemrograman Singkong.

Contoh-contoh yang dibahas mencakup berbagai topik lanjutan dalam pembuatan GUI, sehingga, setelah membaca buku ini, kami mengharapkan Anda mahir bekerja dengan GUI. Oleh karena itu, agar dapat mengikuti pembahasan dengan nyaman, pernah membuat aplikasi GUI dengan bahasa Singkong akan sangat membantu.

Melengkapi contoh program, buku ini juga membahas pengantar untuk analisis kebutuhan pengembangan perangkat lunak dan pemodelan matematika dalam ilmu komputer.

Jakarta, September 2022

Tim penulis

Untuk referensi dan dokumentasi lengkap bahasa Singkong, serta dasar-dasar aplikasi GUI dengan bahasa Singkong, Anda juga dapat membaca buku-buku gratis berikut:

- Mengenal dan Menggunakan Bahasa Pemrograman Singkong (ISBN: 978-602-52770-1-6; Dr. Noprianto; 2020-2022; diterbitkan oleh PT. Stabil Standar Sinergi)
- Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar aplikasi GUI (ISBN: 987-602-52770-3-0; Dr. Noprianto, Dr. Karto Iskandar, Benfano Soewito, Ph.D.; 2022; diterbitkan oleh PT. Stabil Standar Sinergi)

Semua buku tersebut juga dapat di-download dari:
https://singkong.dev

# Persiapan

Pertama-tama, siapkanlah sebuah komputer, yang dilengkapi layar, keyboard, dan mouse/trackpad. Walaupun dengan tablet atau ponsel juga memungkinkan secara teknis, akan diperlukan pengaturan/instalasi tambahan yang tidak dibahas dalam buku ini.

Untuk perangkat keras komputernya, dapat menggunakan spesifikasi komputer mulai dari yang terbaru ataupun yang telah dijual sekitar 20 tahun yang lalu. Yang penting, dapat menjalankan salah satu dari daftar sistem operasi berikut.

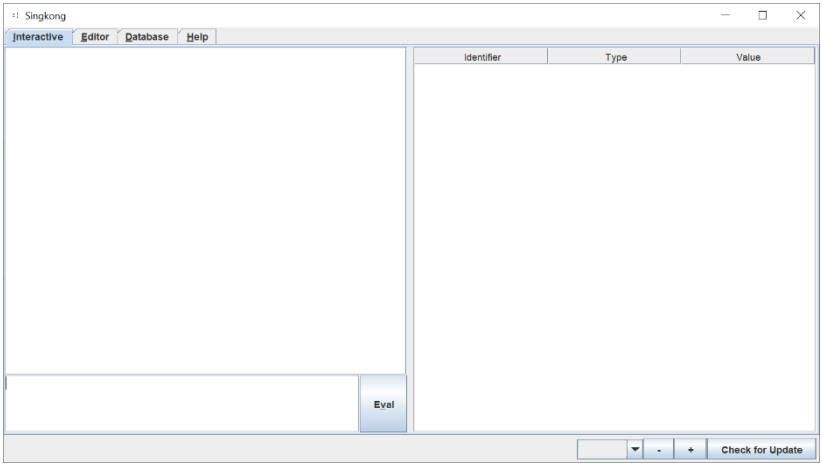
Interpreter Singkong (dan program yang Anda buat nantinya) dapat berjalan pada berbagai sistem operasi berikut:

- macOS (mulai dari Mac OS X 10.4 Tiger)
- Windows (mulai dari Windows 98)
- Linux (mulai yang dirilis sejak awal 2000-an; juga termasuk Raspberry Pi OS dan Debian di Android)
- Chrome OS (sejak tersedia Linux development environment, telah diuji pada versi 101 di chromebook)
- Solaris (telah diuji pada versi 11.4)
- FreeBSD (telah diuji pada versi 13.0 dan 12.1)
- OpenBSD (telah diuji pada versi 7.0 dan 6.6)
- NetBSD (telah diuji pada versi 9.2 dan 9.0)

Setelah komputer siap, lakukanlah instalasi Java, apabila belum terinstal sebelumnya. Secara teknis, Anda hanya membutuhkan Java Runtime Environment, versi 5.0 atau lebih baru. Versi 5.0 dirilis pada tahun 2004 (sekitar 18 tahun lalu pada saat buku ini ditulis). Gunakan versi yang masih didukung secara teknis, apabila memungkinkan. (Kenapa perlu menginstalasi Java? Karena interpreter Singkong ditulis dengan bahasa Java dan bahasa Singkong itu sendiri.)

Sebagai langkah terakhir, downloadlah interpreter Singkong, yang akan selalu didistribusikan sebagai file jar tunggal (Singkong.jar). Downloadlah selalu dari https://nopri.github.io/Singkong.jar. Pada saat buku ini ditulis, ukurannya hanya 4,3 MB dan berisikan semua yang diperlukan untuk mengikuti semua contoh dalam buku ini.

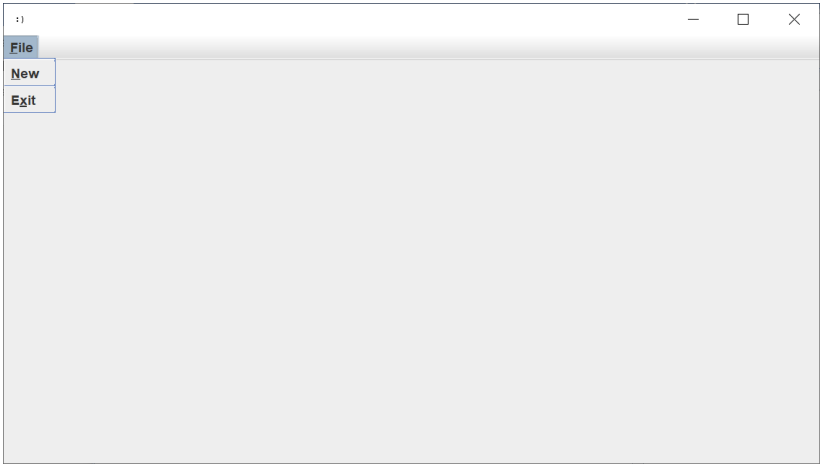
Apabila Singkong.jar dapat dijalankan, maka persiapan sudah selesai (silahkan langsung melanjutkan ke contoh pertama). Perhatikanlah bahwa kita akan aktif pada tab Interactive untuk menguji kode program secara langsung dan tab Editor (sebagian besar contoh) untuk mengetikkan, menyimpan/membuka, dan menjalankan kode program yang lebih panjang.




Apabila langkah detil instalasi Java dan menjalankan Singkong.jar diperlukan, bacalah juga halaman 5 pada buku gratis: *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar aplikasi GUI*.

Dan, apabila Anda perlu mendistribusikan program yang Anda buat dengan bahasa Singkong dalam satu file jar yang dapat dijalankan, bacalah juga bab *Distribusi Aplikasi* pada buku gratis: *Mengenal dan Menggunakan Bahasa Pemrograman Singkong*.

## Contoh 1: Menu bar sederhana




Apabila program yang dikembangkan menyediakan cukup banyak fungsionalitas, kita mungkin perlu mengatur agar program tetap nyaman digunakan. Setidaknya, dari sisi komponen user interface, hanya yang paling penting dan relevan yang tampil.

Salah satu contoh pengaturan yang dapat dilakukan adalah dengan menggunakan menu bar. Fungsi-fungsi yang disediakan dapat dikelompokkan sesuai kategorinya (misal: File, Edit, Help) dan tidak perlu tampil sekaligus. Apabila diperlukan, pengguna dapat memilih dari menu-menu tersebut.

Untuk membuat menu bar, kita cukup memanggil satu fungsi BUILTIN, yaitu menubar. Fungsi ini pun hanya membutuhkan satu argumen ketika dipanggil. Hanya saja, apabila dilihat sekilas dari dokumentasi fungsinya, argumen tersebut perlu dibahas lebih lanjut sebagai berikut.

- Berupa sebuah ARRAY.
    - Dari ARRAY yang terdiri dari 3 elemen.
        - Yang pertama adalah nama menu berupa STRING.
        - Yang kedua adalah indeks mnemonik berupa NUMBER, yang secara visual dapat berupa karakter keberapa yang digarisbawahi.
        - Yang ketiga adalah ARRAY lagi, yang mendefinisikan menu item untuk setiap elemen bertipe ARRAY di dalamnya:
            - Apabila kosong, maka dianggap separator.
            - Apabila tidak, maka ARRAY 4 elemen.
                - Yang pertama adalah item berupa STRING.
                - Yang kedua adalah indeks mnemonik berupa NUMBER.
                - Yang ketiga adalah apakah aktif atau tidak, berupa BOOLEAN.
                - Yang terakhir adalah FUNCTION yang dipanggil apabila item menu ini diaktifkan.

Anda mungkin berpendapat: tak heran buku ini ditulis. Fungsi boleh satu, argumen fungsi boleh satu, tapi sekilas terlihat seperti sepuluh. Dan sejujurnya, ketika dituliskan sebagai poin-poin tersebut, penulis pun berpendapat yang sama.

Mari kita bahas sejelas mungkin, lewat kode program berikut.

```
var menu = [
    ["File", 0,
      [
        ["Exit", 1, true, fn(){}]
      ]
    ]
  ]

  reset()
  menubar(menu)
  show()
```

Fungsi reset dan show telah dibahas pada buku gratis: *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI*, sehingga pada dasarnya, program kita hanya terdiri dari pemanggilan fungsi menubar, dengan argumen berupa ARRAY menu yang dibuat sebelumnya.

- Bisa kita lihat, menu adalah sebuah ARRAY. Maka, kita sudah turuti aturan bahwa argumen ke menubar adalah sebuah ARRAY.
    - Diwarnai biru: ARRAY menu berisi sebuah elemen berupa ARRAY, yang terdiri dari 3 elemen:
        - Yang pertama adalah STRING “File”.
        - Yang kedua adalah NUMBER 0, yang secara visual, huruf F akan digarisbawahi. Dalam hal ini, F adalah indeks ke 0 dari “File”.
        - Yang ketiga adalah ARRAY lagi, dengan satu elemen berupa ARRAY:

- ARRAY ini tidak kosong, maka bukan separator.
- Diwarnai hijau: Sebaliknya, merupakan ARRAY dengan 4 elemen:
    - Yang pertama adalah STRING “Exit”.
    - Yang kedua adalah NUMBER 1, yang secara visual, huruf x akan digarisbawahi, karena indeks pertama dari “Exit” adalah x.
    - Item ini aktif, dalam artian, Exit bisa dipilih atau diklik oleh pengguna. Elemen ketiga ini berupa true yang bertipe BOOLEAN.
    - Dan, elemen terakhir, adalah FUNCTION tanpa nama dan kosong.

Apakah sampai di sini, kita juga sependapat bahwa penjelasan tersebut masih kurang? Apabila iya, mari kita bahas lebih lanjut ARRAY yang mendefinisikan menu item, dengan kode program berikut.

```
var menu = [
  ["File", 0,
    [
        ["New", 0, true, fn(){}],
        [],
```

```
            ["Exit", 1, true, fn(){}]
        ]
      ]
    ]


    reset()
    menubar(menu)
    show()
```

Apabila dijalankan dan ketika menu File diklik misalnya, maka akan tampil:

- Diwarnai biru: New, dengan N digarisbawahi
- Sebuah separator
- Diwarnai hijau: Exit, dengan x digarisbawahi

Kode program tersebut malah menjadikan pembuatan menu bar terlihat lebih rumit? Kalau begitu, mari kita perjelas di kode program berikut (*lagi?*).

```
    var f = fn() {}
    var menu_item_file_new = ["New", 0, true,
f]
    var menu_item_file_exit = ["Exit", 1, true,
f]
    var menu_item_file = [
```

```
    menu_item_file_new,
    [],
    menu_item_file_exit
]
var menu_file = ["File", 0, menu_item_file]

var menu = [
    menu_file
]

reset()
menubar(menu)
show()
```

Lho, kenapa malah lebih panjang kode programnya? Padahal, kalau dijalankan, akan sama saja dengan contoh sebelumnya.

Ini karena, kita mencoba agar ARRAY menu dengan jelas menggambarkan bahwa kita akan memiliki sebuah menu, yang dari nama variabelnya, adalah menu File.

```
var menu = [
    menu_file
]
```

Variabel menu_file pun rasanya lebih mudah dipahami:

```
var menu_file = ["File", 0, menu_item_file]
```

Dapat dilihat, tulisan menu adalah “File”, dengan huruf F (indeks 0), akan digarisbawahi. Lalu, isi dari menu File ini adalah berupa variabel menu_item_file:

```
var menu_item_file = [
    menu_item_file_new,
    [],
    menu_item_file_exit
]
```

Yang menggambarkan dengan jelas bahwa kita akan memiliki tiga item, berupa variabel menu_item_file_new, sebuah ARRAY kosong (separator), dan menu_item_file_exit.

Definisi menu_item_file_new dan menu_item_file_exit pun lebih terbaca:

```
var menu_item_file_new = ["New", 0, true,
f]
var menu_item_file_exit = ["Exit", 1, true,
f]
```

Masing-masing dari mereka mirip dengan menu_file, dengan penambahan elemen berupa variabel f, yang mana berupa fungsi (FUNCTION) kosong:

```
var f = fn() {}
```

Bisa kita lihat bersama, walau dengan kode program lebih panjang, kita dapat membahas per bagian dengan lebih mudah (dan pastinya, bukan agar buku ini menjadi lebih tebal).

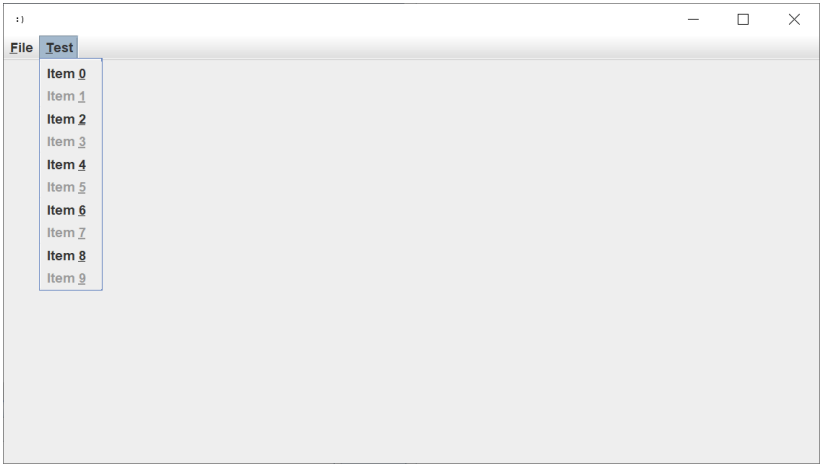
Sebagai bonus, dengan cara serupa, kita bisa membuat satu fungsi yang dipanggil oleh sejumlah menu item berbeda ketika diklik. Misal, dengan mengubah satu fungsi f, New dan Exit akan memiliki aksi yang sama:

```
var f = fn() {
    message("Belum selesai")
}
```

Dan sekarang, supaya kita bisa melanjutkan ke contoh berikut, mari kita akhiri dengan mengubah menu item Exit agar menutup frame, dengan menyesuaikan satu baris kode berikut:

```
var menu_item_file_exit = ["Exit", 1, true, fn() {frame_close()}]
```

## Contoh 2: Pembuatan menu bar secara dinamis




Di contoh sebelumnya, kita bekerja dengan ARRAY untuk menampilkan menu di menu bar. Memang baru ada menu File. Tapi, karena kita bisa mengubah isi ARRAY, termasuk menambahkan, mengurangi, dan mengubah elemen di dalamnya, kita bisa membuat isi dari menu bar secara dinamis.

Jadi, kita masih membahas menu bar? (*Astaga!*) Benar sekali. Dan, pembahasan-pembahasan di contoh kali ini akan lebih menantang dibanding contoh sebelumnya. Tapi, hasilnya akan sebanding.

Mari kita buat menu “Test”, melengkapi “File” yang telah ada sebelumnya, dengan menambahkan dua baris berikut sebelum var menu:

```
var menu_item_test = []
var menu_test = ["Test", 0, menu_item_test]
```

Kemudian, kita ubah var menu sebelumnya menjadi:

```
var menu = [
    menu_file,
    menu_test
]
```

Apabila kode programnya kita jalankan, maka perbedaan dengan contoh sebelumnya hanyalah sebuah menu Test, tanpa item apapun di dalamnya.

Sekarang, mari kita siapkan aksi ketika New diklik:

- Sebuah input akan ditampilkan, meminta label untuk menu item baru.
- Apabila input valid (tidak kosong), maka kita akan buat menu item baru sesuai label tersebut, dan menambahkannya ke menu "Test".

Kita akan menulis 8 baris kode baru, mengganti isi dari fungsi f yang dibuat sebelumnya, menjadi:

```
var f = fn() {
    var s = trim(input("Masukkan nama item: "))
    if (!empty(s)) {
        var m = [s, 0, true, fn(){}]
        var menu_item_test = menu_item_test + m
        menubar(menu)
    }
}
```

Begitu saja? Benar. Cukup begitu saja untuk saat ini. Sekarang, mari kita bahas isi fungsi f tersebut:

- Diwarnai biru: sebuah input box akan ditampilkan, dan pengguna dapat menutup (tanpa mengisi), menekan tombol Cancel, atau mengisikan dengan sejumlah spasi, misalnya. Untuk itu, kita akan menggunakan fungsi trim untuk membuang-dalam hal ini-spasi di depan dan belakang input.
- Diwarnai hijau: Apabila setelah ditrim tidaklah kosong, maka:
    - Diwarnai merah: Kita akan buat item baru sesuai label yang diinput (variabel s). Kita akan berikan indeks mnemonik berupa 0, yang artinya, secara visual, indeks pertama dari item akan dibarisgawahi. Menu item ini aktif (elemen ke-3, true), dan ketika diklik, tidak ada aksi yang dilakukan.
    - Diwarnai oranye: Berikutnya, perhatikanlah bahwa kita menggunakan variabel menu_item_test untuk ARRAY item-item dalam menu Test. Maka, kita tambahkan ke ARRAY tersebut.
    - Baris terakhir sangatlah penting. Isi ARRAY menu_item_test (dan oleh karenanya menu_test serta menu) telah berubah. Tapi, perubahan ini tidak berdampak pada menu bar. Kita akan buat ulang dengan kembali memanggil fungsi menubar.

Dengan cara dan semangat serupa, kita bisa mengubah aktif/tidak sebuah menu item atau bahkan mengubah aksinya ketika diklik.

Mari kita terjemahkan semangat kita menjadi kode program berikut, yang akan membuat sejumlah menu item secara dinamis, kemudian dengan klik pada menu item tertentu, sebagian menu yang dibuat dinamis tersebut, tidak lagi dapat diklik.

```
        var f = fn() {
            each(menu_item_test, fn(e, i) {
                if (i % 2 == 1) {
                    set(e, 2, false)
                }
            })
            menubar(menu)
        }


        var menu_item_file_disable = ["Disable", 0,
true, f]
        var menu_item_file_exit = ["Exit", 1, true,
fn() {frame_close()}]
        var menu_item_file = [
            menu_item_file_disable,
            [],
            menu_item_file_exit
        ]
        var menu_file = ["File", 0, menu_item_file]


        var menu_item_test = []
        each(range(0, 10, 1), fn(e, i) {
            var x = "Item " + e
            var m = [x, len(x)-1, true, fn(){
```

```
                message(x)
            }]
            var t =
variables()["MENU_ITEM_TEST"][1]
            var t = t + m
        })
        var menu_test = ["Test", 0, menu_item_test]


        var menu = [
            menu_file,
            menu_test
        ]


        reset()
        menubar(menu)
        show()
```

Sampai di titik ini, penulis berharap kita semua masih semangat. Pada kenyataannya, ketika menulis kode program tersebut, penulis juga belajar beberapa hal, yang juga dituliskan di penjelasan berikut:

- Pada saat program dijalankan, menu-menu File dan Test akan ditampilkan pada menu bar.
    - File akan berisi Disable dan Exit (yang tidak berubah dari sebelumnya).
    - Test, yang dibuat dinamis, akan terdiri dari Item 0 sampai Item 9, dimana 0 sampai 9 akan digarisbawahi.

