UNIVERSITAS
ATMA JAYA YOGYAKARTA
serviens in lumine veritatis

Dasar Pemrograman (INFT06204)

Minggu 1
Pengantar Algoritma

ALGORITHMS

An *algorithm* is a sequence of discrete actions that when followed, will result in achieving some goal or solving some problem.

ALGORITHMS

Chocolate Chip[1] Cookie Recipe

Ingredients

- 1 cup melted butter
- 2 cups brown sugar
- 2 eggs
- 3 cups flour
- 1 teaspoon baking powder
- 1 teaspoon baking soda
- 2 cups chocolate chips

Directions

1. Preheat the oven to 375 degrees F.
2. Line a cookie sheet with parchment paper
3. In a bowl, stir together the butter, brown sugar and eggs.
4. In a separate bowl, combine the flour, baking powder and baking soda. Gradually combine with the sugar mixture.
5. Add the chocolate chips
6. Fill the cookie sheet with one-spoonful drops of the cookie dough.
7. Bake dough for 9 minutes
8. Cool for five minutes before removing from cookie sheet.

ALGORITHMS

- Computer science
  - any well-defined sequence of actions that takes a set of values as input and produces some set of values as output
  - Computer programmers refer to the meaning of an action as the ***semantics*** of an action. The phrase ***semantics*** refers to the meaning of the actions that occur in an algorithm.

SOFTWARE AND PROGRAMMING LANGUAGES

Computer software, also referred to as a *program*, provides the instructions for telling a computer the algorithm that it should follow to achieve some goal.

A programming language is a language that is designed to precisely and compactly express computational algorithms.

ACTIONS

- Involves knowing what actions a computer can take and also knowing what actions a computer cannot take! Consider, for example

Fuel-Efficient Travel Algorithm

1. Sprint southwest for 50 yards
2. Jump 2440 miles
3. Land in Los Angeles

Name Binding

How to write a name binding. The value on the right side of the arrow is bound to the identifier on the left of the arrow.

Name Binding
IDENTIFIER ← EXPRESSION

1. X ← 3
2. Y ← 4
3. Z ← 3 + 4

Name Binding

Proper Naming

An element is well named if the name descriptively and correctly reflects the central essence of the element.

Two Gas Cost Algorithms

```
1. DollarsPerGallon ← 3.75
2. TankCapacity ← 10
3. DollarsToFill ← TankCapacity x
   DollarsPerGallon
```

```
1. Cents ← 3.75
2. Size ← 10
3. Money ← Size x Cents
```

Convert from USD to BRL Algorithm

1. USD ← 1000
2. ExchangeRate ← 205.5
3. BRL ← USD × ExchangeRate

Name Binding

State

The *computational state* of a program is the collection of name bindings that are active at any single point in time.

1. $X \leftarrow 3$
2. $X \leftarrow 4$
3. $X \leftarrow X + X$

$X \leftarrow 3$

The state is now {X = 3}

2. $Y \leftarrow X + 4$

The state is now {X = 3 and Y = 7}

3. $Y \leftarrow X + Y$

The state is now {X = 3 and Y = 10}

4. $X \leftarrow X * Y$

The state is now {X = 30 and Y = 10}

Convert from Celsius to Fahrenheit Algorithm

1. Celsius ← 33.5
2. Fahrenheit ← Celsius * 9
3. Fahrenheit ← Fahrenheit / 5
4. Fahrenheit ← Fahrenheit + 32

Convert from Degrees Celsius to Fahrenheit

1. Celsius ← 33.5
   The state is now {Celsius = 33.5}
2. Fahrenheit ← Celsius * 9
   The state is now {Celsius=33.5 and Fahrenheit=301.5}
3. Fahrenheit ← Fahrenheit / 5
   The state is now {Celsius=33.5 and Fahrenheit=60.3}
4. Fahrenheit ← Fahrenheit + 32
   The state is now {Celsius=33.5 and Fahrenheit=92.3}

Selection

Control flow is a computational term that refers to the specific order in which the individual actions of a computer program are executed.

