BAB II
FUNCTIONAL PROGRAMMING

PENGERTIAN

Prinsip utama dari functional programming (FP) adalah bahwa nilai suatu ekspresi hanya tergantung kepada nilai sub-ekspresinya. Nilai ekspresi seperti a + b hanya tergantung kepada nilai a dan nilai b. Prinsip ini menghilangkan efek samping dalam ekspresi sehingga FP dapat disebut sebagai pemrograman tanpa assignment. Karena tidak adanya efek samping tersebut , sebuah ekspresi memiliki nilai yang sama setiap kali dievaluasi. Karakteristik lain dari bahasa functional adalah bahwa user tidak perlu khawatir terhadap manajemen penyimpanan data karena:

- adanya manajemen penyimpanan (data) implisit.
- terdapat operasi-operasi built-in untuk alokasi penyimpanan data sesuai keperluan.
- penyimpanan (storage) yang akan menyebabkan kegagalan pengaksesan secara otomatis tidak dialokasikan.
- ketiadaan kode eksplisit untuk dealokasi membuat program lebih sederhana dan lebih ringkas.

Konsekuensi pendekatan tersebut adalah bahwa implementasi bahasa ini harus membuat kumpulan sampah ,"garbage collection", untuk memperoleh kembali storage yang telah menjadi storage tak terkases.

Akhirnya, FP memperlakukan fungsi sebagai "first-class citizen" :

- Function are first-class values.
- Fungsi-fungsi memiliki status sama dengan nilai-nilai yang lain.
- Sebuah fungsi dapat berupa ekspresi, fungsi dapat di-pass sebagai argumen dan fungsi dapat dipakai dalam struktur data.

MENGAPA LISP ?

Lisp telah tersebar luas dan merupakan salah satu bahasa pilihan untuk aplikasi seperti artificial intelligence.

Bahasa fungsional pada umumnya, dan LISP pada khususnya, memainkan peranan penting dalam definisi bahasa. Sebuah definisi bahasa harus ditulis ke dalam notasi-notasi, disebut meta-bahasa(meta-language) atau bahasa yang didefinisikan (defining language), dan bahasa yang didefinisikan cenderung menjadi fungsional. Dalam kenyataannya, implementasi LISP pertama dimulai, ketika LISP digunakan untuk mendefinisikan dirinya sendiri.

Konsep dasar FP berasal dari LISP. LISP dirancang oleh John McCarthy tahun 1958, mungkin LISP adalah bahasa tertua kedua, setelah fortran. Diyakini LISP sebagai penyedia pertama rekursi, kelas-utama fungsi, garbage collection, dan sebuah definisi

bahasa formal (dalam LISP itu sendiri). Implementasi LISP juga mendorong ke arah lingkungan pemrograman terintegrasi, yang menggabungkan editor, interpreter dan debugger.

Dengan banyak keuntungan tersebut, mengapa tidak semua orang menggunakan LISP ?

- Pertama, LISP memiliki sintaks unik, LISP memiliki banyak tanda kurung ().
- Kedua, program-program LISP tidak bertipe;LISP tidak menghubungkan type dengan ekspresi.
- Ketiga, implementasi LISP pada masa lalu tidak efisien. Pada awal tahun 1960-an, LISP sangat lambat untuk perhitungan numerik.

EKSPRESI PREFIX DENGAN TANDA KURUNG

Notasi yang digunakan dalam LISP adalah notasi prefiks dengan tanda kurung. Aritmetika seperti 5 * 7 ditulis sebagai

```
(* 5 7)
35
```

Sintaks ekspresi dalam LISP adalah

```
(E1 E2 E3 ... Ek)
```

Di sini, ekspresi E1 menyatakan sebuah operator yang diaplikasikan ke nilai-nilai E2,...,Ek. Urutan evaluasi ekspresi E1, E2, ..., Ek tidak dispesifikasikan; Bagaimanapun, semua nilai subekspresi akan dihitung sebelum nilai E1 diaplikasikan ke operannya.

Ekspresi aritmetika dengan beberapa operator dapat diterjemahkan ke dalam LISP dengan mengikuti susunan subekspresinya. Ekspresi 4+5*7 adalah jumlah dari 4 dan 5*7:

```
(+ 4 (* 5 7))
39
```

QUOTING

Quoting diperlukan untuk memperlakukan ekspresi sebagai data. Item yang di-quot menghitung untuk item itu sendiri. Sintaks yang di-quot akan dijelaskan sebagai simbol. Symbol adalah obyek dengan ejaan. Quoting digunakan untuk memilih apakah ejaan diperlakukan sebagai simbol atau sebagai nama variabel.

Item dapat di-quot dengan 2 cara :

```
(quote <item>)
<item>
```

DEFINISI FUNGSI

Berikut ini salah satu cara mendefinisikan fungsi :
(defun (<nama fungsi> <peubah fungsi>) <ekspresi>)

Sebagai contoh

```
(defun (kuadrat x) (* x x))
kuadrat
(kuadrat 5)
25
```

Fungsi tersebut menghubungkan nilai fungsi dengan nama kuadrat. Fungsi lalu mengambil sebuah peubah dan mengalikan peubah dengan peubah itu sendiri.

Mengaitkan nama fungsi dengan nilai fungsi lebih jelas pada sintaks berikut

```
(defun <nama fungsi> <nilai fungsi>)
```

Untuk menggunakan sintaks tersebut, diperlukan notasi untuk nilai fungsi. LISP menyediakan notasi berikut:

```
(lambda ( <peubah fungsi>) <ekspresi>)
```

untuk nilai fungsi tak bernama. Jadi, fungsi dengan peubah x dan fungsinya (* x x) ditulis

```
(lamda (x) (* x x))
```

Kuadrat kini dapat didefinisikan sebagai

```
(defun kuadrat (lambda (x) (* x x)))
kuadrat
```

Fungsi tanpa nama juga dapat dianggap sebagai operator dalam ekspresi

```
((lambda (x) (* x x))5)
25
```

CONDITIONAL

Nilai boolean benar (true) dan salah (false) ditulis berturut-turut T dan NIL. Predikat adalah ekspresi yang akan menghasilkan true atau false bila dievaluasi. Ekspresi kondisional memiliki 2 macam bentuk :

```
(if P E1 E2)  : jika P maka E1 lainnya E2
```

Bentuk yang lain adalah berupa pilihan sekuensial :

```
(cond (P1 E1)  ; jika P1 maka E1
 ...
 ...
(Pk Ek)   ; jika Pk maka Ek
(T Ek+1)  ; lainnya Ek+1
```

Conditional diperlukan untuk fungsi rekursi, seperti definisi fungsi faktorial berikut :

```
(defun faktorial (n)
(cond ((= n 0) 1)
        (T (* n (faktorial (- n 1)))
))
```