    - Setiap menu item yang dibuat dinamis dapat diklik dan akan menampilkan message sesuai itemnya.
    - Ketika Disable diklik, maka item-item di dalam Test, yang indeks nya ganjil (Item 1, Item 3, Item 5, Item 7, Item 9) tidak lagi dapat diklik.
- Diwarnai biru: dengan fungsi each pada range tertentu, kita akan melakukan iterasi untuk setiap elemen dari nilai yang dihasilkan dari pemanggilan fungsi range tersebut. Contoh penggunaan each telah dibahas dibuku gratis: *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI*. Kita kemudian:
    - Membuat menu item secara dinamis.
    - Mengatur indeks mnemonik dengan indeks panjang string dikurangi 1 (karena dimulai dari 0).
    - Membuat fungsi secara dinamis, yang akan dipanggil ketika item tersebut diklik, dengan isi fungsi adalah menampilkan message.
- Diwarnai hijau: sesekali, Anda mungkin akan menemukan yang seperti ini di buku-buku bahasa Singkong:
    - Variabel yang dibuat pada sebuah fungsi (misal fn (e, i) {} pada each) akan lokal pada fungsi tersebut.
    - Kita akan mengakses variabel menu_item_test yang didefinisikan di luar fungsi tersebut.
    - Gunakanlah key berupa STRING, dalam huruf besar dan rujuklah ke indeks 1 (karena indeks 0 adalah tipenya).
    - Kita kemudian tambahkan menu item yang dibuat ke menu_item_test.
- Diwarnai merah: untuk setiap elemen dalam menu_item_test, kita:
    - Cek apakah indeksnya adalah ganjil (argumen kedua pada fungsi yang digunakan dalam each adalah indeks untuk elemen; operator % adalah sisa bagi).

    - Apabila ya, kita hanya perlu mengubah nilai ARRAY (dengan fungsi set) untuk elemen dengan indeks 2 (aktif/tidak, BOOLEAN) menjadi false.
- Diwarnai oranye: sudah dibahas sebelumnya, tapi, kadang-kadang penulis juga melupakan ini. Setiap kali selesai melakukan pengubahan pada ARRAY menu, agar berdampak, kembalilah panggil fungsi menubar.

Barangkali masih tetap semangat, Anda mungkin bertanya: bagaimana kalau, alih-alih mendisable, menu-menu item tersebut tidak ingin ditampilkan? Cara yang paling mudah adalah dengan mengganti baris berikut:

```
set(e, 2, false)
```

Menjadi:

```
set(menu_item_test, i, null)
```

Lalu, bagaimana kalau ingin menggantinya menjadi separator? Gantilah null dengan [] seperti contoh berikut:

```
set(menu_item_test, i, [])
```

Tidak lagi ingin menampilkan menubar di tengah program berjalan? Anda pasti sudah tahu jawabannya: panggil fungsi menubar dengan argumen berupa ARRAY kosong, atau [].

Sebelum kita melanjutkan ke contoh berikut dan menutup pembahasan menu bar (*akhirnya!*), ingatlah bahwa pada dasarnya, kita bekerja dengan ARRAY ketika memanggil fungsi menubar. Dengan demikian, operasi yang valid pada ARRAY, selama sesuai aturan argumen menubar, akan berdampak pada menu-menu yang ditampilkan.

*Halaman ini sengaja dikosongkan*

# Contoh 3: Menampilkan informasi pada status bar


Ada kalanya, kita perlu menampilkan informasi tertentu di frame, dimana informasi tersebut tidak membutuhkan perhatian khusus atau aksi dari pengguna, seperti halnya message atau konfirmasi yang ditampilkan pada sebuah dialog. Program pun tetap akan berjalan dengan atau tanpa pengguna memperhatikan informasi tersebut.

Karena sifatnya, informasi tersebut dapat ditempatkan di bagian yang tidak mencolok. Sebagai contoh, berupa sebaris teks, di sisi bawah frame, yang umum kita kenal sebagai status bar.

Sesuai namanya, informasi yang tampil umumnya berupa status. Apabila kita membangun file manager, kita mungkin akan menampilkan berapa jumlah file di dalam direktori aktif. Pada aplikasi yang terhubung ke server, kita mungkin menampilkan status koneksi atau nama user. Sementara, untuk aplikasi pendukung bisnis, informasi yang turut tampil dapat berupa tanggal/waktu dan shortcut keyboard.

Karena umumnya terdapat beberapa informasi atau status yang tampil, status bar dapat dibagi menjadi beberapa bagian. Kadang kala, kita mungkin ingin mendisable bagian tertentu, yang memvisualisasikan kondisi non aktif tertentu.

Bagi Anda yang pernah menggunakan bahasa Singkong dalam membangun aplikasi GUI, Anda mungkin bertanya, bukankah kita cukup menggunakan fungsi add_s untuk menampilkan ARRAY dari COMPONENT label?

Penulis juga sepakat dengan Anda. Setidaknya, inilah yang awalnya pengembang Singkong gunakan ketika perlu menampilkan status. Tapi, dalam perkembangan, muncul beberapa kondisi atau alasan yang menjadikan solusi ini tidak lagi nyaman digunakan:

- Kita perlu membuat COMPONENT dan menambahkannya secara manual. Walaupun memang masih banyak hal yang harus dikerjakan secara manual di Singkong, ini terlalu merepotkan untuk sekedar status. Ingatlah bahwa status kadang berubah, sehingga kita perlu menggunakan variabel yang merujuk ke COMPONENT tersebut, supaya kita bisa mengkonfigurasi dengan fungsi config (key: text) setiap kali isinya akan diubah.
- Untuk mendisable COMPONENT tersebut, kita juga perlu mengkonfigurasi dengan config (key: enabled). Mungkin ada kesan dibuat-buat supaya daftar alasan ini menjadi lebih panjang.
- Dan, apabila Anda merasa pengembang Singkong berlebihan untuk kedua alasan sebelumnya, Anda mungkin akan sepakat dengan ini: bagaimana kalau kita sudah punya sejumlah COMPONENT yang akan ditempatkan di sisi bawah frame dengan add_s? Benar kan? Kita tahu bahwa kita tidak bisa menggunakan add_s untuk ARRAY sejumlah COMPONENT, lebih dari sekali, karena, yang terakhir yang akan berdampak.

Mari kita lihat kode program berikut:

```
reset()
var a = component("label", "Label 1")
var b = component("label", "Label 2")
var c = component("label", "Label 3")
var d = component("label", "Label 4")

config(b, "enabled", false)

add_s([a, b])
show()
```

Ketika dijalankan, Label 1 dan Label 2 akan tampil di sisi bawah frame, dengan Label 2 dalam kondisi disabled. Kita tahu, apabila menambahkan Label 3 dan Label 4 dengan add_s, maka yang sebelumnya akan tidak lagi tampil. Perhatikanlah kode berikut:

```
reset()
var a = component("label", "Label 1")
var b = component("label", "Label 2")
var c = component("label", "Label 3")
var d = component("label", "Label 4")

config(b, "enabled", false)
```

```
add_s([a, b])
add_s([c, d])
show()
```

Untuk alasan terakhir sebelumnya, Anda mungkin berpendapat, kenapa kita tidak menambahkan semua COMPONENT yang diperlukan (termasuk yang tadinya ingin ditempatkan di sisi bawah) ke dalam sebuah grid atau panel, yang kemudian ditambahkan ke frame dengan fungsi add? Biarkanlah add_s hanya untuk status? Atau, sekalian saja gunakan grid, termasuk untuk status bar.

Pendapat Anda tersebut sepenuhnya valid, dan dalam user interface program yang kompleks, cara tersebut mungkin diperlukan. Bahkan, kita akan mencontohkan cara ini, di akhir bab ini.

Tapi, sebelum itu, mari kita gunakan cara yang mudah dulu, dengan memanggil fungsi statusbar. Perhatikanlah beberapa catatan berikut:

- Di Singkong, status bar dibagi menjadi 8 bagian, dari indeks 0 sampai 7.
- Masing-masing bagian tersebut dapat di-disable.
- Masing-masing bagian hanya dapat berisikan informasi teks.
- Fungsi statusbar membutuhkan 3 argumen: bagian (NUMBER 0 sampai 7), teks (STRING), dan apakah enabled (BOOLEAN).

Perhatikanlah kode program berikut:

```
reset()
statusbar(7, singkong()["name"], true)
```

```
show()
```

Kembalian dari pemanggilan fungsi singkong akan mengembalikan HASH informasi seputar interpreter bahasa pemrograman Singkong. Dengan key “name”, valuenya adalah “Singkong”. Pemanggilan statusbar seperti contoh akan menyebabkan teks Singkong ditempatkan di bagian ke-8.

Kode program berikut akan menampilkan teks “Status: “ diikuti indeks bagian, dan mendisable bagian dengan indeks ganjil:

```
reset()
each(range(0,8), fn(e, i) {
    statusbar(e, "Status: " + e, i%2 == 0)
})
show()
```

Sebenarnya, pembahasan tentang status bar telah selesai sampai di sini. Tapi, ini baru bab awal, dan penulis ingin sedikit menabung pembahasan demi mencukupkan jumlah halaman (*apakah ini benar diperlukan?*). Sehingga, mari kita bahas apa yang dapat kita lakukan ketika ingin menampilkan status yang bukan berupa teks dan jelas tidak bisa dilakukan dengan fungsi statusbar.

Misal, kita ingin menampilkan progress bar dengan fungsi statusbar:

```
reset()
var p = component("progress", "")
config(p, "contents", 50)
statusbar(7, p, true)
```

```
show()
```

Kode tersebut tidak dapat dijalankan dengan pesan kesalahan sebagai berikut:

```
ERROR: [line: 4] type mismatch: got=NUMBER,
COMPONENT, and BOOLEAN, want=NUMBER, STRING, and
BOOLEAN
```

Jadi, kita perlu mencoba cara lain. Bagaimana kalau dengan fungsi add_s yang mudah?

```
reset()

var a = component("label", "Label 1")

var b = component("label", "Label 2")

var c = component("label", "")

var d = component("label", "")

var p = component("progress", "")

config(p, "contents", 50)

add_s([a, b, c, d, p])

show()
```

Dengan fungsi add_s, semua COMPONENT tersebut akan ditampilkan di sisi bawah frame, dengan progress bar di paling kanan. Ini artinya, COMPONENT utama dari program GUI Anda perlu ditempatkan di tengah dengan add. Dapat berupa grid, panel, ataupun COMPONENT lain.

Kode program tersebut menutup pembahasan kita tentang status bar.

# Contoh 4: Menangani event mouse, keyboard, frame, dan focus

Penanganan event, bersama pembahasan tentang proses yang berjalan lama (dengan thread), barangkali akan menjadi yang terpanjang dalam buku ini. Tujuannya jelas bukan sekedar menambah halaman (supaya buku ini cepat selesai penulisannya).

Kita akan mulai dengan analogi pertama (*memangnya, ada berapa?*). Pernahkah Anda begitu menantikan sesuatu, sehingga secara berkala, memeriksa apakah yang dinantikan tersebut sudah tiba? Sebagai contoh, Anda menyukai singkong goreng dan sedang menanti penjual gorengan yang berkeliling setiap sore. Setiap beberapa menit, Anda melihat ke luar jendela. Berkali-kali melihat, rupanya penjual gorengan yang dinanti tidak juga tiba. Setelah cukup lama, barulah tiba. Untungnya, singkong goreng masih tersedia.

Sekarang, kita lanjut ke yang kedua. Di tengah penantian, Anda mungkin menjadi kelelahan dan bertekad tidak mengecek secara berkala lagi. Sebagai gantinya, Anda berpesan agar begitu tiba di depan rumah, tolong beri sinyal yang jelas bahwa singkong goreng telah tersedia. Anda kemudian cukup membuka pintu dan memesan.

Mana yang lebih nyaman menurut Anda? Sebagian besar dari Anda mungkin akan menjawab yang terakhir. Dan, inilah yang akan kita bahas di dalam bab ini (dalam tahap tertentu perkembangan teknologi komputer, dan/atau di tingkatan tertentu pustaka atau teknologi yang digunakan saat ini, cara pada analogi pertama mungkin digunakan walau dengan mekanisme yang lebih kompleks).

Mari kita lanjut. Singkong goreng tersedia dan sinyal diberikan (misal dengan mengetuk pintu) mirip dengan COMPONENT button ditekan. Atau ketika combobox diubah isinya. Atau, ketika frame diubah ukurannya. Atau ketika mouse diklik. Dan seterusnya. Program tidak

terus menerus menanti kapan pengguna program menekan sebuah button. Program menyiapkan fungsi tertentu yang akan dipanggil ketika button ditekan. Apa yang terjadi di balik layar tidak perlu dikhawatirkan oleh pengembang program.

Di Singkong, sejumlah COMPONENT memiliki event default. Ini artinya, event yang akan ditangani dengan pendaftaran lewat fungsi event. Default dalam hal ini merujuk pada asumsi fungsi utama sebuah COMPONENT mungkin digunakan, sebagaimana dirinci pada tabel berikut.

| **COMPONENT** | **Event** |
|---|---|
| Button | Ketika ditekan |
| Combobox | Ketika item terpilih berubah |
| Checkbox | Ketika di-check atau di-uncheck |
| Radio | Ketika di-select atau di-deselect |
| Tab | Ketika item terpilih berubah |
| Table | Ketika item terpilih berubah |
| Date | Ketika isinya berubah |
| Edit | Ketika isinya berubah |
| Password | Ketika isinya berubah |
| Spin | Ketika isinya berubah |
| Text | Ketika isinya berubah |

Dalam menangani event default, kita tahu bahwa kita akan selalu cukup menggunakan fungsi event, yang membutuhkan dua argumen: COMPONENT dan FUNCTION. Yang terakhir haruslah merupakan fungsi tanpa argumen.

Anda mungkin telah berkali-kali melihat contoh kode serupa:

```
reset()

var b = component("button", "Tombol")
```

```
add_s(b)
show()


event(b, fn() {
    message("Tombol ditekan")
})
```

Penjelasan:

- Diwarnai hijau: ini sebenarnya tidak lagi perlu dijelaskan. Tapi, penulis ingin memberi penekanan bahwa kita membuat sebuah COMPONENT button, dan mengassign ke variabel b. Ini adalah opsional, tapi dalam event, tampaknya tidak ada cara lain. Mari bahas ini lebih lanjut setelahnya.
- Diwarnai biru: ini adalah pemanggilan fungsi event. Fungsi membutuhkan dua argumen.
- Diwarnai oranye: argumen pertama merujuk pada COMPONENT button b.
- Diwarnai merah: argumen kedua adalah fungsi tanpa argumen, yang akan dipanggil, manakala event default terjadi.

Anda lihat? Tidak ada yang baru apabila Anda telah membaca buku-buku Singkong lainnya. Tapi, mari kita tambahkan sedikit bumbu:

- Bumbu manis: baris yang diwarnai hijau adalah opsional ketika Anda tidak ingin memroses event. Kita dapat tuliskan ulang sebagai berikut:

```
reset()
add_s(component("button", "Tombol"))
show()
```

- Bumbu pedas: baris-baris yang diwarnai merah mungkin dapat diberikan nama tersendiri. Mari kita tuliskan ulang sebagai berikut.

```
reset()
var b = component("button", "Tombol")
add_s(b)
show()

var f = fn() {
    message("Tombol ditekan")
}
event(b, f)
```

Bumbu yang manis sudah jelas. Ketika kita perlu tambahkan ke frame dan menangani event default, maka kita perlu merujuk ke sebuah variabel.

Bagaimana dengan bumbu yang pedas? Anda bisa berpendapat bahwa kode program menjadi lebih nyaman dibaca. Tapi ada manfaat lain. Yaitu ketika sejumlah COMPONENT ingin menggunakan fungsi yang sama untuk menangani event default. Perhatikanlah contoh kode berikut:

```
reset()
var b = component("button", "Tombol")
var c = component("combobox", "A,B,C")
var d = component("date", "")
add_s([b, c, d])
show()

var f = fn() {
    message("Event ditangani")
}
event(b, f)
event(c, f)
event(d, f)
```

Daripada dituliskan tiga kali padahal isinya sama, yaitu masing-masing untuk setiap COMPONENT, tentunya dengan menuliskan fungsi sekali dan mengassign ke variabel tersendiri akan lebih membantu.

Sekarang, Anda mungkin ingin menanyakan sesuatu dan penulis sudah khawatir bahwa jawabannya akan mengecewakan Anda. Apakah gerangan hal tersebut? Tak lain dan tak bukan adalah: bagaimana fungsi f tersebut tahu COMPONENT mana (misal dari b, c, dan d) yang memanggil dirinya? Anda sudah tahu jawabannya dan mungkin kecewa: hal tersebut tidak memungkinkan.

Kenapa? Karena tidak ada cara bagi fungsi f untuk mengetahui hal tersebut. Fungsi f tidak menerima argumen dan Singkong (setidaknya untuk saat ini) tidak menyediakan mekanisme lain yang memungkinkan hal tersebut terjadi.

Kenapa? Anda mungkin bertanya lagi. Jawabannya adalah hal tersebut dikesampingkan ketika GUI dan penanganan event dirancang, dengan harapan bahwa Singkong akan lebih sederhana.

Kenapa? Lagi-lagi Anda bertanya. Sama seperti hadirnya event default (sehingga cukup dengan fungsi event) dan setiap program GUI di Singkong hanya memiliki satu frame (apabila diperlukan, bacalah juga buku gratis: *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI)*, salah satu tujuan dari Singkong adalah GUI dapat dibuat dengan semudah dan sesederhana mungkin.

Kenapa? Semoga ini pertanyaan terakhir (*dan bukannya untuk menambah satu paragraf hanya untuk mempertebal buku ini*). Pada akhir tahun 2019, ketika Singkong pertama kali dirilis ke publik, GUI bahkan tidak tersedia. Adanya GUI, tak lama kemudian, menjadikan Singkong lebih berguna, bagi siapapun untuk dapat menulis program dengan sesederhana mungkin.

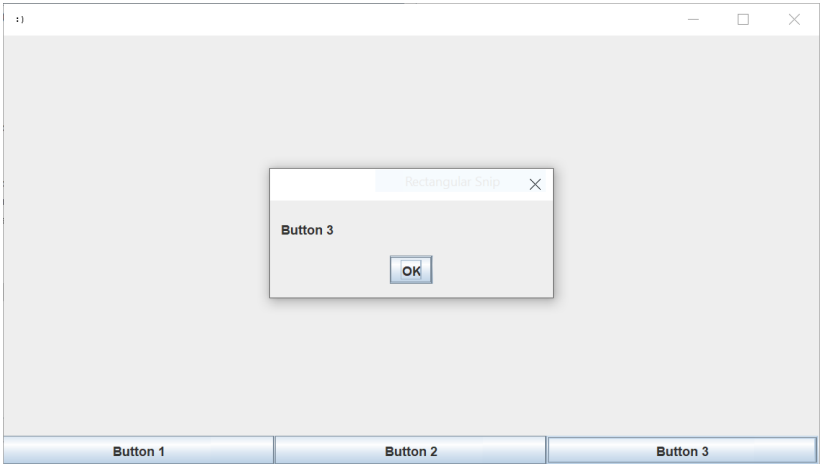
Lalu, bagaimana kalau kita perlu membuat 10 button dengan 10 fungsi yang hanya berbeda sedikit? Misal ketika button 1 diklik, menampilkan “button 1”. Ketika button 2 diklik, menampilkan “button 2”. Dan seterusnya. Kita tidak dapat menggunakan satu fungsi yang sama dan memeriksa COMPONENT mana yang ditangani. Membuat sejumlah fungsi yang masing-masing hanya berbeda sedikit juga rasanya berlebihan. Solusinya, walau tidak ideal, bisa menggunakan perulangan. Perhatikanlah contoh kode berikut:

```
reset()
var b1 = component("button", "Button 1")
var b2 = component("button", "Button 2")
var b3 = component("button", "Button 3")
var bb = [b1, b2, b3]

add_s(bb)
show()

each(bb, fn(e, i) {
    var m = component_info(e)[6]
    event(e, fn() {
        message(m)
    })
})
```

Penjelasan:

- Diwarnai hijau: kita memiliki tiga button, masing-masing b1, b2, dan b3. Kita tambahkan ke ARRAY. Tujuan utamanya adalah untuk digunakan di perulangan.
- Diwarnai biru: untuk setiap elemen dalam bb, fungsi dengan dua argumen (e dan i) dipanggil. Kita membahas ini di buku sebelumnya: e adalah elemen itu sendiri dan i adalah indeksnya (dari 0).
- Diwarnai merah: fungsi component_info akan mengembalikan informasi sebuah COMPONENT. Indeks ke-6 adalah namanya.