Control flow statements is flexibility is supported in programming languages by Element.

Selection statement is a control flow statement that allows a computer to make choices regarding whether certain actions should be performed.

Selection

One-way Statement

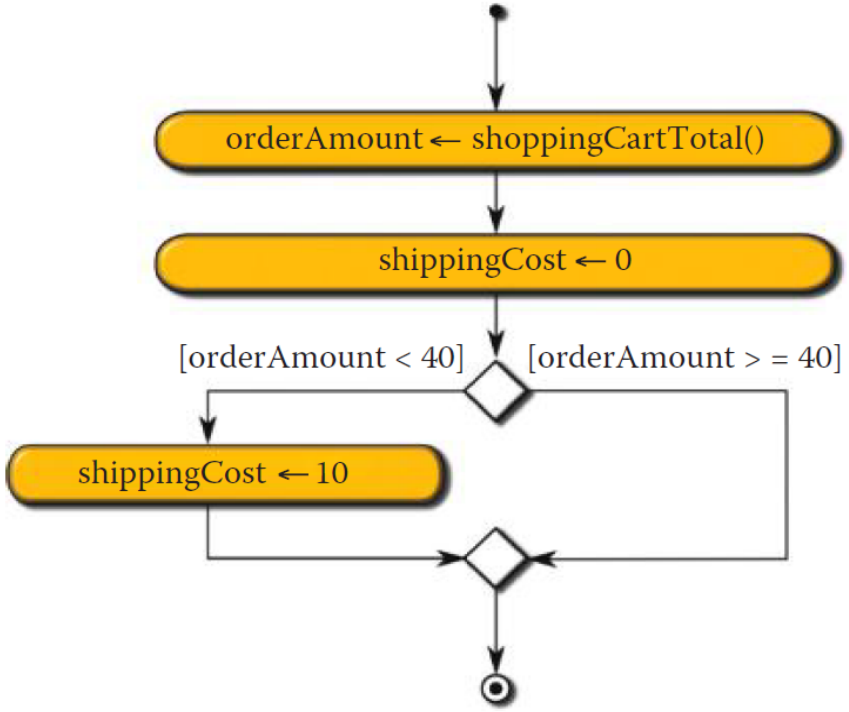
One-way Statement allows a programmer to either perform an action or skip the action.

One-Way Selection

```
if CONDITION then
   ACTIONS
endif
```

Program to Compute Shipping Cost

```
1. orderAmount ← shoppingCartTotal()
2. shippingCost ← 0
3. if orderAmount < 40 then
4.    shippingCost ← 10
5. endif
```



Selection

A two-way selection

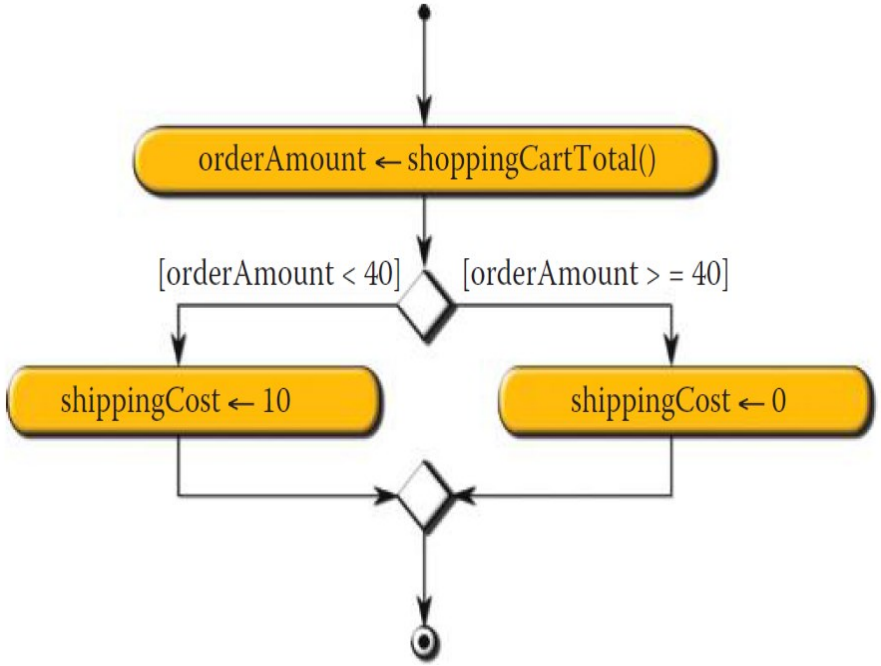
Statement allows the computer to choose one of exactly two actions.