- Diwarnai oranye: variabel e adalah setiap COMPONENT button dalam ARRAY. Jadi, kita membuat event handler secara dinamis. Di dalamnya, kita menampilkan sebuah message, yang mana, isinya, adalah nama COMPONENT. Akan berbeda-beda untuk setiap COMPONENT dalam contoh ini.




Catatan: Kita tidak perlu mengulang pembahasan untuk COMPONENT lain yang juga memiliki event default. Caranya akan sama.

**Mouse**

Seperti disebutkan di awal, bab ini akan panjang. Kita bahkan belum seperempat jalan. Jadi, mari istirahat dulu.

Kalau dipikir-pikir, sebuah button misalnya, dapat ditekan dengan bantuan tombol pada keyboard ataupun klik dengan mouse. Demikian juga dengan combobox. Lalu, untuk apa kita membahas mouse di sini?

Baiklah. Istirahat kita sudah selesai dan kita perlu melanjutkan perjalanan.

Bagaimana kalau kita perlu menangani ketika pointer mouse masuk atau keluar sebuah COMPONENT? Atau ketika tombol mouse ditekan, dilepas, atau diklik ketika berada pada sebuah COMPONENT? Atau ketika drag dilakukan? Itulah contoh alasan kenapa kita mungkin perlu menangani event mouse secara khusus. Alasan lain adalah misal ketika kita perlu membuat aplikasi paint sederhana dimana pengguna dapat menggambar dengan mouse (lebih lanjut, dapat dibaca pada dokumentasi Singkong). Alasan lain lagi adalah misal ketika kita baru memroses tindakan ketika dilakukan klik sebanyak 5 kali pada sebuah tombol.

Sebagaimana penanganan event default, penanganan event mouse pada COMPONENT cukup dengan memanggil sebuah fungsi, yaitu event_mouse. Fungsi ini juga menerima dua argumen, yaitu COMPONENT di mana event mouse terjadi, dan sebuah FUNCTION yang akan dipanggil manakala event mouse pada COMPONENT tersebut perlu ditangani. Bedanya, apabila pada event, fungsi tidak boleh menerima argumen, fungsi ini harus menerima satu argumen.

Semoga bukan satu argumen yang terasa seperti sepuluh, seperti ketika kita membahas menu bar. Tidak, tidak akan seperti itu. Hanya terasa seperti empat. Ya ampun!

Lebih spesifiknya, sebuah argumen yang harus berupa ARRAY, yang terdiri dari empat elemen:

- tipe, bertipe STRING, salah satu dari: CLICKED, ENTERED, EXITED, PRESSED, RELEASED, MOVED, DRAGGED. Ini akan membedakan tindakan mouse apa yang mungkin kita perlu tahu.
- x: posisi x ketika event ini terjadi.
- y: posisi y ketika event ini terjadi.
- jumlah klik

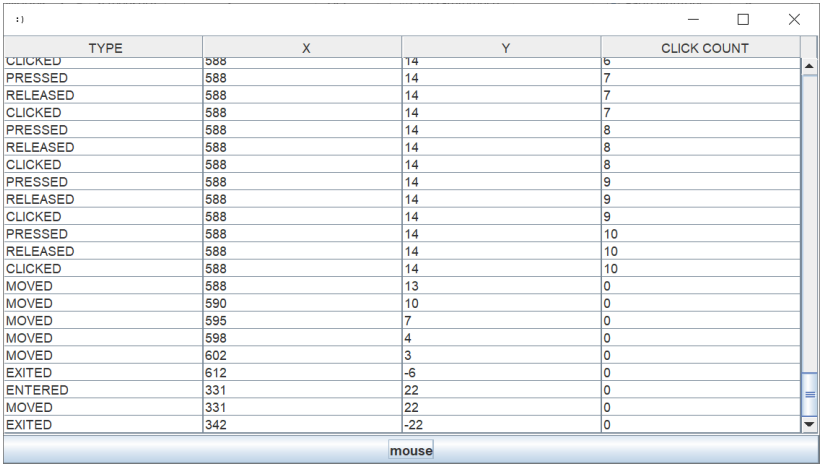
Mari kita lihat contohnya:

```
reset()

var b = component("button", "mouse")

var t = component("table", "TYPE, X, Y, CLICK COUNT",
true)


event_mouse(b, fn(x) {

    table_add(t, [x])

    var c = len(get(t, "contents"))

    table_scroll(t, c-1)

})


add(t)

add_s(b)


show()
```

Contoh ini, walaupun singkat, tapi mendemonstrasikan hal yang keren. Benar begitu? Mari kita bahas:

- Diwarnai hijau: kita menyiapkan sebuah button dimana event mouse akan terjadi pada COMPONENT tersebut. Kita akan tangani dan tampilkan pada table. Ini yang kita sebut keren sebelumnya. Karena, kita bisa melihat secara mendetil apa yang terjadi (*begitu saja sudah keren?*).

- Diwarnai biru: kita menggunakan fungsi event_mouse dan argumen kedua fungsi adalah sebuah fungsi dengan satu argumen. Argumen x ini akan bertipe ARRAY dengan empat elemen sebagaimana kita bahas sebelumnya.
- Diwarnai merah: untuk menambahkan ke table, kita gunakan fungsi table_add, yang akan menerima ARRAY dari ARRAY. Walau x adalah sebuah ARRAY, kita perlu tambahkan ke sebuah ARRAY ([x]).
- Diwarnai oranye: baris-baris ini sebenarnya cukup merepotkan :) Pertama, kita dapatkan isi sebuah table (dengan fungsi get untuk "contents"). Ini mungkin bisa ratusan baris, ketika Anda menggunakan mouse cukup lama pada button. Setelah itu, kita dapatkan jumlahnya dengan fungsi len. Seolah belum cukup merepotkan, kita scroll table ke paling bawah (jumlah dikurangi satu, karena mulai dari 0). Jangan lupa, ini pun dipanggil bahkan ketika kita menggerakkan mouse sedikit saja.




Ada kemungkinan, dari ARRAY yang dilewatkan pada fungsi yang dipanggil, kita hanya akan menangani aksi tertentu. Misal hanya

ketika mouse memasuki atau keluar dari button. Perhatikanlah contoh berikut:

```
reset()
var b = component("button", "mouse")
add_s(b)


event_mouse(b, fn(x) {
    var t = x[0]
    if (t == "ENTERED") {
        message("Mouse masuk")
    }
    if (t == "EXITED") {
        message("Mouse keluar")
    }
})


show()
```

Kita tahu bahwa elemen pertama dari x adalah salah satu dari: CLICKED, ENTERED, EXITED, PRESSED, RELEASED, MOVED, DRAGGED. Dengan demikian, kita bisa tangani sesuai keperluan.

Bagaimana kalau kita akan tangani hanya kalau button diklik 5 kali (dianggap sebagai event klik, bukan klik sekali sebanyak lima kali)? Mari kita lihat contoh penutup ini:

```
reset()
var b = component("button", "mouse")
add_s(b)

event_mouse(b, fn(x) {
    var t = x[0]
    var c = x[3]
    if ((t == "CLICKED") & (c == 5)) {
        message("Anda telah klik 5 kali")
    }
})

show()
```

**Mouse pada frame**

Bagaimana, seru bukan? Tak terasa, kita telah mencapai seperempat perjalanan lewat sedikit :) Kita tidak akan istirahat karena penulis berjanji, bagian ini akan singkat.

Sebelumnya, kita menangani event mouse yang terjadi pada COMPONENT. Sekarang, kita akan tangani ketika event mouse terjadi pada frame. Fungsi yang digunakan adalah event_mouse_frame, yang hanya akan menerima satu argumen, yaitu FUNCTION, yang menerima satu argumen sebagaimana sebelumnya pada event_mouse.

Mari kita lihat contoh berikut:

```
reset()

var t = component("table", "TYPE, X, Y, CLICK
COUNT", true)

add_w(t)

show()


event_mouse_frame(fn(x) {

    table_add(t, [x])

    var c = len(get(t, "contents"))

    table_scroll(t, c-1)

})
```

Di program tersebut, table ditambahkan di sisi kiri frame. Event mouse frame dapat dilakukan di sisi kanan yang berupa area kosong.

**Keyboard**

Setelah mouse, kita membahas event pada keyboard, dan dengan demikian, kita telah mencapai setengah perjalanan bab ini. Istirahat dulu?

Tampaknya perlu. Kita hanya akan membahas penggunaan satu fungsi: event_keyboard_frame, yang hanya membutuhkan satu argumen berupa FUNCTION. Fungsi dimaksud membutuhkan satu argumen berupa ARRAY. Tapi serasa 12.

Astaga!

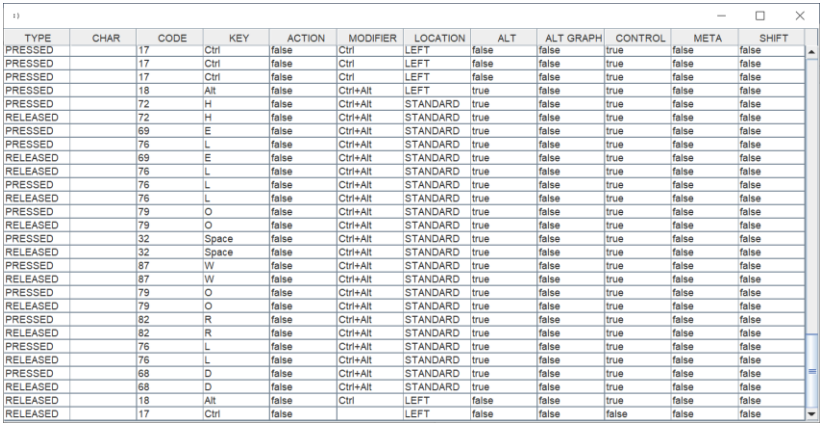
Tapi tidak apa, karena penulis yakin, setelah event mouse pada COMPONENT dan frame, kita sudah terbiasa. Elemen ARRAY dimaksud, yang berjumlah 12 tersebut, dapat dirinci sebagai berikut:

- Tipe. Berupa STRING, salah satu dari: TYPED, PRESSED, RELEASED.
- Character. Berupa STRING, ketika tersedia.
- Code. Berupa NUMBER.
- Teks key. Berupa STRING.
- Apakah berupa action.
- Teks modifier.
- Lokasi key. Berupa STRING, salah satu dari: STANDARD, LEFT, RIGHT, NUMPAD, UNKNOWN.
- Apakah tombol alt down.
- Apakah tombol alt graph down.
- Apakah tombol control down.
- Apakah tombol meta down.
- Apakah tombol shift down.

Berikut adalah contoh kode programnya:

```
reset()

var t = component("table", "TYPE, CHAR, CODE, KEY,
ACTION, MODIFIER, LOCATION, ALT, ALT GRAPH,
CONTROL, META, SHIFT", true)

add(t)

show()


event_keyboard_frame(fn(x) {

    table_add(t, [x])

    var c = len(get(t, "contents"))
```

```
    table_scroll(t, c-1)
})
```



| TYPE | CHAR | CODE | KEY | ACTION | MODIFIER | LOCATION | ALT | ALT GRAPH | CONTROL | META | SHIFT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PRESSED | | 17 | Ctrl | false | Ctrl | LEFT | false | false | true | false | false |
| PRESSED | | 17 | Ctrl | false | Ctrl | LEFT | false | false | true | false | false |
| PRESSED | | 17 | Ctrl | false | Ctrl | LEFT | false | false | true | false | false |
| PRESSED | | 18 | Alt | false | Ctrl+Alt | LEFT | true | false | true | false | false |
| PRESSED | | 72 | H | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 72 | H | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 69 | E | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 76 | L | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 69 | E | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 76 | L | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 76 | L | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 76 | L | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 79 | O | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 79 | O | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 32 | Space | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 32 | Space | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 87 | W | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 87 | W | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 79 | O | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 79 | O | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 82 | R | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 82 | R | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 76 | L | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 76 | L | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| PRESSED | | 68 | D | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 68 | D | false | Ctrl+Alt | STANDARD | true | false | true | false | false |
| RELEASED | | 18 | Alt | false | Ctrl | LEFT | false | false | true | false | false |
| RELEASED | | 17 | Ctrl | false | | LEFT | false | false | false | false | false |

Di screenshot tersebut, penekanan tombol control dan alt dilakukan, sambil menekan karakter-karakter: hello world.

**Frame**

Setengah perjalanan lewat sedikit, kita akan membahas event pada frame itu sendiri. Walaupun serupa dengan pembahasan-pembahasan sebelumnya, ada sedikit catatan tambahan.

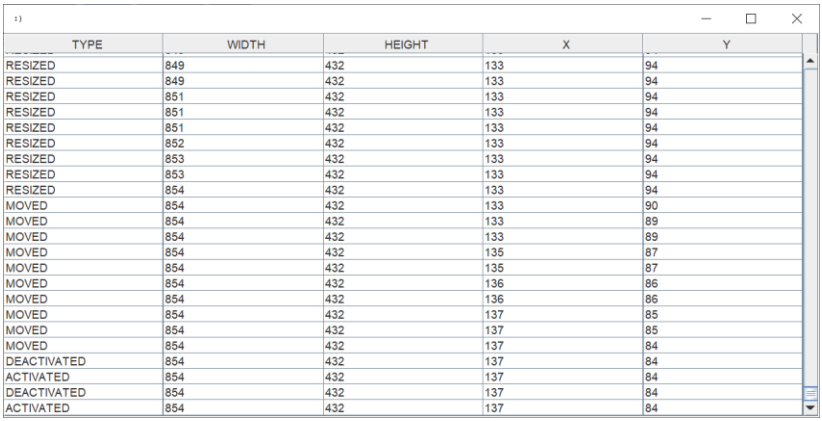
Fungsi yang digunakan adalah event_frame, yang menerima satu argumen berupa FUNCTION. Sama seperti contoh sebelumnya, fungsi tersebut menerima satu argumen berupa ARRAY, yang kali ini berisi lima elemen, sebagai berikut:

- Tipe. Berupa STRING, salah satu dari: ACTIVATED, CLOSED, DEACTIVATED, DEICONIFIED, GAINEDFOCUS, ICONIFIED, LOSTFOCUS, OPENED, STATECHANGED, RESIZED, MOVED.

- Lebar frame.
- Tinggi frame.
- x: posisi x frame ketika event ini terjadi.
- y: posisi y frame ketika event ini terjadi.

Berikut adalah contoh kode programnya:

```
reset()

var t = component("table", " TYPE, WIDTH, HEIGHT,
X, Y", true)

add(t)

show()


event_frame(fn(x) {

    table_add(t, [x])

    var c = len(get(t, "contents"))

    table_scroll(t, c-1)

})
```



| TYPE | WIDTH | HEIGHT | X | Y |
|---|---|---|---|---|
| RESIZED | 849 | 432 | 133 | 94 |
| RESIZED | 849 | 432 | 133 | 94 |
| RESIZED | 851 | 432 | 133 | 94 |
| RESIZED | 851 | 432 | 133 | 94 |
| RESIZED | 851 | 432 | 133 | 94 |
| RESIZED | 852 | 432 | 133 | 94 |
| RESIZED | 853 | 432 | 133 | 94 |
| RESIZED | 853 | 432 | 133 | 94 |
| RESIZED | 854 | 432 | 133 | 94 |
| MOVED | 854 | 432 | 133 | 90 |
| MOVED | 854 | 432 | 133 | 89 |
| MOVED | 854 | 432 | 133 | 89 |
| MOVED | 854 | 432 | 135 | 87 |
| MOVED | 854 | 432 | 135 | 87 |
| MOVED | 854 | 432 | 136 | 86 |
| MOVED | 854 | 432 | 136 | 86 |
| MOVED | 854 | 432 | 137 | 85 |
| MOVED | 854 | 432 | 137 | 85 |
| MOVED | 854 | 432 | 137 | 84 |
| DEACTIVATED | 854 | 432 | 137 | 84 |
| ACTIVATED | 854 | 432 | 137 | 84 |
| DEACTIVATED | 854 | 432 | 137 | 84 |
| ACTIVATED | 854 | 432 | 137 | 84 |

Pada screenshot tersebut, pengguna program mengubah ukuran, memindahkan, mendeaktifasi (pengguna pindah ke program lain), dan kembali ke frame.

Apabila kita menambahkan konfirmasi tutup frame berikut:

```
closing("Are you sure you want to quit this
application?",  "Please confirm")
```

Perhatikanlah bahwa ketika dialog konfirmasi ditampilkan, frame akan dideaktifasi dan ketika dialog ditutup tidak keluar dari aplikasi, frame kembali diaktifasi.

**Fokus**

Akhirnya kita mencapai tiga perempat perjalanan di bab ini.

Kita akan membahas penanganan event fokus untuk COMPONENT. Fungsi yang digunakan adalah event_focus, yang akan menerima dua argumen. Yang pertama, tentu saja adalah COMPONENT, yang kedua, sudah pasti, FUNCTION. Fungsi yang dimaksud, seperti sebelumnya, menerima sebuah ARRAY, yang terdiri tiga elemen. Jadi, semakin sedikit :). Perinciannya adalah:

- Tipe. Berupa STRING, salah satu dari: GAINED, LOST.
- x: posisi x
- y: posisi y

Berikut adalah contoh programnya:

```
reset()
```

```
var i = component("label", "Focus event: text")
var t = component("table", "TYPE, X, Y", true)
var x = component("text", "Click")

event_focus(x, fn(x) {
    table_add(t, [x])
})

add_n(i)
add(t)
add_s(x)

show()
```

Ketika program dijalankan, cobalah klik pada COMPONENT text di sisi bawah frame untuk memberikan fokus (GAINED) pada COMPONENT tersebut. Kemudian, pindahlah fokus ke COMPONENT lain, dimana COMPONENT text akan kehilangan (LOST) fokus.

Bab ini akan kita tutup dengan satu utang contoh. Yaitu, bagaimana kita memanfaatkan penanganan event fokus untuk menampilkan popup pada COMPONENT tertentu.

Mari kita istirahat sejenak sebelum membahas popup di bab berikut. Pembahasannya mungkin akan menyenangkan :)

*Halaman ini sengaja dikosongkan*

## Contoh 5: Bekerja dengan popup

Di buku gratis *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI*, kita menggunakan panel dialog ketika menampilkan dialog untuk menambahkan pesanan. Menggunakan dialog pada dasarnya akan lebih mudah dan umum, untuk kepentingan tersebut: sebuah dialog dengan title bar, tombol OK dan Cancel, serta isi dialog yang dapat terdiri dari sejumlah COMPONENT, yang ditambahkan ke panel ataupun grid.

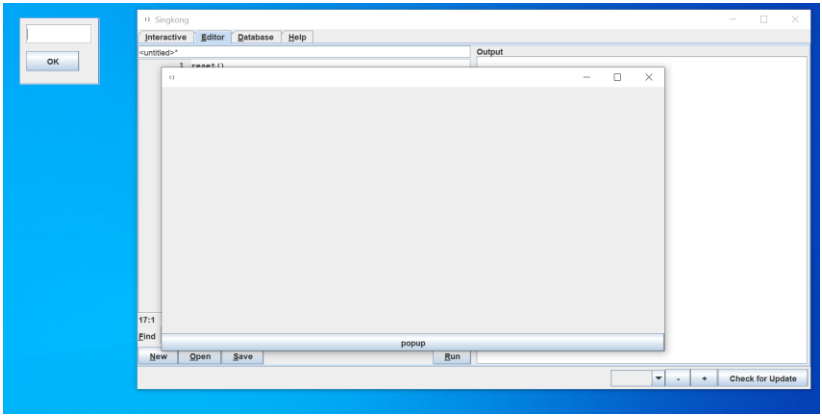
Walau nyaman, beberapa contoh hal berikut tidak dimungkinkan dengan dialog di Singkong:

- Kita ingin tampilkan di posisi tertentu, dengan ukuran tertentu. Posisi tertentu yang dimaksud bisa relatif terhadap layar, frame, ataupun COMPONENT.
- Kita tidak ingin menampilkan title bar.
- Kita tidak membutuhkan tombol OK dan Cancel.
- Kita ingin menampilkan banyak sekaligus.
- Kita ingin tetap tampil dan pengguna bisa tetap bekerja pada frame.

Sebaliknya, semua hal tersebut dimungkinkan dengan popup. Mari kita mulai dari yang sederhana dulu, dengan fokus pada isi popupnya. Seperti halnya pada dialog, kita dapat menggunakan panel ataupun grid. Kita akan mulai dengan panel:

```
reset()

var t = component("text", "")

var b = component("button", "OK")

var p = component("panel", "")

panel_add(p, t, 10, 10, 100, 30)

panel_add(p, b, 10, 50, 80, 30)
```

```
var r = [-1]


var c = component("button", "popup")
event(c, fn() {
    var x = popup_show(p, 120, 100, 50, 50, false)
    set(r, 0, x)
})
add_s(c)
show()
```




Ketika button popup ditekan, sebuah popup tampil. Sementara pengguna bisa berinteraksi dengan COMPONENT di dalam popup tersebut, pengguna juga tetap dapat bekerja di frame. Tidak ada title bar pada popup. Tidak ada button OK dan Cancel.

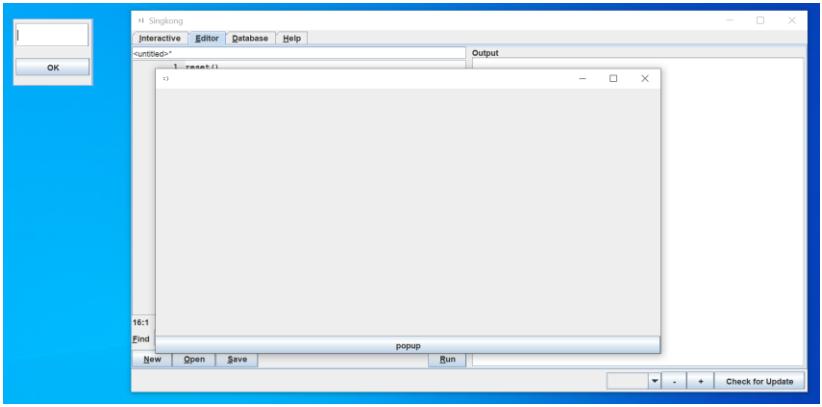
Penjelasan:

- Diwarnai hijau: kita membuat COMPONENT text dan button yang akan ditampilkan dalam popup. Semua COMPONENT tersebut ditambahkan ke sebuah panel p.
- Diwarnai biru: ketika button ditekan, tampilkan popup dengan fungsi popup_show. Fungsi popup_show dalam hal ini menerima argumen:
    - Tampilkan panel atau grid
    - Lebar popup
    - Tinggi popup
    - Posisi x
    - Posisi y
    - Apakah x dan y relatif terhadap frame
- Dalam hal ini, posisi popup tidak relatif terhadap frame.
- Diwarnai merah: mari kita bahas ini nanti, ketika membahas bagaimana menyembunyikan popup (*penjelasan seperti apa pula ini!*).

Seru bukan? Walau, tentu saja, sebagaimana halnya pada panel, kita perlu tentukan ukuran dan posisi COMPONENT dalam popup. Sebagai alternatif, mari kita gunakan grid seperti contoh berikut:

```
reset()

var t = component("text", "")

var b = component("button", "OK")

var g = component("grid", "")

grid_add(g, t, 0, 0, 1, 1, 1, 1, 3, 0)

grid_add(g, b, 0, 1, 1, 1, 1, 1, 1, 0)


var r = [-1]
```

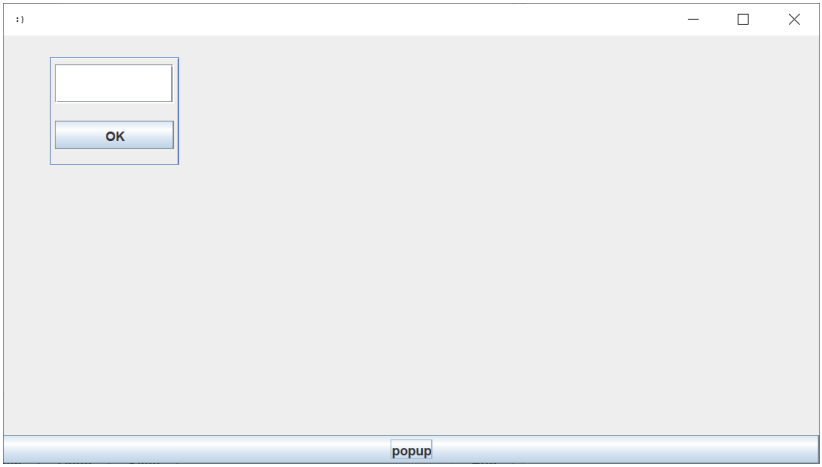
```
var c = component("button", "popup")
event(c, fn() {
    var x = popup_show(g, 120, 100, 50, 50, false)
    set(r, 0, x)
})
add_s(c)
show()
```




Dalam kondisi yang lebih umum, Anda mungkin ingin menampilkan popup relatif terhadap posisi frame. Mari kita ubah sedikit contoh sebelumnya:

```
reset()
var t = component("text", "")
var b = component("button", "OK")
var g = component("grid", "")
grid_add(g, t, 0, 0, 1, 1, 1, 1, 3, 0)
grid_add(g, b, 0, 1, 1, 1, 1, 1, 1, 0)
```

```
var r = [-1]


var c = component("button", "popup")
event(c, fn() {
    var x = popup_show(g, 120, 100, 50, 50, true)
    set(r, 0, x)
})
add_s(c)
show()
```




Cobalah pindahkan posisi frame. Dalam hal ini, popup yang tampil akan ikut dan tetap berada dalam frame.

Sejauh ini, kita bekerja dengan popup di level frame. Bagaimana kalau kita ingin tampilkan popup di COMPONENT tertentu? Apakah kita harus secara manual mengetahui posisi COMPONENT tersebut

dan tampilkan popup di posisi tersebut? Tidak. Kita bisa gunakan fungsi popup_component.

Contoh berikut akan melunasi utang yang tersisa di bab sebelumnya, ketika bekerja dengan event fokus. Kita akan siapkan sebuah combobox. Apabila pengguna fokus pada combobox tersebut, kita tampilkan sebuah popup, sesuai lebar combobox. Sebagai bonus, kita berikan warna latar berbeda pada popup. Berikut adalah contoh kode programnya:

```
reset()
var t = component("text", "Focus on combobox")
var c = component("combobox", "")
var g = component("grid", "")
config(g, "background", "gray")

var r = [-1]

event_focus(c, fn(e) {
    if (e[0] == "GAINED") {
        var i = component_info(c)
        var x = popup_component(g, c, i[4], 300, 0,
i[5])
        set(r, 0, x)
    }
})
```

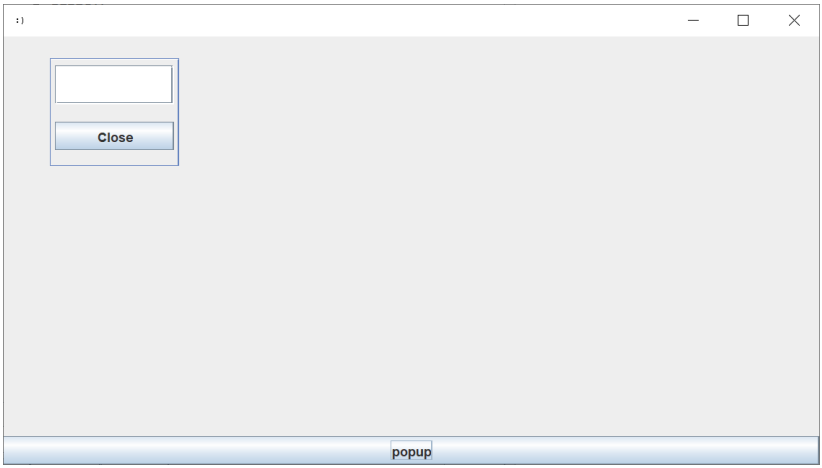
```
add_n([t, c])

show()
```


Penjelasan:

- Diwarnai hijau: kita menggunakan grid untuk popup dan berikan warna latar.
- Diwarnai biru: kita menangani event fokus untuk COMPONENT. Dalam hal ini, pada combobox.
- Diwarnai merah: kita tahu bahwa tipe yang dikembalikan bisa berupa GAINED atau LOST. Mari kita tangani ketika fokus diberikan (GAINED).
- Diwarnai oranye: kita dapatkan informasi COMPONENT combobox dan tampilkan popup pada COMPONENT tersebut. Fungsi popup_component:
    - Panel atau grid
    - COMPONENT
    - Lebar popup
    - Tinggi popup
    - Offset x
    - Offset y

Sejauh ini, kita hanya menampilkan popup. Bagaimana kalau kita perlu menyembunyikan popup yang telah tampil? Kita gunakan fungsi popup_hide. Lebih seru lagi (*apanya?*), tombol untuk menyembunyikan popup ditempatkan di dalam popup itu sendiri. Berikut adalah kode programnya:

```
reset()
var t = component("text", "")
var b = component("button", "Close")
var g = component("grid", "")
grid_add(g, t, 0, 0, 1, 1, 1, 1, 3, 0)
grid_add(g, b, 0, 1, 1, 1, 1, 1, 1, 0)

var r = [-1]

var c = component("button", "popup")
event(c, fn() {
    var x = popup_show(g, 120, 100, 50, 50, true)
    set(r, 0, x)
})
add_s(c)
show()

event(b, fn() {
    popup_hide(r[0])
})
```




Penjelasan:

- Diwarnai hijau: akhirnya kita akan membahas apa guna ARRAY dengan satu elemen ini. Dalam hal ini, gunanya hanya menampung nilai kembalian dari pemanggilan fungsi popup_show.
- Diwarnai biru: fungsi popup_show mengembalikan NUMBER, berupa indeks lebih besar atau sama dengan 0. Kita berada di dalam fungsi tersendiri, dan menggunakan bantuan variabel r, untuk menampung nilai kembalian. Kita hanya bekerja dengan indeks ke-0.
- Diwarnai merah: pada akhirnya, kita menyembunyikan popup dengan fungsi popup_hide, yang membutuhkan argumen berupa NUMBER, yang dalam hal ini, adalah indeks ke-0 dari ARRAY r, yang diset sebelumnya.

Contoh terakhir berikut akan lebih seru lagi: tampilkan popup ketika COMPONENT menerima fokus, dan sembunyikan popup ketika kehilangan fokus. Penulis yakin, berdasarkan contoh sebelumnya, hal ini tidaklah sulit. Mari kita lihat bersama:

```
reset()
var t = component("text", "Focus on combobox")
var c = component("combobox", "")
var g = component("grid", "")
config(g, "background", "gray")

var r = [-1]
event_focus(c, fn(e) {
    if (e[0] == "GAINED") {
        var i = component_info(c)
        var x = popup_component(g, c, i[4], 300, 0,
i[5])
        set(r, 0, x)
    } else {
        popup_hide(r[0])
    }
})

add_n([t, c])
show()
```

Ketika program dijalankan, fokuslah pada combobox, dan popup akan ditampilkan. Fokuslah pada text, sehingga combobox kehilangan fokus, maka popup yang tadinya tampil, akan disembunyikan.

# Pengantar analisis kebutuhan perangkat lunak

**Penulis: Dr. Wartika**

Analisis kebutuhan adalah suatu proses untuk mendapatkan informasi, mode, dan spesifikasi tentang perangkat lunak yang diinginkan klien/pengguna.

Baik klien ataupun pembuat perangkat lunak terlibat aktif dalam analisis kebutuhan perangkat lunak ini. Informasi dari klien akan menjadi acuan untuk melakukan perancangan perangkat lunak. Ada beberapa keterangan mengenai analisis kebutuhan, yaitu:

- Analisis kebutuhan merupakan satu diantara banyak aktivitas kritis pada proses rekayasa kebutuhan perangkat lunak untuk memahami ranah permasalahan dari sistem yang berjalan dan ranah solusi dari sistem yang akan dibuat.
- Analisis kebutuhan merupakan bagian dari proses kebutuhan perangkat lunak yang berperan menjembatani perbedaan yang terjadi antara level rekayasa kebutuhan dan perancangan perangkat lunak.
- Analisis kebutuhan bertujuan menyempurnakan kebutuhan-kebutuhan yang ada untuk memastikan pemangku kepentingan memahaminya dan menemukan kesalahan-kesalahan, kalalaian, dan kekurangan lainnya jika ada.

Tujuan dari analisis kebutuhan ini adalah:

- Mengolah hasil elisitasi kebutuhan untuk menghasilkan dokumen spesifikasi kebutuhan yang isi keseluruhannya sesuai dengan apa yang diinginkan pengguna.

- Mengembangkan persyaratan kualitas yang memadai dan rinci, sehingga para manajer dapat membuat perkiraan proyek yang realistis dan staf teknis dapat melanjutkan dengan perancangan, implementasi, dan pengujian.
- Membangun pemahaman tentang karakteristik ranah permasalahan dan sekumpulan kebutuhan untuk menemukan solusi.

Sedangkan kebutuhan perangkat lunak adalah kondisi, kriteria, syarat atau kemampuan yang harus dimiliki oleh perangkat lunak untuk memenuhi apa yang disyaratkan atau diinginkan pemakai.

Ada tiga buah jenis kebutuhan perangkat lunak, diantaranya adalah sebagai berikut:

1. Kebutuhan fungsional (*functional requirement*) / kebutuhan operasional. Merupakan kebutuhan yang berkaitan dengan fungsi atau proses transformasi yang harus mampu dikerjakan oleh perangkat lunak.
2. Kebutuhan antarmuka (*interface requirement*). Kebutuhan antarmuka yang menghubungkan perangkat lunak dengan elemen perangkat keras, perangkat lunak, atau basis data.
3. Kebutuhan unjuk kerja (*performance requirement*). Kebutuhan yang menetapkan karakteristik unjuk kerja yang harus dimiliki oleh perangkat lunak, misalnya: kecepatan, ketepatan, frekuensi.

Adapun teknik kebutuhan perangkat lunak meliputi 3 tahap, yaitu *elicitation* (pengumpulan informasi), *specification* (sepsifikasi), dan *validation* (validasi).

Metode analisis kebutuhan diantaranya adalah sebagai berikut:

1. Interviews. Wawancara (*interview*) merupakan percakapan antara dua orang atau lebih dan berlangsung antara narasumber dan pewawancara.
2. Kuesioner (*questionnaire*), yang merupakan teknik pengumpulan data yang dilakukan dengan cara memberi seperangkat pertanyaan tertulis kepada responden untuk dijawabnya, dapat diberikan secara langsung atau melalui pos atau internet. Jenis kuesioner ada dua, yaitu tertutup dan terbuka.
3. Analisis prosedur (*procedure analysis*), yang bertujuan untuk mengetahui lebih jelas bagaimana cara kerja sistem tersebut sehingga kelebihan dan kekurangan sistem dapat diketahui.
4. Survey dokumen (*document survey*), yang merupakan proses mengumpulkan informasi tentang topik tertentu dengan tujuan menggunakan data.

Ada tiga faktor yang harus dipenuhi ketika melakukan analisis kebutuhan ini, yaitu lengkap, detail, dan benar. Lengkap artinya semua yang diharapkan oleh klien telah didapatkan oleh pihak yang melakukan analisis. Detail maksudnya adalah berhasil mengumpulkan informasi yang terperinci. Semua data dari analisis kebutuhan ini haruslah benar, sesuai apa yang dimaksud oleh klien, bukan benar menurut apa yang dipikirkan oleh pihak analisis.

Analisis kebutuhan yang dilakukan terhadap perangkat lunak akan menghasilkan spesifikasi perangkat lunak tersebut. Analisis kebutuhan ini terdiri dari lima langkah pokok:

1. Identifikasi masalah
2. Evaluasi dan sintesis
3. Pemodelan
4. Spesifikasi
5. Review

Tahap analisis adalah tahapan pengumpulan kebutuhan-kebutuhan dari semua elemen sistem perangkat lunak yang akan di bangun. Pada tahap ini dibentuk spesifikasi kebutuhan perangkat lunak, fungsi perangkat lunak yang dibutuhkan, performansi (unjuk kerja) sistem perangkat lunak, penjadwalan proyek, identifikasi sumber daya (manusia, perangkat keras, dan perangkat lunak yang dibutuhkan) dan taksiran biaya pengembangan perangkat lunak. Pada dasarnya, aktivitas analisis dibutuhkan dalam setiap proses dalam daur hidup pengembangan perangkat lunak. Di dalam proses rekayasa kebutuhan, analisis pun dilakukan dalam setiap aktivitas-aktivitasnya. Penjelasan dari masing-masing aktivitas tersebut sebagai berikut:

1. *Domain understanding*: dalam tahap ini perekayasa kebutuhan perangkat lunak harus mengetahui bagaimana organisasi perusahaan beroperasi dan apa yang menjadi permasalahan pada sistem yang sedang berjalan pada saat ini. Perekayasaan perlu memfokuskan kepada 'Apa' yang menjadi permasalahan. Perekayasaan hendaknya tidak berhenti pada menemukan "gejala" dari permasalahan itu terjadi untuk menemukan akar dari pemasalahan dari sistem yang berjalan tersebut.
2. *Requirements collection*: tahapan ini merupakan tahapan pengumpulan kebutuhan akan sistem yang akan dibangun. Pada tahapan ini diperlukan adanya interaksi intensif dengan pemangku kepentingan terutama dengan pengguna akhir.
3. *Classification*: pada tahapan sebelumnya kumpulan kebutuhan masih tidak terstruktur. Untuk itu kebutuhan yang saling berkaitan dikelompokan, baik menurut kelas penggunaannya maupun jenis kebutuhanannya. Kebutuhan-kebutuhan tersebut diorganisir ke dalam kelompok-kelompok yang koheren. Perekayasaan perlu memisahkan antara kebutuhan dan keinginan dari pengguna.

4. *Conflict resolution*: pada tahapan ini adalah menemukan dan menyelesaikan kebutuhan yang di dalamnya terdapat konflik.
5. *Prioritisation*: pada tahapan ini dilakukan interaksi dengan pemangku kepentingan untuk mengidentifikasikan kebutuhan-kebutuhan prioritas dari masing-masing kebutuhan agar sumber daya yang tersedia pada organisasi dialokasikan untuk mengimplementasikan kebutuhan terutama dari pemangku kepentingan.
6. *Requirements Checking*: menganalisis sekumpulan kebutuhan dari hasil tahapan sebelumnya untuk memverifikasi dan memvalidasi berdasarkan aspek kelengkapan, konsistensi, dan kebutuhan nyata.

Dalam rekayasa kebutuhan, analisa kebutuhan yang baik hendaklah menitikberatkan pada ranah permasalahan dan bukan pada ranah solusi. Tujuan utamanya adalah untuk mencapai pemahaman tentang sifat dari ranah permasalahan dan permasalahan yang ada didalamnya. Pada dasarnya, analisis kebutuhan diawali dengan spesifikasi (layanan, atribut, properti, kualitas, batasan) dari sistem solusi yang hendak dibangun.

*Halaman ini sengaja dikosongkan*

# Pengantar pemodelan matematika dalam ilmu komputer

**Penulis: Dr. Ford Lumban Gaol**

Pemodelan matematika menggambarkan suatu proses dan suatu objek dengan menggunakan bahasa matematika. Suatu proses atau objek disajikan dalam "bentuk murni" dalam pemodelan matematika ketika pengaruh eksternal yang mengganggu muncul.

Simulasi komputer merupakan kelanjutan dari pemodelan matematika. Simulasi komputer dapat dianggap sebagai eksperimen komputer yang sesuai dengan eksperimen di dunia nyata. Perlakuan seperti itu agak terkait dengan simulasi numerik. Simulasi simbolik menghasilkan lebih dari sekadar eksperimen. Mereka dapat dianggap sebagai transformasi model matematika oleh komputer, karena simulasi simbolik menyimpan parameter model dalam bentuk simbolis yang sesuai dengan serangkaian eksperimen yang sebenarnya.

Seseorang dapat memperoleh hasil numerik seperti dalam eksperimen aktual hanya setelah penggantian parameter simbolik dengan data numerik. Oleh karena itu, simulasi simbolik melengkapi model matematika dan banyak digunakan dalam eksperimen yang sebenarnya. Pemodelan matematika dari proses stokastik didasarkan pada teori probabilitas, khususnya, yang mengarah pada penggunaan jalan acak, metode Monte Carlo dan alat statistik standar.

Simulasi simbolik biasanya diwujudkan dalam bentuk penyelesaian persamaan di salah satu yang tidak diketahui, ke sistem persamaan aljabar linier, baik persamaan diferensial biasa maupun parsial (ODE dan PDE). Metode diskrit seperti Metode Elemen Hingga atau

Metode Perbedaan Hingga untuk menemukan solusi pendekatan ODE dan PDE biasanya digunakan dalam bentuk simulasi numerik.