Two-Way Selection

```
if CONDITION then
   IF-TRUE-ACTIONS
else
   IF-FALSE-ACTIONS
endif
```

Program to Compute Shipping Cost

```
1. orderAmount ← shoppingCartTotal()
2. if orderAmount < 40 then
3.    shippingCost ← 10
4. else
5.    shippingCost ← 0
6. endif
```

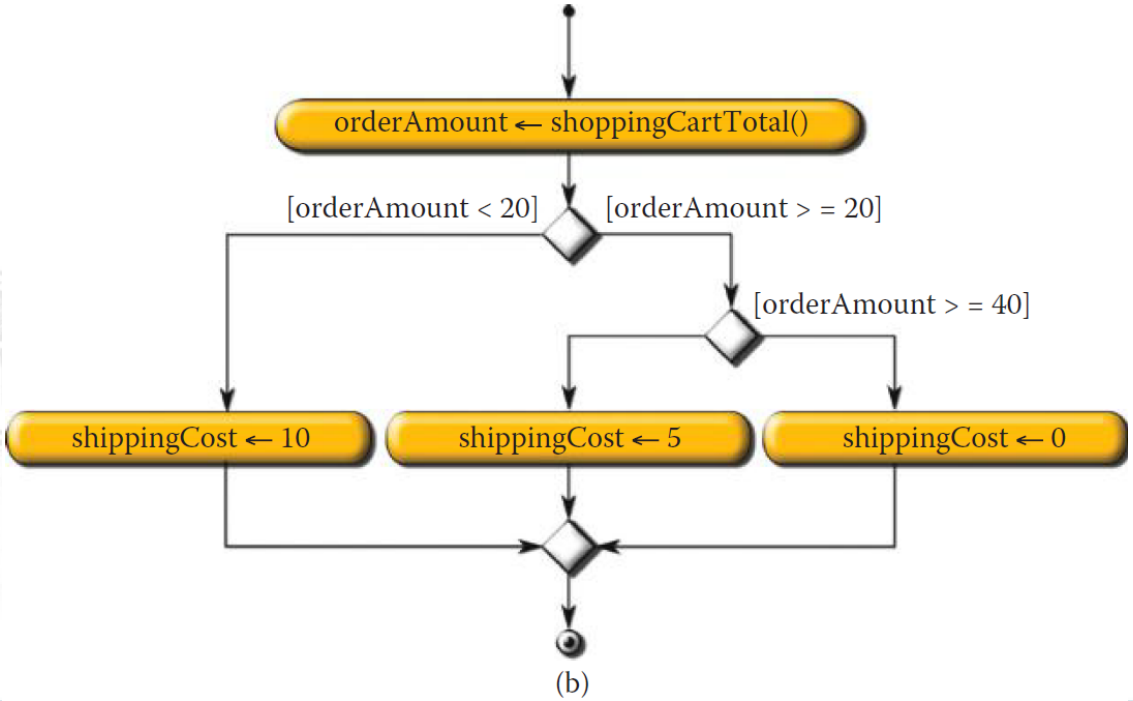


Selection

Multiway selection

Statement allows the computer to choose one of several alternatives.