**Latar Belakang Pemodelan Matematika**

Model menggambarkan keyakinan kita tentang bagaimana dunia berfungsi. Dalam pemodelan matematika, dijemahkan keyakinan tersebut ke dalam bahasa matematika. Ini memiliki banyak keuntungan:

1. Matematika adalah bahasa yang sangat tepat. Ini membantu kita merumuskan ide dan mengidentifikasi asumsi yang mendasarinya.
2. Matematika adalah bahasa yang ringkas, dengan aturan manipulasi yang jelas.
3. Semua hasil yang telah dibuktikan oleh para ahli matematika selama ratusan tahun tersedia untuk kita.
4. Komputer dapat digunakan untuk melakukan perhitungan numerik.

Ada elemen besar kompromi dalam pemodelan matematika. Mayoritas sistem yang berinteraksi di dunia nyata terlalu rumit untuk dimodelkan secara keseluruhan.

Oleh karena itu, kompromi tingkat pertama adalah mengidentifikasi bagian terpenting dari sistem. Ini akan dimasukkan dalam model, sisanya akan dikeluarkan.

Tingkat kompromi kedua menyangkut jumlah manipulasi matematika yang bermanfaat. Meskipun matematika memiliki potensi untuk membuktikan hasil umum, hasil ini sangat bergantung pada bentuk persamaan yang digunakan. Perubahan kecil dalam

struktur persamaan mungkin memerlukan perubahan besar dalam metode matematika. Menggunakan komputer untuk menangani persamaan model mungkin tidak pernah menghasilkan hasil yang elegan, tetapi jauh lebih kuat terhadap perubahan.

**Tujuan Pemodelan Matematika**

Pemodelan matematika dapat digunakan untuk sejumlah alasan yang berbeda. Seberapa baik tujuan tertentu tercapai tergantung pada status pengetahuan tentang sistem dan seberapa baik pemodelan dilakukan.

Contoh jangkauan tujuan adalah:

1. Mengembangkan pemahaman ilmiah: melalui ekspresi kuantitatif dari pengetahuan terkini tentang suatu sistem (selain menampilkan apa yang kita ketahui, ini juga dapat menunjukkan apa yang tidak kita ketahui).
2. Menguji pengaruh perubahan dalam suatu sistem.
3. Membantu pengambilan keputusan, termasuk (i) keputusan taktis oleh manajer; (ii) keputusan strategis oleh para perencana.

**Klasifikasi Pemodelan Matematika**

Saat mempelajari model, akan sangat membantu untuk mengidentifikasi kategori model yang luas. Klasifikasi model individu ke dalam kategori ini memberitahu kita segera beberapa hal penting dari struktur mereka. Satu divisi antara model didasarkan pada jenis hasil yang mereka prediksi.

Model deterministik mengabaikan variasi acak, dan karenanya selalu memprediksi hasil yang sama dari titik awal yang diberikan. Di sisi

lain, model mungkin lebih bersifat statistik dan dapat memprediksi distribusi hasil yang mungkin. Model seperti itu dikatakan stokastik.

Metode kedua untuk membedakan antara jenis model adalah dengan mempertimbangkan tingkat pemahaman yang menjadi dasar model. Penjelasan paling sederhana adalah dengan mempertimbangkan hierarki struktur organisasi dalam sistem yang dimodelkan.

Sebuah model yang menggunakan sejumlah besar informasi teoritis umumnya menggambarkan apa yang terjadi pada satu tingkat dalam hirarki dengan mempertimbangkan proses di tingkat yang lebih rendah. Ini disebut model mekanistik, karena mereka memperhitungkan mekanisme melalui mana perubahan terjadi.

Dalam model empiris, tidak ada mekanisme yang menyebabkan perubahan pada sistem terjadi. Sebaliknya, hanya dicatat bahwa mereka memang terjadi, dan model mencoba untuk memperhitungkan secara kuantitatif perubahan yang terkait dengan kondisi yang berbeda. Dua divisi di atas, yaitu deterministik/stokastik dan mekanistik/empiris, merupakan ekstrem dari berbagai jenis model. Di antaranya terletak seluruh spektrum tipe model. Juga, dua metode klasifikasi saling melengkapi. Misalnya, model deterministik dapat berupa mekanistik atau empiris (tetapi tidak stokastik).

Satu jenis model lebih lanjut, model sistem, layak disebutkan. Ini dibangun dari serangkaian sub-model, yang masing-masing menggambarkan esensi dari beberapa komponen yang berinteraksi. Metode klasifikasi di atas kemudian merujuk lebih tepat ke sub-model: berbagai jenis sub-model dapat digunakan dalam satu model sistem.

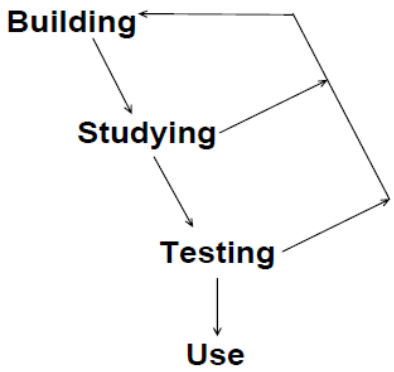
Sebagian besar literatur pemodelan mengacu pada 'model simulasi'. Mengapa mereka tidak termasuk dalam klasifikasi? Alasan untuk penghilangan yang jelas ini adalah bahwa 'simulasi' mengacu pada

cara perhitungan model dilakukan - yaitu dengan simulasi komputer. Model sebenarnya dari sistem tidak berubah dengan cara matematika yang diperlukan dilakukan, meskipun interpretasi model kita mungkin bergantung pada akurasi numerik dari pendekatan apa pun.

**Langkah-langkah Pemodelan**

Akan sangat membantu untuk membagi proses pemodelan ke dalam empat kategori besar aktivitas, yaitu building, studying, testing, dan use. Meskipun mungkin menyenangkan untuk berpikir bahwa proyek pemodelan berjalan dengan lancar dari membangun hingga menggunakan, ini hampir tidak pernah terjadi.

Secara umum, cacat yang ditemukan pada tahap belajar dan pengujian diperbaiki dengan kembali ke tahap pembangunan. Perhatikan bahwa jika ada perubahan pada model, maka tahap belajar dan pengujian harus diulang. Representasi bergambar dari rute potensial melalui tahapan pemodelan dapat dilihat sebagai berikut.



**Memulai Pemodelan**

Sebelum memulai proyek pemodelan, kita harus jelas tentang tujuan kita. Ini menentukan arah masa depan proyek dalam dua cara.

Pertama, tingkat detail yang dimasukkan dalam model tergantung pada tujuan model akan digunakan. Misalnya, dalam pemodelan pertumbuhan hewan untuk bertindak sebagai bantuan untuk penasihat pertanian, model empiris yang berisi istilah-istilah untuk faktor penentu pertumbuhan yang paling penting mungkin cukup memadai. Model dapat dianggap sebagai ringkasan pemahaman saat ini. Model seperti itu jelas sangat terbatas penggunaannya sebagai alat penelitian untuk merancang eksperimen untuk menyelidiki proses nutrisi ruminansia.

Kedua, kita harus membuat pemisahan antara sistem yang akan dimodelkan dan lingkungannya. Pembagian ini baik dilakukan jika lingkungan mempengaruhi perilaku sistem, tetapi sistem tidak mempengaruhi lingkungan. Misalnya, dalam memodelkan pertumbuhan perkebunan konifer kecil untuk memprediksi hasil kayu, disarankan untuk memperlakukan cuaca sebagai bagian dari lingkungan. Pengaruhnya terhadap pertumbuhan dapat digabungkan dengan menggunakan ringkasan statistik iklim di lokasi yang sama dalam beberapa tahun terakhir. Namun, model apa pun untuk pertumbuhan hutan dunia hampir pasti harus mengandung istilah untuk efek pertumbuhan pada cuaca. Tutupan pohon diketahui memiliki pengaruh besar terhadap cuaca melalui tingkat karbon dioksida di atmosfer.

## Contoh 6: Proses yang berjalan lama

Sampai juga kita pada pembahasan panjang lainnya. Awalnya, topik ini rencananya akan dibahas pada buku berbeda. Akan tetapi, karena judul buku mengandung kata mahir, maka rasanya tak terhindarkan untuk dibahas.

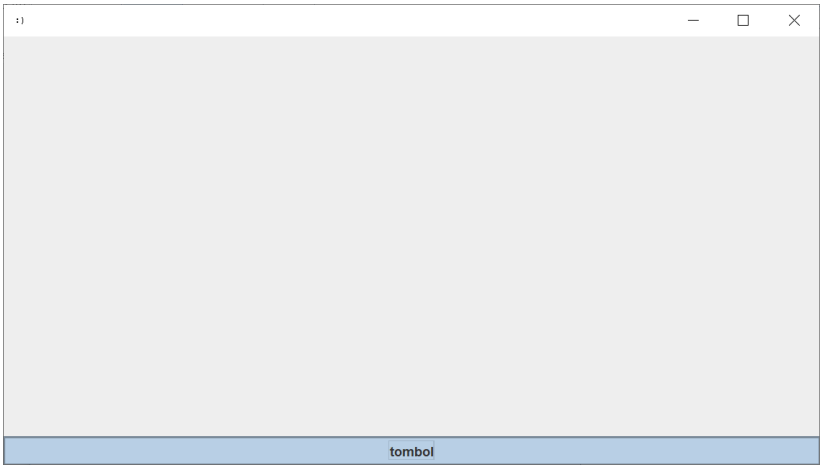
Dan, mari kita buka pembahasan kita, langsung dengan sebuah permasalahan:

```
reset()

var b = component("button", "tombol")

add_s(b)

event(b, fn() {

    delay(5000)

})

show()
```

Tidak ada yang spesial pada contoh kode program tersebut. Ketika button ditekan, event handler yang berisikan satu baris kode berikut akan dipanggil:

```
delay(5000)
```

Fungsi delay, per dokumentasinya, akan sleep untuk sekian milidetik. Dalam hal ini, maka 5.000 milidetik, atau 5 detik. Pada kenyataannya, memang itu yang terjadi. Sayangnya, user interface menjadi tidak lagi merespon terhadap input. Buttonnya saja, setidaknya di sistem yang penulis gunakan untuk menulis buku ini, berada dalam status visual ditekan. Setelah 5 detik, barulah kembali normal.




Apakah ini sungguh menjadi masalah, sehingga perlu dibahas dalam bab tersendiri? Bisa iya, bisa tidak. Lho, bagaimana ini?

Maksud penulis, tentunya, untuk program sederhana, tentu tidak menjadi masalah. Akan tetapi, apabila delay tersebut sebagai contoh digantikan dengan HTTP request yang membutuhkan waktu lebih lama, maka rasanya kurang begitu baik, dari sisi pengalaman pengguna program.

Singkong menyediakan dua cara: wait dan bekerja dengan thread secara manual.

### Wait

Mari kita lihat cara pertama, yaitu dengan fungsi wait. Perhatikanlah contoh kode program berikut:

```
reset()
var a = component("label", "please wait...")
config(a, "font", ["monospaced", 1, 20])
```

```
var b = component("button", "Wait")
add_s(b)

event(b, fn() {
    wait(a, delay, 5000)
})

show()
```


Ketika button Wait ditekan, button menghilang dari frame, dan sebaliknya digantikan dengan tulisan please wait… sebagaimana yang tampil pada screenshot.

Apa pula ini?

Berikut adalah penjelasannya:

- Diwarnai hijau: kita siapkan sebuah label yang akan ditampilkan, ketika suatu proses yang berjalan lama harus dijalankan. Tidak ada yang khusus dengan baris kode ini. Bahwa kita mengatur font agar lebih besar, di baris setelahnya, hanyalah supaya terlihat lebih jelas saja.
- Diwarnai biru: baris ini adalah kuncinya. Fungsi wait membutuhkan argumen:
    - COMPONENT, yang hanya dapat berupa sebuah label.
    - FUNCTION atau BUILTIN, yang akan dijalankan pada thread terpisah.
    - Argumen opsional berikutnya, apabila ada, akan dilewatkan ke FUNCTION atau BUILTIN sebelumnya.

Mari kita coba ganti label dengan COMPONENT lain:

```
wait(component("button", ""), delay, 5000)
```

Ketika button Wait ditekan, pesan kesalahan akan ditampilkan, menginformasikan bahwa hanya label yang dapat diterima. Mari kita terima, untuk sementara ini, kita harus gunakan label.

Fungsi delay, dalam hal ini, hanyalah contoh. Mari kita gunakan FUNCTION, alih-alih delay, yang merupakan BUILTIN. Kita ganti pemanggilan fungsi event menjadi:

```
event(b, fn() {
    wait(a, fn() {
        delay(5000)
    })
})
```

Sekarang, andaikata fungsi yang dibuat membutuhkan argumen, kita bisa lewatkan ketika memanggil fungsi wait:

```
event(b, fn() {
    wait(a, fn(x) {
        delay(x)
    },
    5000)
})
```

Atau, kita bisa gunakan FUNCTION dengan nama:

```
var f = fn(x) {
    delay(x)
}
event(b, fn() {
    wait(a, f, 5000)
})
```

Bahwa dijalankan pada thread terpisah, sebagai salah satu bentuk dari konkurensi, memungkinkan program lebih nyaman digunakan. Dalam konkurensi, tugas-tugas dijalankan pada waktu yang saling overlap.

Dalam contoh sederhana, button tidak lagi dalam kondisi seperti ditekan, yang baru akan kembali normal setelah proses yang membutuhkan waktu lama tersebut selesai.

Anda mungkin berpendapat: tentu saja buttonnya tidak lagi demikian. Buttonnya saja menghilang :)

Baiklah, pendapat yang sangat tepat, bukan? Solusi ini juga mungkin tidak ideal.

Mari kita tambahkan satu baris kode berikut:

```
add(component("edit", ""))
```

Sebuah COMPONENT edit akan ditampilkan. Anggap saja ini mewakili sejumlah COMPONENT lain di frame.

Ketika button Wait ditekan, semua COMPONENT yang tampil, bukan saja button Wait, akan turut menghilang. Baru muncul lagi setelah proses berjalan lamanya selesai.

Dengan demikian, dapat kita simpulkan, bahwa walaupun mudah, fungsi wait akan mengganti sementara semua COMPONENT yang ada dengan sebuah label. Tergantung kebutuhan Anda, ini mungkin dapat atau mungkin tidak dapat diterima.

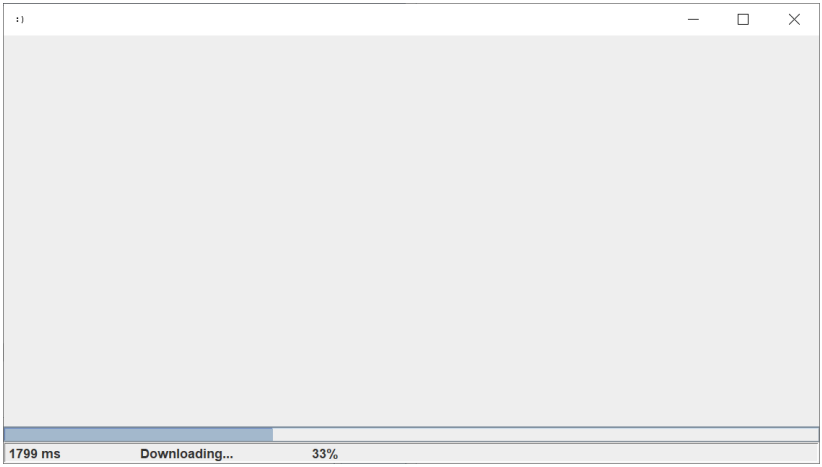
**Bekerja dengan thread**

Mari kita sajikan sebuah contoh yang rumit. Tujuannya tentu bukan supaya Anda berhenti membaca buku ini sampai di sini. Melainkan ingin memperlihatkan bahwa beberapa hal perlu dibuat secara manual, tapi mungkin dapat memenuhi kebutuhan Anda.

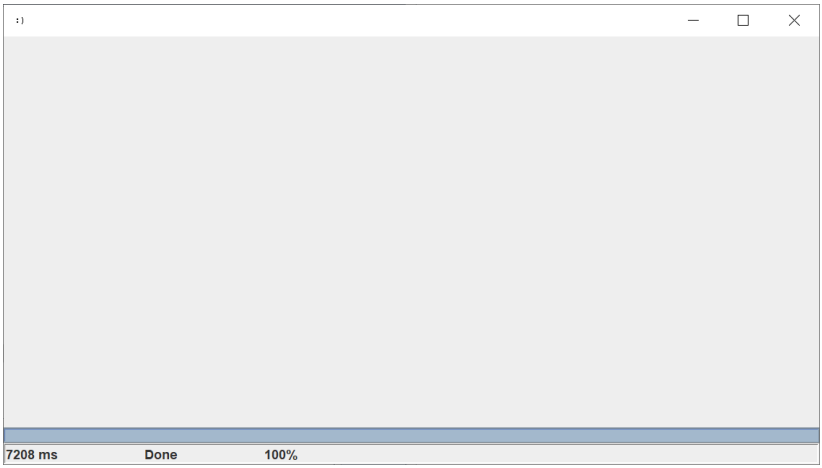
```
var p = [0, "Connecting..."]


var f = fn() {
    delay(random(1000, 1500))
    set(p, 0, random(30, 35))
```

```
    set(p, 1, "Downloading...")
    delay(random(2000, 2500))
    set(p, 0, random(70, 75))
    set(p, 1, "Processing...")
    delay(random(3000, 3500))
    set(p, 0, 100)
    set(p, 1, "Done")
    delay(1000)
}

var t = thread(f, p)

var s = number(@)

reset()
var x = component("progress", "")
add_s(x)

var c = fn() {
  if (thread_alive(t)) {
    var m = number(@) - s
    statusbar(0, m + " ms", true)
    statusbar(1, p[1], true)
```

```
    statusbar(2, p[0] + "%", true)
    config(x, "contents", p[0])
  } else {
    stop()
  }
}

timer(random(100, 500), c)
show()
```




Tunggulah selama beberapa saat, dan setelahnya, gambar layar berikut akan ditampilkan.




Ketika program dijalankan:

- Statusbar, pada bagian pertama (indeks 0), akan menampilkan waktu program telah berjalan sampai simulasi proses yang berjalan lama selesai.
- Statusbar, pada bagian kedua, akan berisikan informasi apa yang sedang terjadi. Pada contoh program ini, dapat berupa: Connecting..., Downloading..., Processing..., dan Done.
- Statusbar, pada bagian ketiga, akan menampilkan berapa persen.
- Progresbar akan tampil sesuai persentase.
- Secara berkala setiap waktu tertentu (awalnya berupa waktu acak antara 100 sampai 500 milidetik), sebuah fungsi akan dipanggil. Kita telah membahas contoh serupa pada buku sebelumnya.
- Proses yang berjalan lama disimulasikan dengan:
    - Menunda dengan nilai acak antara 1000 sampai 1500.
    - Mengatur progress dengan nilai acak 30 dan 35 persen.
    - Mengatur informasi: Downloading....
    - Menunda dengan nilai acak antara 2000 sampai 2500.
    - Mengatur progress dengan nilai acak 70 dan 75 persen.

    - Mengatur informasi: Processing....
    - Menunda dengan nilai acak antara 3000 sampai 3500.
    - Mengatur progress 100 persen.
    - Mengatur informasi: Done.
    - Menunda 1000 milidetik.
- Selama thread masih berjalan (poin-poin sebelumnya), update pada user interface akan dilakukan.

Bisa dilihat, selama kita mendapatkan status berapa persen dan apa yang sedang dilakukan, kita bisa menggunakan berbagai cara untuk memvisualisasikan. Tentunya, ini tidak seperti fungsi wait.

Penjelasan kode program:

- Diwarnai hijau: fungsi c akan dipanggil secara berkala dengan timer. Apa yang dilakukan fungsi c pada dasarnya adalah:
    - Apabila thread masih aktif, visualisasikan progress.
    - Apabila tidak, maka hentikan timer. Dengan demikian, fungsi c tidak lagi dipanggil. Catatan: fungsi stop akan menghentikan semua timer. Anda mungkin ingin menggunakan fungsi stop_timer untuk timer tertentu saja.
- Diwarnai hijau: Bagaimana memeriksa apakah thread masih aktif? Kita gunakan fungsi thread_alive.
- Diwarnai hijau, diwarnai oranye: Untuk mendapatkan waktu yang telah berjalan, kita konversi waktu ke NUMBER, dikurangi ketika pertama kali waktu dicatat (variabel s).
- Diwarnai merah, diwarnai biru: kita membuat thread baru dengan fungsi thread. Dalam hal ini, variabel t merujuk pada id thread yang dibuat. Thread t dibuat untuk menjalankan fungsi f. Dalam kondisi normal, ketika f selesai dijalankan, maka thread t tersebut tidak lagi aktif. Butuh sekian waktu (dengan rentang acak tertentu) sampai fungsi f selesai, karena ada pemanggilan delay.

- Perhatikanlah bahwa ARRAY p, hanya diupdate pada fungsi f. Fungsi c hanyalah membaca isinya.

Contoh ini mengakhiri pembahasan tentang proses yang berjalan lama ketika bekerja dengan GUI. Akan tetapi, seperti sebelumnya, penulis masih perlu menambah jumlah halaman :) Lagipula, ketika bekerja dengan beberapa thread, Anda mungkin tertarik dengan beberapa contoh program berikut.

Yang pertama cukup jelas:

```
var a = [0]


var f = fn() {
        var i = 0
        repeat {
                set(a, 0, a[0] + 1)
                var i = i + 1
                if (i >= 100000) {
                        return i
                }
        }
}


f()
println(a)
```

Dalam program tersebut, setiap 100.000 kali perulangan dilakukan (di dalam fungsi f), elemen pertama dalam ARRAY a akan ditambahkan satu. Dengan demikian, di akhir program, ARRAY a akan berisi nilai [100000].

Yang kedua, kita dua thread menjalankan fungsi f, dan berusaha untuk mengubah ARRAY a yang sama:

```
var a = [0]


var f = fn() {
        var i = 0
        repeat {
                set(a, 0, a[0] + 1)
                var i = i + 1
                if (i >= 100000) {
                        return i
                }
        }
}


var t1 = thread(f)
var t2 = thread(f)
thread_join(t1)
thread_join(t2)
```

```
println(a)
```

Jalankanlah program kedua ini beberapa kali. Hasilnya akan berbeda-beda. Padahal, kita mengharapkan nilai a pada akhir program adalah [200000], karena total perulangan adalah 200.000 kali. Ini tidak terjadi, diantaranya karena, operasi penambahan satu (diwarnai merah) setiap kali perulangan ini tidaklah atomik, dan perpindahan konteks thread dapat terjadi (sebuah thread mungkin telah menjumlahkan, tapi belum sempat mengubah ARRAY, sudah pindah ke thread lain, dan seterusnya).

Mari kita perbaiki dengan program ketiga, yang menggunakan intrinsic lock (diwarnai hijau):

```
var a = [0]


var f = fn() {
        var i = 0
        repeat {
                set(a, 0, a[0] + 1)
                var i = i + 1
                if (i >= 100000) {
                        return i
                }
        }
}
```

```
var t1 = thread(f, a)
var t2 = thread(f, a)
thread_join(t1)
thread_join(t2)

println(a)
```

Ketika program ini dijalankan beberapa kali misalnya, nilai a akan selalu [200000]. Kode programnya tidak beda banyak dengan program kedua. Bedanya hanyalah pada penggunaan lock ketika memanggil fungsi thread.

Anda dapat menggunakan null, true, atau false sebagai intrinsic lock. Selama menggunakan lock yang sama. Sebagai contoh, pada potongan kode berikut:

```
var t1 = thread(f, null)
var t2 = thread(f, null)
```

Sementara, potongan kode berikut tidak menggunakan lock yang sama (karena merujuk pada dua STRING berbeda, walau nilainya sama):

```
var t1 = thread(f, "Singkong")
var t2 = thread(f, "Singkong")
```

Bagaimana, kita dapat beberapa halaman tambahan bukan?

## Contoh 7: Pencetakan ke printer

Untuk mencetak ke printer, tersedia tiga fungsi bawaan: printer, table_print, dan x_edit_print. Mari kita bahas fungsi pertama.

Fungsi printer akan menampilkan dialog print untuk mencetak ARRAY dari STRING. Fungsi ini membutuhkan argumen berikut:

- ARRAY dari STRING yang akan dicetak.
- Ukuran font, berupa NUMBER.
- Margin sisi kiri, berupa NUMBER.
- Margin sisi atas, berupa NUMBER
- Opsional: nama font berupa STRING.

Dengan demikian, apabila kita ingin mencetak misal:

Singkong

goreng

Maka, kita siapkan ARRAY: ["Singkong", "goreng"], kemudian panggil fungsi printer, seperti contoh berikut:

```
printer(["Singkong", "goreng"], 30, 50, 60)
```

Begitu saja? Hanya satu baris? Benar.

Tapi kita memang belum sepenuhnya selesai, karena kita dapat memberikan nama font yang akan digunakan, seperti contoh berikut:

```
printer(["Singkong", "goreng"], 30, 50, 60,
"SansSerif.bold")
```

atau

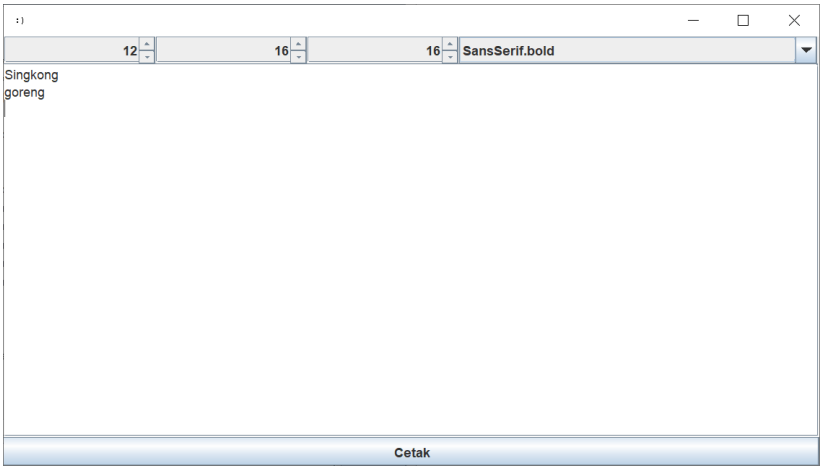
```
printer(["Singkong", "goreng"], 30, 50, 60,
"SansSerif.italic")
```

Untuk mendapatkan nama font yang tersedia, kita bisa gunakan fungsi fonts, yang akan kita bahas dalam bab tersendiri.

Sampai di sini, kita telah selesai membahas penggunaan fungsi printer. Untuk contoh yang lebih umum, mari kita buat program lengkap berikut:

```
reset()
var c = component("combobox", join(",", fonts()))
var s1 = component("spin", "12, 4, 128, 1")
var s2 = component("spin", "16, 4, 128, 1")
var s3 = component("spin", "16, 4, 128, 1")
var t = component("edit", "")
var b = component("button", "Cetak")
add_n([s1, s2, s3, c])
add(t)
add_s(b)
show()

event(b, fn() {
    var _t = trim(get(t, "contents"))
    var _s1 = get(s1, "contents")
    var _s2 = get(s2, "contents")
    var _s3 = get(s3, "contents")
    var _c = get(c, "text")
    if (!empty(_t)) {
```

```
        var _t = split(_t, lf())
        printer(_t, _s1, _s2, _s3, _c)
    } else {
        message("Tidak ada yang dicetak", "Info")
    }
})
```




Penjelasan:

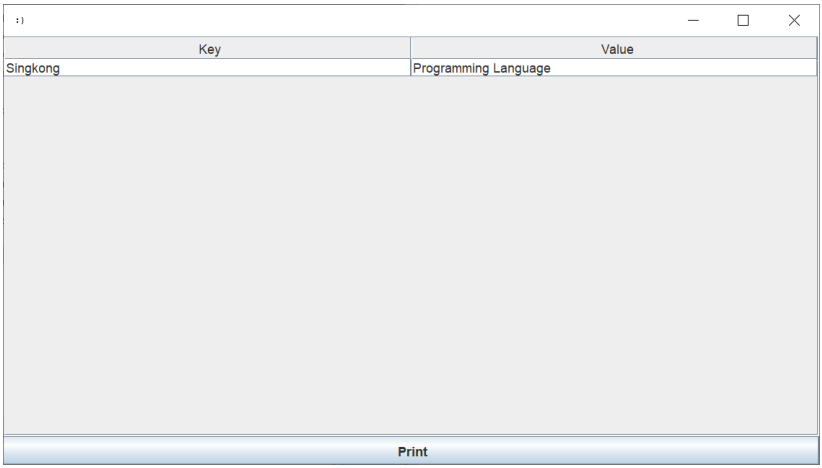
- Diwarnai hijau: isi awal combobox mengharapkan nilai STRING yang dipisahkan koma. Sementara fungsi fonts mengembalikan ARRAY. Oleh karena itu, kita join setiap elemen dalam ARRAY dengan sebuah koma.
- Diwarnai biru: untuk contoh ini, kita melakukan trim untuk isi COMPONENT edit.
- Diwarnai merah: "contents" yang didapatkan dari COMPONENT edit akan berupa STRING yang dipisahkan line feed. Oleh karena

itu, kita split dengan pemisah berupa line feed (yang dikembalikan dengan pemanggilan fungsi lf).

Pencetakan lain yang lebih spesifik dapat menggunakan fungsi table_print dan x_edit_print. Sesuai namanya, table_print digunakan untuk mencetak tabel dan x_edit_print digunakan untuk mencetak isi COMPONENT edit.

Berikut adalah contoh penggunaan table_print:

```
reset()

var t = component("table", "Key, Value")

var b = component("button", "Print")

table_add(t, [["Singkong", "Programming Language"]])

add(t)

add_s(b)

show()


event(b, fn() {
    var h = input("Header")
    var f = input("Footer")
    table_print(t, h, f)
})
```

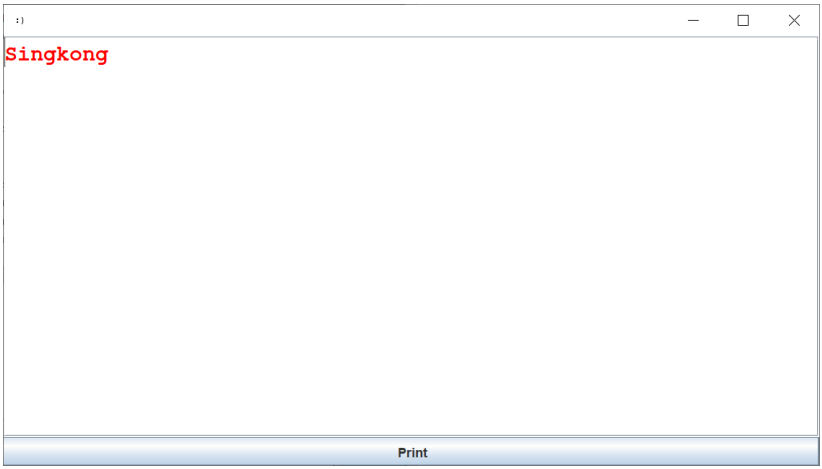



Ketika button Print ditekan, sebuah input dialog untuk header, kemudian input dialog untuk footer (diwarnai hijau) akan ditampilkan. Berdasarkan nilai-nilai tersebut, table t akan dicetak (diwarnai biru). Yang perlu diperhatikan di sini adalah keseluruhan table akan dicetak, apa adanya. Andaikata ukuran kolom diubah, maka akan tercermin dalam hasil cetak.

Berikut adalah contoh penggunaan x_edit_print:

```
reset()
var e = component("edit", "Singkong")
config(e, "foreground", "red")
config(e, "font", ["monospaced", 1, 20])
var b = component("button", "Print")
add(e)
add_s(b)
show()
```

```
event(b, fn() {
    var h = input("Header")
    var f = input("Footer")
    x_edit_print(e, h, f)
})
```




Ketika button Print ditekan, sebuah input dialog untuk header, kemudian input dialog untuk footer akan ditampilkan. Setelah itu, isi dari COMPONENT edit tersebut akan dicetak. Perhatikanlah catatan berikut:

- Diwarnai merah: warna foreground dan font yang digunakan akan turut dicetak. Dengan demikian, dalam contoh ini, hasil cetakan menggunakan font berwarna merah, termasuk header dan footer. Isi COMPONENT edit akan dicetak sesuai font yang digunakan.

- Bahwa nama fungsi diawali dengan x_ artinya fungsi ini merupakan fungsi extended di Singkong, dan membutuhkan versi runtime Java > 5.0 (yang dirilis 2004). Dalam hal ini, x_edit_print membutuhkan Java runtime *minimum* versi 6, yang dirilis pada tahun 2006 (16 tahun lalu, pada saat buku ini ditulis). Apabila tidak terpenuhi, fungsi akan tetap tersedia, namun selalu mengembalikan NULL.

*Halaman ini sengaja dikosongkan*

## Contoh 8: Bekerja dengan tanggal dan kalender

Untuk mendapatkan input berupa tanggal atau tanggal/waktu, kita dapat menggunakan COMPONENT date, yang memungkinan formatnya ditentukan.

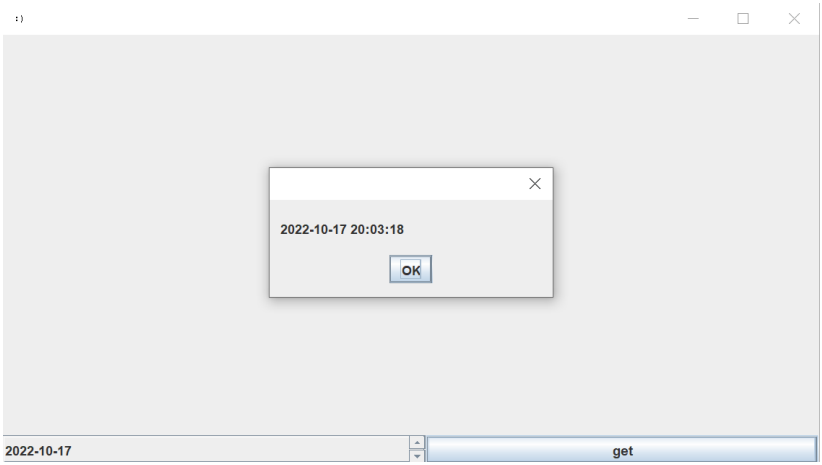
Mari kita lihat contoh berikut:

```
reset()
var d = component("date", "")
var b = component("button", "get")
add_s([d, b])
show()


event(b, fn() {
    var c = get(d, "contents")
    message(c)
})
```

Penjelasan:

- Diwarnai hijau: argumen kedua (nama COMPONENT) pada COMPONENT date adalah format tanggal, yang apabila tidak diberikan (seperti contoh ini), maka akan berupa: yyyy-MM-dd. Anda mungkin ingin mencoba format:
    - yyyy-MM-dd HH:mm:ss
    - EEE, yyyy-MMM-dd
    - EEEE, yyyy-MMMM-dd
- Diwarnai biru: untuk mendapatkan nilai COMPONENT date, kita get dengan key berupa “contents”. Perhatikanlah bahwa apa

yang kita dapatkan adalah tipe data DATE. Ini termasuk tanggal dan waktu, walaupun format tampilannya tidak.




Pada contoh gambar layar tersebut, walaupun yang tampil hanya tanggal, kita juga mendapatkan waktu (yang sekaligus menunjukkan bahwa penulis sedang lembur untuk menyiapkan buku ini dirilis keesokan harinya).

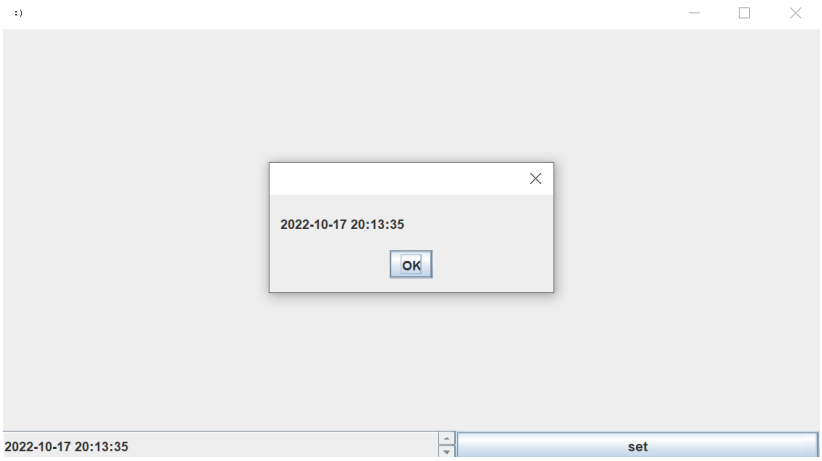
Sekarang, mari kita balik, dengan mengkonfigur COMPONENT date sesuai tanggal dan waktu saat ini, lewat penekanan button:

```
reset()

var d = component("date", "yyyy-MM-dd HH:mm:ss")

var b = component("button", "set")

add_s([d, b])

show()


event(b, fn() {

    var n = @
```

```
    config(d, "contents", n)
    message(n)
})
```

Penjelasan:

- Diwarnai hijau: untuk mendapatkan tanggal/waktu aktif, kita bisa gunakan @. Ini merupakan tipe data DATE.
- Diwarnai biru: untuk mengkonfigur, kita dapat menggunakan tipe DATE ataupun STRING. Dalam contoh ini, kita menggunakan DATE.




Untuk tipe berupa STRING, fungsi parts dapat digunakan. Fungsi ini mengkonversi DATE menjadi STRING (format: yyyyMMddHHmmss), seperti contoh berikut (diketikkan pada tab Interactive):

```
> parts(@)
"20221017202038"
```

Dengan demikian, baris yang diwarnai biru sebelumnya, dapat pula dituliskan sebagai:

```
config(d, "contents", parts(n))
```

Ingatlah bahwa DATE adalah tipe data di Singkong, yang dapat dituliskan sebagai: @ @Y @YY @YYY @YYYY @YYYYM @YYYYMM @YYYYMMD @YYYYMMDD @YYYYMMDDh @YYYYMMDDhh @YYYYMMDDhhm @YYYYMMDDhhmm @YYYYMMDDhhmms @YYYYMMDDhhmmss

Andaikata terdapat fungsi yang membutuhkan konversi dari DATE ke NUMBER, kita dapat menggunakan fungsi number.

**Kalender**

Bagaimana kalau kita perlu menampilkan kalender? Terdapat sebuah modul, yang sepenuhnya ditulis dengan bahasa Singkong, yaitu ui_calendar, yang pada saat buku ini ditulis, menyediakan fungsi-fungsi berikut:

```
create_calendar
create_calendar_basic
create_calendar_basic_compact
create_calendar_simple
create_calendar_simple_compact
```

Perbedaan utama diantara semua fungsi tersebut (kecuali create_calendar), adalah pada style yang digunakan dalam menampilkan kalender.

Sementara, yang sama diantara semua fungsi tersebut (juga kecuali create_calendar) adalah bahwa semua memanggil fungsi create_calendar :)

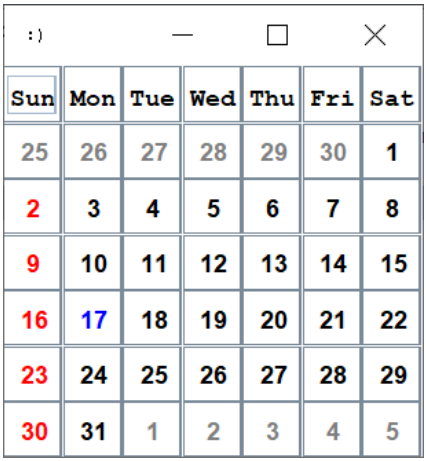
Untuk style yang sederhana, kita bisa gunakan create_calendar_simple seperti contoh berikut:

```
load_module("ui_calendar")
reset()
var d = part(@)
var g = create_calendar_simple(d[0], d[1])
add(g)
show()
```


Sebuah kalender, dengan style sederhana (font warna hitam default), termasuk warna untuk hari minggu (merah), tanggal saat ini (biru), dan tanggal bulan lalu/depan (abu-abu) digunakan.