Shipping Cost Policy	
Order Amount	**Shipping Cost**
$0.00 to $19.99	$10.00
$20.00 to $39.99	$5.00
$40.00 and up	$0.00



(b)

Program to Compute Shipping Cost

```
1. orderAmount ← shoppingCartTotal()
2. if orderAmount ≥ 0 and orderAmount < 20 then
3.    shippingCost ← 10
4. elseif orderAmount ≥ 20 and orderAmount < 40 then
5.    shippingCost ← 5
6. else
7.    shippingCost ← 0
8. endif
```

Program to Compute Shipping Cost

```
1. orderAmount ← shoppingCartTotal()
2. if orderAmount < 20 then
3.    shippingCost ← 10
4. elseif orderAmount < 40 then
5.    shippingCost ← 5
6. else
7.    shippingCost ← 0
8. endif
```

Repetition

A ***loop*** is a control structure that repeatedly executes a sequence of actions

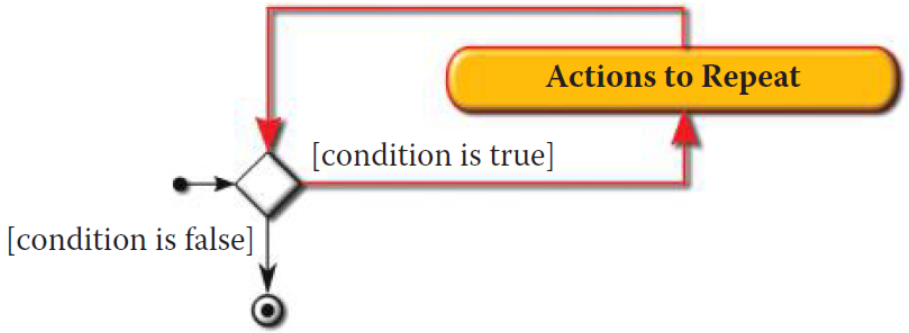
A ***while loop*** is a type of loop where a sequence of actions is repeated as long as some logical condition holds.

A ***counting loop*** is one that contains a variable to keep track of the number of times the loop is actually executed.

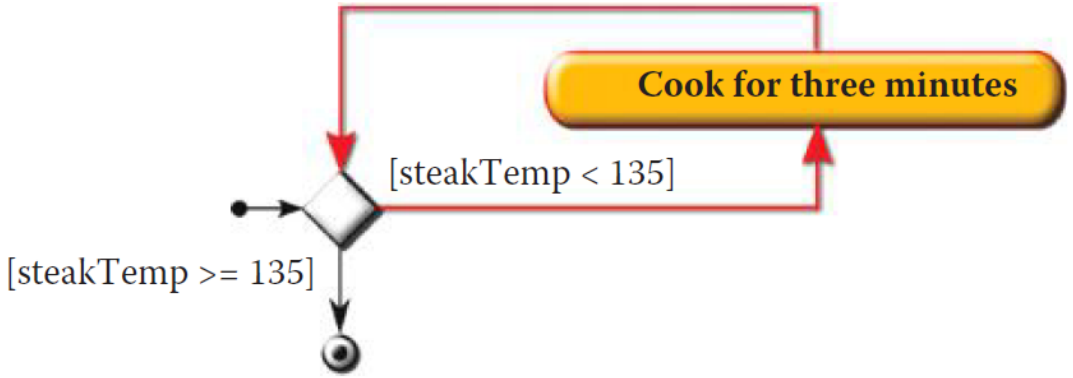
Repetition

A While Loop

```
while CONDITION do
    ACTIONS
endwhile
```



Flowchart structure for a while loop.



Counting Loop (Repetition)

Grill a Steak Program

```
1. steakTemp ← 75
2. while steakTemp < 135 do
3.     steakTemp ← steakTemp + 13
4. endwhile
```

Time to Grill a Steak Program

```
1. steakTemp ← 75
2. minOnGrill ← 0
3. while steakTemp < 135 do
4.     steakTemp ← steakTemp + 13
5.     minOnGrill ← minOnGrill + 3
6. endwhile
```