Apabila kita membutuhkan tombol dengan margin yang lebih kecil, kita bisa gunakan create_calendar_simple_compact, seperti contoh berikut. Ubahlah ukuran frame untuk melihat perbedaannya.

```
load_module("ui_calendar")

reset()

var d = part(@)

var g =
create_calendar_simple_compact(d[0], d[1])

add(g)

show()
```




Ingin menampilkan kalender dengan style dasar? Fungsi create_calendar_basic dapat digunakan, seperti contoh berikut:

```
load_module("ui_calendar")

reset()
```

```
var d = part(@)
var g = create_calendar_basic(d[0], d[1])
add(g)
show()
```

Versi compact-nya, create_calendar_basic_compact, juga tersedia:

```
load_module("ui_calendar")
reset()
var d = part(@)
var g = create_calendar_basic_compact(d[0],
d[1])
add(g)
show()
```

Ingin melakukan sesuatu ketika tanggal diklik? Kita bisa gunakan fungsi create_calendar langsung, dengan mengatur sebagian dari key pemetaan yang digunakan. Perhatikanlah contoh berikut:

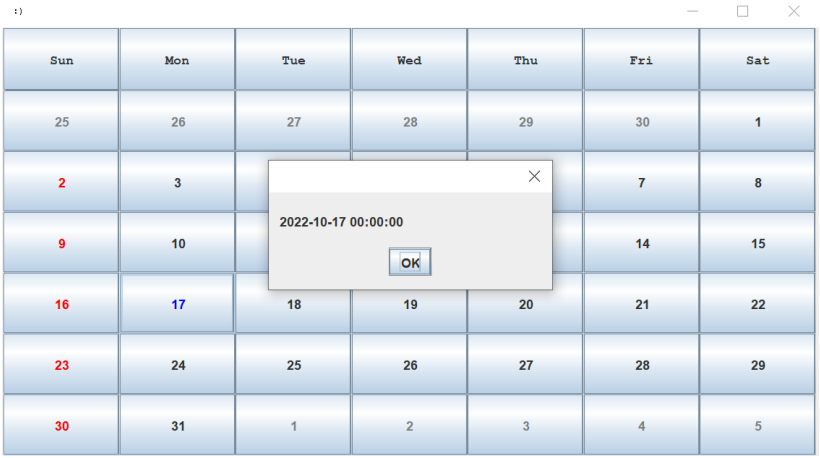
```
load_module("ui_calendar")
reset()
var h = fn(d, s, dow) {
    message(d)
    message(s)
    message(dow)
}
```

```
var d = part(@)
var data = {
    "handler": h,
    "prev_handler": h,
    "next_handler": h,
    "today_foreground": "blue",
    "weekend_days": [1],
    "weekend_foreground": "red",
    "prev_foreground": "gray",
    "next_foreground": "gray",
    "margin": [0, 0, 0, 0]
}
var g = create_calendar(d[0], d[1], data)
add(g)
show()
```

Penjelasan:

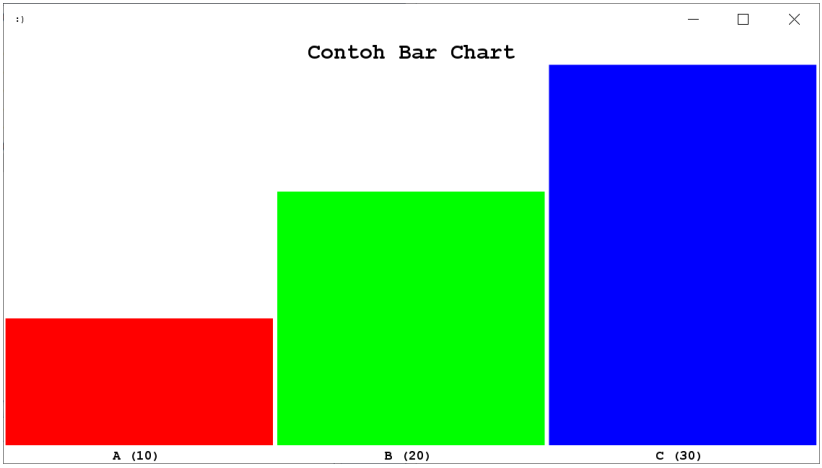
- Kita akan menggunakan fungsi create_calendar secara langsung. Artinya, kita bisa mengatur sepenuhnya sesuai yang didukung oleh fungsi ini.
- Diwarnai hijau: apabila tanggal dipilih (button ditekan), maka sebuah fungsi dengan tiga argumen diperlukan: DATE, representasi STRING dari tanggal, dan day of week (1=minggu). Dalam contoh ini, kita gunakan handler yang sama, baik untuk bulan aktif, bulan lalu, atau bulan berikutnya.

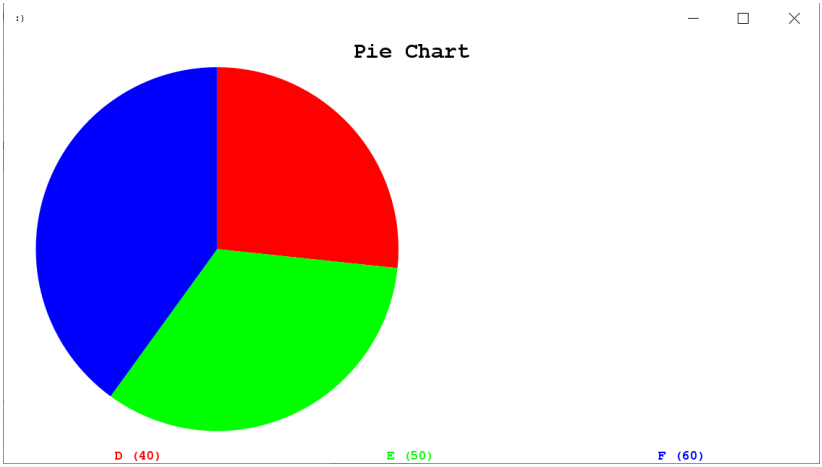
- Diwarnai biru: untuk mendapatkan ARRAY [year, month, day, hour, minute, second] dari sebuah DATE, kita dapat menggunakan fungsi part. Dalam contoh ini, kita membutuhkan year dan month.
- Diwarnai merah: kita perlu melewatkan sebuah HASH ketika menggunakan create_calendar. Untuk selengkapnya, kita dapat melihat source code dari modul ui_calendar, terutama terkait key dan value yang dapat diterima.



*Halaman ini sengaja dikosongkan*

## Contoh 9: Menampilkan barchart dan piechart




Untuk chart, Singkong menyediakan dua COMPONENT bawaan, yaitu barchart (gambar layar pertama) dan piechart. Sama seperti COMPONENT user interface lain di Singkong, keduanya juga dapat dibuat dengan satu baris kode (fungsi component), dikonfigur dengan fungsi config, dan bisa didapatkan konfigurasinya dengan fungsi get.

Yang perlu diperhatikan hanyalah Anda mungkin ingin:

- Mengatur warna latar
- Mengatur warna font judul
- Mengatur font yang digunakan
- Menentukan warna bar (pada barchart) atau potongan (pada piechart), beserta nilai dan warnanya.

Jadi, bab ini bukan bab yang panjang?

Benar. Hanya sisa 1 halaman lagi. Masing-masing contoh gambar layar tersebut adalah program yang tidak sampai 10 baris kode. Dan perbedaan antara pembuatan barchart dan piechart hanya pada 1 baris kode.

Mari kita lihat contoh barchart terlebih dahulu:

```
reset()

var bc = component("barchart", "")

config(bc, "foreground", "black")

config(bc, "background", "white")

config(bc, "font", ["monospaced", 1, 20])

config(bc, "text", "Contoh Bar Chart")

config(bc, "contents", [[10, "A (10)", "red"], [20,
"B (20)", "green"], [30, "C (30)", "blue"]])

add(bc)

show()
```

Penjelasan:

- Diwarnai hijau: pembuatan barchart sama seperti pembuatan COMPONENT lain.

- Diwarnai biru: kita mungkin perlu mengatur foreground (warna font pada judul), background (warna latar), font, ataupun text (judul).
- Diwarnai merah: apa yang ditampilkan pada chart adalah ARRAY dari ARRAY [nilai, label, dan warna].

Bedanya dengan pembuatan piechart? Terlepas dari nama variabel dan contoh ARRAY yang ditampilkan, perbedaannya hanya pada pemanggilan fungsi component:

```
reset()

var pc = component("piechart", "")

config(pc, "foreground", "black")

config(pc, "background", "white")

config(pc, "font", ["monospaced", 1, 20])

config(pc, "text", "Pie Chart")

config(pc, "contents", [[40, "D (40)", "red"], [50,
"E (50)", "green"], [60, "F (60)", "blue"]])

add(pc)

show()
```

Satu hal terakhir yang mungkin perlu disampaikan, terkait barchart dan piechart, adalah ARRAY yang ditampilkan tidak bisa didapatkan dengan get (dengan key: “contents”). Oleh karena itu, Anda mungkin ingin menggunakan variabel tersendiri.

*Halaman ini sengaja dikosongkan*

## Contoh 10: Image, draw, dan suara

Anda mungkin berpendapat: sudah sejauh ini, baru kita membahas bagaimana menampilkan gambar? Bukankah ini seharusnya di awal buku atau bahkan pada buku sebelumnya?

Penulis juga setuju. Hanya saja, karena image dan memainkan suara dapat terkait file di luar kode program, ada catatan yang perlu kita bahas tersendiri.

Tapi, mari kita jadikan ini juga bab yang singkat. Mari mulai dari yang paling sederhana: kita memiliki sebuah file gambar (contoh: "singkong.png" di direktori aktif) yang ingin ditampilkan:

```
reset()

var g = component("image", "singkong.png")

add(g)

show()
```


Yang perlu kita buat adalah sebuah COMPONENT image (diwarnai hijau). Nama COMPONENT, dalam hal ini adalah nama file di

filesystem (seperti pada contoh) atau nama file dalam arsip jar interpreter.

Dalam arsip jar interpreter? Ketika mendistribusikan aplikasi yang Anda buat, yang mungkin mengandung file-file gambar, suara, file teks, dan lainnya, cara yang nyaman barangkali adalah dengan membundel interpreter Singkong (Singkong.jar) dan semua file yang dibutuhkan, sebagai satu file jar yang dapat dijalankan. Dengan demikian, file-file gambar misalnya, berada dalam file arsip jar interpreter tersebut.

Ketika kita perlu mengkonfigur agar gambar dibaca bukan dari filesystem, melainkan dalam arsip jar interpreter, kita akan memanggil fungsi config dengan argumen ketiga berupa true.

Contoh:

```
var c = component("image", "")

config(c, "contents", "file.png", true)
```

Pada contoh tersebut, “file.png” harus ditempatkan pada direktori /resource di dalam arsip jar interpreter.

Untuk informasi selengkapnya, yang tidak kita bahas dalam buku ini, Anda mungkin ingin membaca bab *Distribusi Aplikasi* pada buku gratis: *Mengenal dan Menggunakan Bahasa Pemrograman Singkong*.

**Memainkan suara**

Singkong menyediakan dua fungsi untuk memainkan suara: beep dan play_sound. Yang pertama tidak membutuhkan argumen apapun, cukup:

beep()

Maka, suara ‘beep’ yang dikonfigur pada sistem akan dimainkan.

Untuk memainkan suara dari file, baik yang tersimpan pada filesystem ataupun di dalam arsip jar interpreter, kita menggunakan play_sound. Fungsi ini membutuhkan dua argumen:

- STRING: nama file pada filesystem atau /resource/<nama_file> dalam arsip jar interpreter.
- BOOLEAN: true apabila file suara dibaca dari arsip jar interpreter.

Contoh memainkan file sound.wav di dalam direktori aktif:

play_sound("sound.wav", false)

Contoh memainkan file /resource/sound.wav dalam arsip jar interpreter:

play_sound("sound.wav", true)

**draw**

Butuh untuk menggambar sendiri pada sebuah COMPONENT? Mari gunakan COMPONENT draw. Untuk membuatnya, kita perlu menentukan ukuran lebar dan tinggi dalam pixel, sebagai nama COMPONENT (default: 100, 100).

Sebagai contoh:

```
reset()

var dr = component("draw", "500, 500")

add(dr)
```

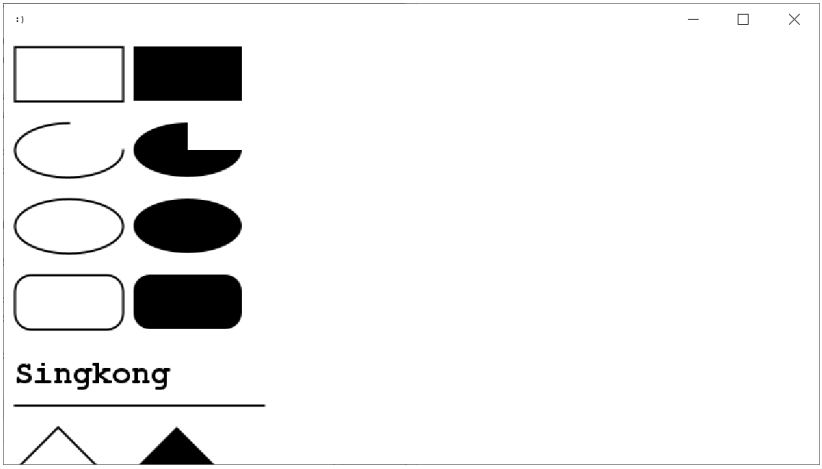
Selanjutnya, untuk menggambar, kita bisa menggunakan fungsi-fungsi berikut:

```
draw_width
draw_rect
fill_rect
draw_arc
fill_arc
draw_oval
fill_oval
draw_round_rect
fill_round_rect
draw_string
draw_line
draw_polygon
fill_polygon
draw_polyline
draw_read
draw_write_png
draw_write_jpg
draw_write_bmp
draw_get_pixel
draw_set_pixel
```

Contoh:

```
reset()
var dr = component("draw", "500, 500")
add(dr)

config(dr, "foreground", "black")
config(dr, "background", "white")

draw_width(dr, 2)

draw_rect(dr, 10, 10, 100, 50)
fill_rect(dr, 120, 10, 100, 50)

draw_arc(dr, 10, 80, 100, 50, 90, 270)
fill_arc(dr, 120, 80, 100, 50, 90, 270)

draw_oval(dr, 10, 150, 100, 50)
fill_oval(dr, 120, 150, 100, 50)

draw_round_rect(dr, 10, 220, 100, 50, 30, 30)
fill_round_rect(dr, 120, 220, 100, 50, 30, 30)

config(dr, "font", ["monospaced", 1, 30])
```

```
draw_string(dr, "Singkong", 10, 320)

draw_line(dr, 10, 340, 240, 340)

draw_polygon(dr, [10, 50, 90], [400, 360, 400])
fill_polygon(dr, [120, 160, 200], [400, 360, 400])

draw_polyline(dr, [10, 50, 90], [480, 440, 480])

show()
```




Contoh lain adalah sebuah program untuk menggambar dengan mouse dan menambahkan teks, membuka/menyimpan file, mengubah nilai pixel, dalam kurang dari 200 baris kode, dimana contohnya bisa dibaca dalam dokumentasi Singkong (atau buku gratis: *Mengenal dan Menggunakan Bahasa Pemrograman Singkong).*

## Contoh 11: Komponen user interface dan contoh lain

Akhirnya, kita sampai di bab terakhir, sebelum pembahasan bonus. Kita akan membahas beberapa COMPONENT yang belum dibahas:

| COMPONENT | Penjelasan |
|---|---|
| mask | Dapat digunakan untuk input teks dengan format tertentu.<br><br>Format:<br>#: number<br>U: upper case<br>L: lower case<br>A: karakter atau number<br>?: karakter<br>*: apa saja<br>H: karakter heksadesimal |
| password | Digunakan untuk input berupa password |
| radio | Memilih dengan radio button |
| tab | Bekerja dengan tab |
| view | Menampilkan HTML |

Contoh kode COMPONENT mask, dimana kita bisa mengisikan input seperti: (123) 456-789, tanpa harus menekan tombol (, ), spasi, atau -. Karakter lain (misal huruf) akan diabaikan.

```
reset()

var s = component("mask", "(###) ###-###")

add_s(s)

show()
```

Contoh kode COMPONENT password, dimana karakter yang diisikan diganti dengan simbol tertentu sehingga apa yang diisikan tidak terlihat:

```
reset()

var p = component("password", "test")

add_s(p)

show()
```

Untuk pilihan ya atau tidak, kita bisa menggunakan checkbox. Untuk memilih dari daftar, kita bisa menggunakan combobox. Untuk memilih dari beberapa, namun yang bisa langsung terlihat, radio dapat digunakan. Sebagai contoh:

```
reset()

var r = component("radio", "Radio Button")

add_s(r)

show()
```

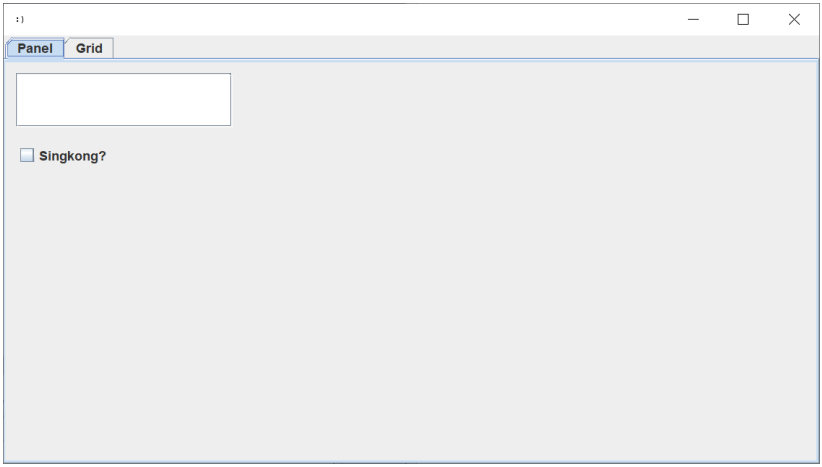
Contoh tersebut tentu tidak mencerminkan memilih dari beberapa sebagaimana disebutkan sebelumnya. Bahkan, dengan kode berikut, semuanya dapat dipilih:

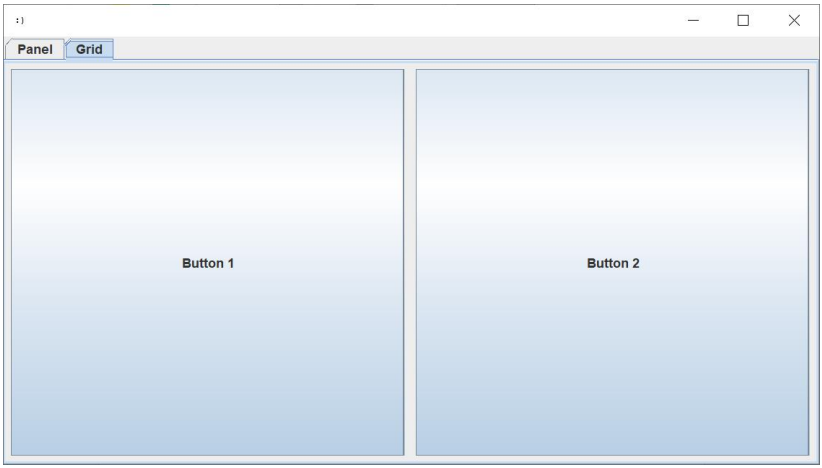
```
reset()

var r1 = component("radio", "Singkong")

var r2 = component("radio", "Pangsit")

var r3 = component("radio", "Perkedel")

add_s([r1, r2, r3])
```

```
show()
```

Apabila yang kita inginkan adalah user hanya dapat memilih satu dari sekian pilihan, kita dapat menggunakan fungsi radio_group, yang membuat set mutual-exclusion dari ARRAY COMPONENT radio. Contoh sebelumnya dapat diperbaiki dengan menambah sebaris kode (diwarnai hijau):

```
reset()
var r1 = component("radio", "Singkong")
var r2 = component("radio", "Pangsit")
var r3 = component("radio", "Perkedel")
add_s([r1, r2, r3])
radio_group([r1, r2, r3])
show()
```






Perlu bekerja dengan banyak COMPONENT dalam satu frame? Kita dapat menggunakan COMPONENT tab, dimana sejumlah COMPONENT dapat ditampilkan pada tab masing-masing. Untuk isinya, kita bisa menggunakan panel atau grid. Contoh:

```
reset()
var e = component("edit", "")
var c = component("checkbox", "Singkong?")
var p = component("panel", "Panel")
panel_add(p, e, 10, 10, 200, 50)
panel_add(p, c, 10, 70, 100, 30)

var g = component("grid", "Grid")
var b1 = component("button", "Button 1")
var b2 = component("button", "Button 2")
grid_add(g, b1, 0, 0, 1, 1, 0.5, 1.0, 3, 0, 5, 5, 5, 5)
```

```
        grid_add(g, b2, 1, 0, 1, 1, 0.5, 1.0, 3, 0,
5, 5, 5, 5)


        var t = component("tab", "")

        tab_add(t, p)

        tab_add(t, g)


        add(t)

        show()
```

Contoh tersebut menggunakan fungsi tab_add untuk menambahkan panel dan grid ke tab. Untuk menghapus dari tab, fungsi tab_remove dapat digunakan. Untuk menghapus semua tab, gunakanlah tab_clear.

Untuk contoh terakhir pembuatan COMPONENT, kita akan menampilkan kode HTML (sampai versi tertentu) dalam COMPONENT view:

```
reset()

var v = component("view",
"<b>Singkong</b><br>Programming")

add(v)

show()
```

**Catatan**

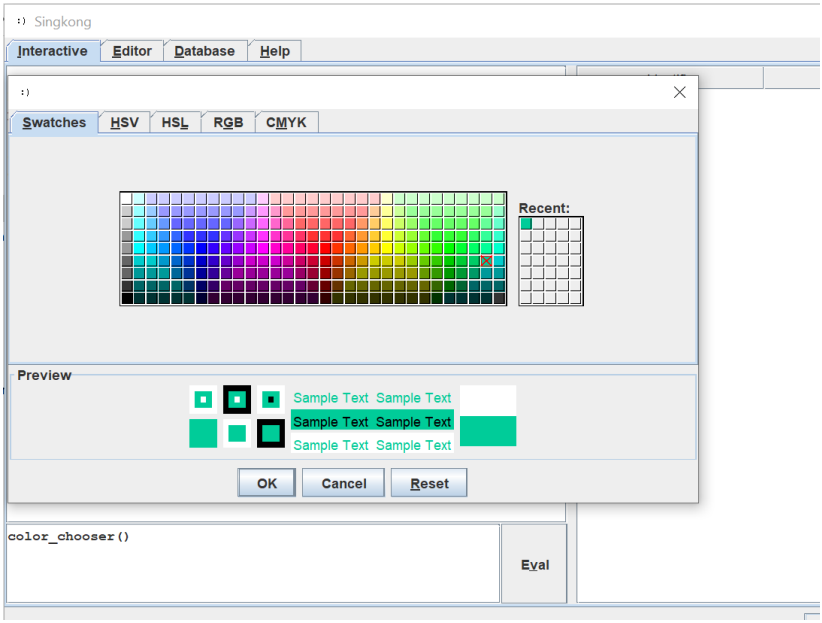
Sebelum menutup pembahasan, beberapa catatan berikut mungkin berguna ketika bekerja dengan GUI.

Warna

Anda mungkin penasaran ketika bekerja dengan warna dan kita memberikan warna seperti red, green, blue, dan beberapa warna yang namanya telah didefinisikan. Bagaimana dengan warna lain?

Pertama, selain nama, kita juga bisa memberikan dengan STRING berupa angka tertentu, yang bisa didapatkan dengan fungsi color_chooser.

Aktiflah di tab Interactive, dan jalankanlah statement color_chooser():



Setelah selesai, tekanlah tombol OK dan sebuah NUMBER akan dikembalikan. Misal sesuai warna dalam gambar layar sebelumnya, nilai berikut dikembalikan: -16724839.

Selanjutnya, kita bisa melewatkan nilai tersebut ketika mengkonfigur background atau foreground. Sebagai contoh:

```
reset()

var b = component("button", "Hello")

config(b, "background", "-16724839")

add_s(b)

show()
```

## Font

Dalam beberapa contoh, kita mengkonfigur font COMPONENT. Untuk mendapatkan daftar font yang tersedia, kita bisa gunakan fungsi fonts. Tapi ini mengembalikan nama. Sementara, kita memberikan ARRAY seperti contoh berikut (misal pada contoh barchart):

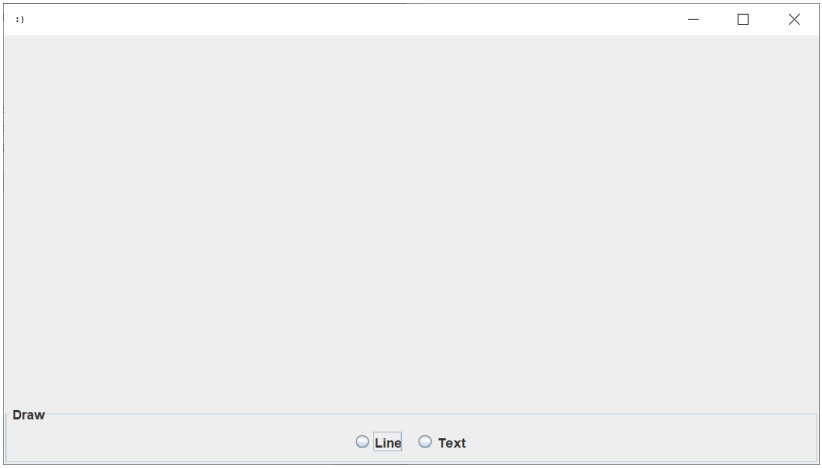
```
config(bc, "font", ["monospaced", 1, 20])
```

Dalam contoh ini, 1 adalah bold. Nilai lain yang mungkin adalah 0=plain, 2=italic, dan 3=bold and italic. Sementara 20 adalah ukuran font.

## Border

Terkadang, sebuah COMPONENT mungkin ingin diberikan border, yang dapat berupa judul atau keterangan tertentu. Contoh cuplikan pada program menggambar (pada baris yang diwarnai hijau):

```
reset()
var gd = component("grid", "")
config(gd, "border", "Draw")
var rl = component("radio", "Line")
var rt = component("radio", "Text")
radio_group([rl, rt])
grid_add(gd, rl, 0, 0, 1, 1, 0, 0, 0, 1, 4, 4, 4, 4)
grid_add(gd, rt, 1, 0, 1, 1, 0, 0, 0, 1, 4, 4, 4, 4)
add_s(gd)
show()
```




Konfigurasi COMPONENT

Key yang dapat digunakan dalam konfigurasi COMPONENT adalah:

- border: STRING: border dengan judul.

- enabled: BOOLEAN.
- visible: BOOLEAN.
- focus: BOOLEAN.
- foreground/background: STRING: nama warna atau nilai RGB.
- font
- text: STRING, label dari button/checkbox/label/radio, item terpilih combobox, title dari barchart/piechart.
- active: BOOLEAN (checkbox/radio), NUMBER (indeks item terpilih combobox/tab/table), NUMBER (indeks mnemonic button), NUMBER (text, hanya dapat diterapkan pada grid).
- contents: STRING (isi dari edit/mask/password/text/view), STRING (file untuk image), ARRAY of ARRAY (table), ARRAY (combobox), DATE/STRING (date), NUMBER (progress/spin), ARRAY of ARRAY ([NUMBER (value), STRING (label), STRING (nama warna or nilai RGB)] untuk barchart/piechart).
- margin: ARRAY dari NUMBER [top, left, bottom, right], hanya berlaku untuk button pada grid.

Key tersebut berlaku untuk config ataupun get, hanya saja, seperti halnya pada barchart atau piechart, tidak semua yang dikonfigur bisa dibaca kembali. Oleh karena itu, dapat di-assign ke sebuah variabel.

Selain itu, apabila key tidak didukung, tidak akan terjadi error, dan akan diabaikan.

Fungsi config sendiri dapat diberikan argumen keempat, dimana:

- Dilewatkan BOOLEAN, apabila true (default adalah false); dan
- Key adalah “contents”; dan
- Tipe COMPONENT adalah: edit/image/password/text/view; dan
- Tipe value adalah STRING; maka

Akan membaca /resource/<value> dalam file jar interpreter.

Jumlah kolom COMPONENT text

Terkadang, kita perlu menentukan jumlah kolom dalam COMPONENT text. Untuk itu, kita bisa mengkonfigur dengan key "active". Akan tetapi, sebagaimana disinggung sebelumnya pada konfigurasi COMPONENT, ini hanya berlaku apabila kita menggunakan grid. Contoh:

```
reset()
var g = component("grid", "")
var t1 = component("text", "Text 1")
config(t1, "active", 10)
var t2 = component("text", "Text 2")
grid_add(g, t1, 0, 0, 1, 1, 1, 1, 0, 0)
grid_add(g, t2, 1, 0, 1, 1, 1, 1, 0, 0)
add(g)
show()
```

Table ke teks dan HTML

Untuk mendapatkan versi teks dan HTML dari sebuah table, kita bisa menggunakan fungsi-fungsi table_to_text dan table_to_html dari modul ui_util. Contoh:

```
load_module("ui_util")
reset()
```

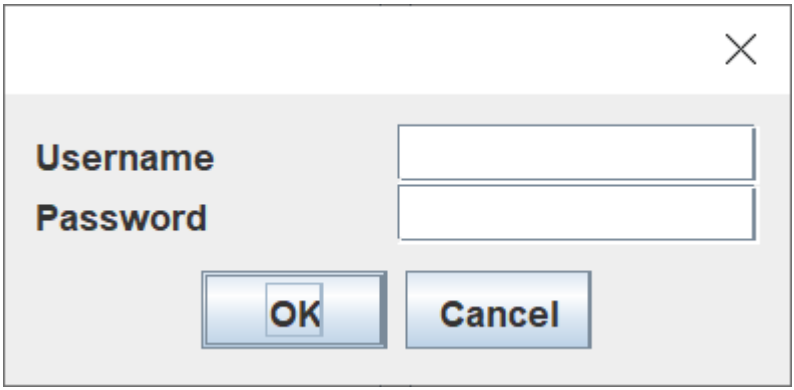
```
var t = component("table", "A,B,C")
table_left(t, 0)
table_center(t, 1)
table_right(t, 2)
var e = component("edit", "")
config(e, "font", ["monospaced", 0, 10])
table_add_fill(t, 0)
var tt = table_to_text(t, [10, 20, 30])
config(e, "contents", tt)
add([t, e])
println(tt)
show()
```

Dialog login

Ingin membuat dialog login yang mudah? Kita bisa gunakan fungsi login_dialog. Cara termudah adalah memanggil fungsi tanpa argumen apapun, dan kita akan mendapatkan label dan text username dan password default. Contoh:

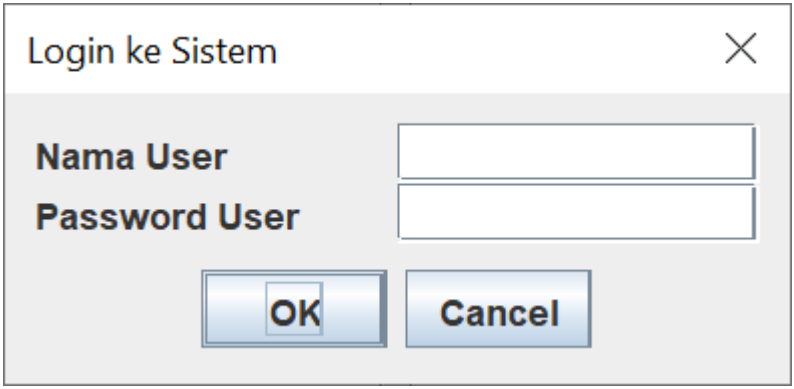
```
var d = login_dialog()
print(d)
```

Apabila pengguna memilih Cancel, maka [] dikembalikan. Selebihnya, ARRAY STRING [username, password] dikembalikan.




Ingin mengatur judul, label username, dan label password? Kita bisa memberikannya sebagai argumen pertama, kedua, dan ketiga, seperti contoh berikut:

```
var d = login_dialog("Login ke Sistem", "Nama User",
"Password User")

print(d)
```



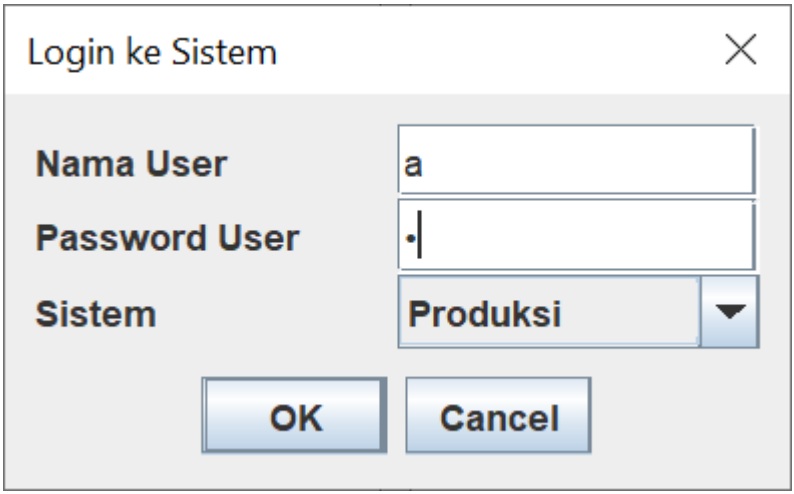

Sekarang, andaikata user dapat login ke beberapa sistem, yang mana dapat dipilih di dialog login. Mari kita gunakan argumen keempat dari

fungsi login_dialog ini, dimana berupa ARRAY dimana elemen pertama adalah label. Contoh:

var d = login_dialog("Login ke Sistem", "Nama User", "Password User", ["Sistem", "Produksi", "Uji"])

print(d)

Dalam hal ini, fungsi akan mengembalikan ARRAY dengan tiga elemen: [username, password, option].




Apabila nama user adalah a, password user adalah b, dan sistem adalah Produksi, maka ketika tombol OK ditekan, fungsi akan mengembalikan: ["a", "b", "Produksi"].

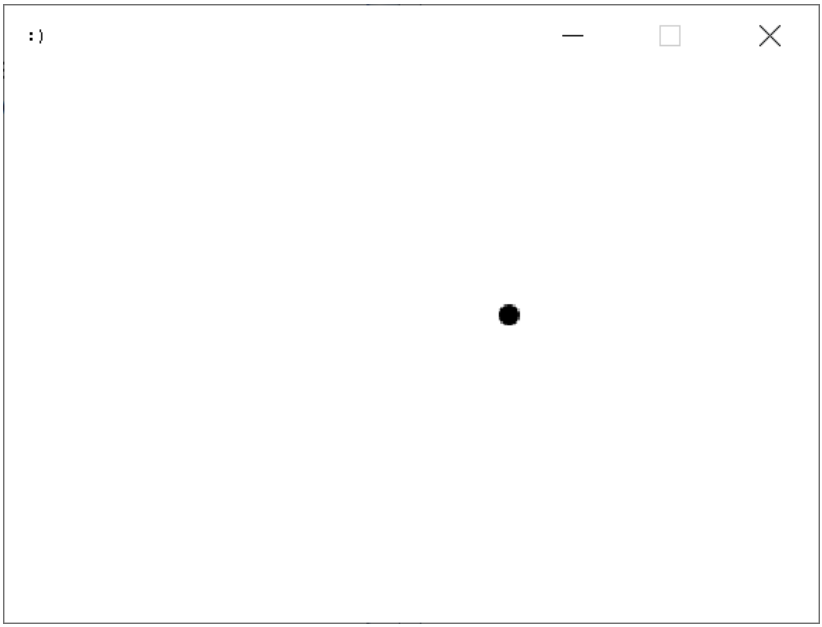
Catatan tentang dialog login menutup bab ini.

*Halaman ini sengaja dikosongkan*

## Bonus: Bola pantul (perbaikan)

Di buku gratis *Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar aplikasi GUI*, terdapat pembahasan bonus berupa bola pantul. Program berfungsi, dimana setiap 100 milidetik, COMPONENT label ditambahkan ke panel, dengan posisi yang bergeser sebanyak 10 pixel.

Di pembahasan ini, kita menggambar (fill) sebuah oval pada COMPONENT draw. Bagi Anda yang kreatif, tentunya bisa menggambar lebih baik.




Contoh kode berikut telah diubah agar interval diubah ke 10 milidetik dan posisi bergeser sebanyak 2 pixel.

```
reset()

var w = 400
```

```
var h = 300
var d = [w/2, h/2, 2, 2]
var s = 30
var p = component("draw", "400,300")
config(p, "background", "white")
add(p)
size(w, h)
resizable(false)
show()
timer(10, fn() {
    var x = d[0]
    var y = d[1]
    var dx = d[2]
    var dy = d[3]
    var xs = x + s
    var ys = y + s
    if (xs >= w | xs < s) {
        var dx = dx * -1
    }
    if (ys >= h | ys < s) {
        var dy = dy * -1
    }
    config(p, "foreground", "white")
```

```
    fill_oval(p, x-2, y-2, 14, 14)
    var x = x + dx
    var y = y + dy
    set(d, 0, x)
    set(d, 1, y)
    set(d, 2, dx)
    set(d, 3, dy)
    config(p, "foreground", "black")
    fill_oval(p, x, y, 10, 10)
})
```

Berikut adalah perbedaan kode program dibandingkan dengan versi sebelumnya:

```
4c4
< var d = [w/2, h/2, 10, 10]
---
> var d = [w/2, h/2, 2, 2]
6,7c6,7
< var b = component("label", "0")
< var p = component("panel", "")
---
> var p = component("draw", "400,300")
> config(p, "background", "white")
12c12
< timer(100, fn() {
---
> timer(10, fn() {
24a25,26
>     config(p, "foreground", "white")
>     fill_oval(p, x-2, y-2, 14, 14)
31c33,34
<     panel_add(p, b, x, y, s, s)
---
>     config(p, "foreground", "black")
>     fill_oval(p, x, y, 10, 10)
```

*Halaman ini sengaja dikosongkan*

# Daftar Pustaka

Noprianto. (2020). Mengenal dan Menggunakan Bahasa Pemrograman Singkong. PT. Stabil Standar Sinergi.

Noprianto, Iskandar, K., & Soewito, B. (2022). Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI. PT. Stabil Standar Sinergi.

**Buku ini berisi sejumlah contoh source code dan penjelasan langkah demi langkah yang mudah dipahami untuk membuat program komputer yang dilengkapi Graphical User Interface (GUI), dengan bahasa pemrograman Singkong.**

**Contoh-contoh yang dibahas mencakup berbagai topik lanjutan dalam pembuatan GUI, sehingga, setelah membaca buku ini, Anda akan mahir bekerja dengan GUI. Oleh karena itu, agar dapat mengikuti pembahasan dengan nyaman, pernah membuat aplikasi GUI dengan bahasa Singkong akan sangat membantu.**

**Melengkapi contoh program, buku ini juga membahas pengantar untuk analisis kebutuhan pengembangan perangkat lunak dan pemodelan matematika dalam ilmu komputer.**

Dr. Noprianto mengembangkan bahasa pemrograman Singkong dan interpreternya sejak akhir 2019.

Beliau menyukai pemrograman, dan mendirikan serta mengelola perusahaan pengembangan software dan teknologi Singkong.dev (PT. Stabil Standar Sinergi).

Noprianto menyelesaikan pendidikan doktor ilmu komputer dan telah menulis beberapa buku pemrograman (termasuk Python, Java, dan Singkong).

Buku dan softwarenya dapat didownload dari https://nopri.github.io

Dr. Wartika menyelesaikan pendidikan S1 di bidang Manajemen Informatika, S2 Teknik Informatika, dan S3 Computer Science.

Saat ini, selain sebagai dosen, beliau juga menjabat struktural sebagai Ketua Program Studi Manajemen Informatika dan Sekretaris Program Studi Sistem Informasi di Fakultas Teknik dan Ilmu Komputer UNIKOM.

Dalam melaksanakan tri dharma perguruan tinggi, selain aktif melaksanakan pengajaran dan penelitian serta pengabdian kepada masyarakat, beliau juga berkontribusi dalam menyelesaikan proyek-proyek di bidang sistem informasi.

Dr. Wartika dapat dihubungi lewat email wartika31@yahoo.co.id.

Dr. Ford saat ini adalah Ketua Jurusan Program Doktor Ilmu Komputer Binus University. Beliau menyelesaikan pendidikan S1 dalam bidang Matematika, S2 dan S3 dalam bidang Ilmu Komputer, semuanya dari Universitas Indonesia. Pendidikan profesi Insinyur diperoleh dari Binus University.

Dalam bidang komunitas ilmiah, Dr. Ford juga adalah Ketua IEEE Computer Society Indonesia Chapter, Ketua IIAI Region ASEAN dan Advisory Board member dari Sridhar University India. Beliau juga menjadi visiting professor di berbagai universitas di luar negeri, dan telah mendapatkan berbagai hibah internasional.

Dr. Ford telah menghasilkan lebih dari 289 publikasi yang terindeks SCOPUS yang tersebar di berbagai jurnal Q1, Q2, serta proceeding.

**singkong.dev**
Alamat: Puri Indah Financial Tower
Lantai 6, Unit 0612
Jl. Puri Lingkar Dalam Blok T8
Kembangan, Jakarta Barat 11610
Email: info@singkong.dev

ISBN 978-602-52770-4-7
9 786025 277047
Rp. 50.000