Line number	Comment	State after execution
1	Bind 75 to steakTemp	*{steakTemp=75}*
2	The condition steakTemp < 135 is true (therefore repeat)	*{steakTemp=75}*
3	steamTemp = steakTemp + 13	*{steakTemp=88}*
2	The condition steakTemp < 135 is true (therefore repeat)	*{steakTemp=88}*
3	steamTemp = steakTemp + 13	*{steakTemp=101}*
2	The condition steakTemp < 135 is true (therefore repeat)	*{steakTemp=101}*
3	steamTemp = steakTemp + 13	*{steakTemp=114}*
2	The condition steakTemp < 135 is true (therefore repeat)	*{steakTemp=114}*
3	steamTemp = steakTemp + 13	*{steakTemp=127}*
2	The condition steakTemp < 135 is true (therefore repeat)	*{steakTemp=127}*
3	steamTemp = steakTemp + 13	*{steakTemp=140}*
2	The condition steakTemp < 135 is **false** (therefore stop)	*{steakTemp=140}*

Design pattern states (Repetition)

1. Initialization
2. Condition
3. Progress

Infinite Loops (Repetition)

An **infinite loop** is a loop that will never terminate because the loop never makes progress towardtermination

Time to Grill a Steak Program (incorrect)

```
1. steakTemp ← 75
2. minOnGrill ← 0
3. while steakTemp ≠ 135 do
4.     steakTemp ← steakTemp + 13
5.     minOnGrill ← minOnGrill + 3
6. endwhile
```

Modularization

Modularization is a vital element of programming that allows us to define new computable actions by assigning a name to some computable process.

Algorithms can be **modularized** by breaking them into independent subprocesses

Modularization

The grillSteak module

Grill a Steak Module

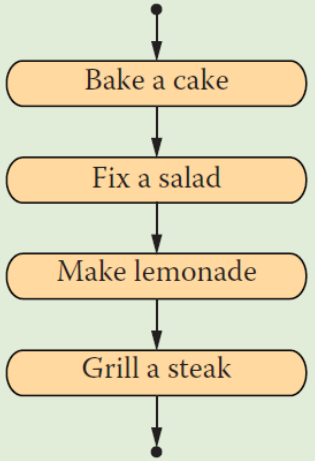
```
1.  module grillSteak() is
2.      steakTemp ← 75
3.      while steakTemp < 135 do
4.        steakTemp ← steakTemp + 13
5.      endwhile
6.  endmodule
```

Modularization

Making a dinner expressed as a sequence of subprocesses.

Make Dinner Module

```
1. module makeDinner() is
2.     bakeCake()
3.     fixSalad()
4.     makeLemonade()
5.     grillSteak()
6. endmodule
```



Modularization

Criteria :

1. *Understandability*

   Every module is self-contained, which implies that it can be fully understood without any knowledge of actions that take place outside of the module itself.

2. *Encapsulation*

   Every module affects only the data that it contains. Any errors that arise from a module are also contained within the module.

3. *Composition*

   Every module can be incorporated into a larger module without special treatment.

Modularization

- Modules are made **flexible** by allowing users to feed **input(*formal parameter*)** values into the code.
  - The initial values provided by the module user are known as ***arguments*** or ***actual parameter***

```
module NAME(V1, V2, ...., VN) is
   ACTIONS
endmodule
```

Modularization

- A more flexible grillSteak module

Flexible Grill a Steak Module

```
1. module grillSteak(steakTemp) is
2.     while steakTemp < 135 do
3.       steakTemp ← steakTemp + 13
4.     endwhile
5. endmodule
```

Modularization

- An optimally flexible grillSteak module

Most Flexible Grill a Steak Module

```
1. module grillSteak(steakTemp, targetTemp, increase-
   Amount) is
2.    while steakTemp < targetTemp do
3.       steakTemp ← steakTemp + increaseAmount
4.    endwhile
5. endmodule
```

Modularization

- Ex.
  - grillSteak(65, 130, 2)
    - This causes the steak to be grilled from a starting point of **65 degrees** until it reaches a temperature of at least **130 degrees** where the temperature increases by **2 degrees** for every three minutes of grill time.

Karakteristik Algoritma

- Tepat

  Pasti, tidak ambigu
- Selesai
- Efektif
- Umum (General)

  Dapat digunakan untuk menyelesaikan setiap instans dari masalah

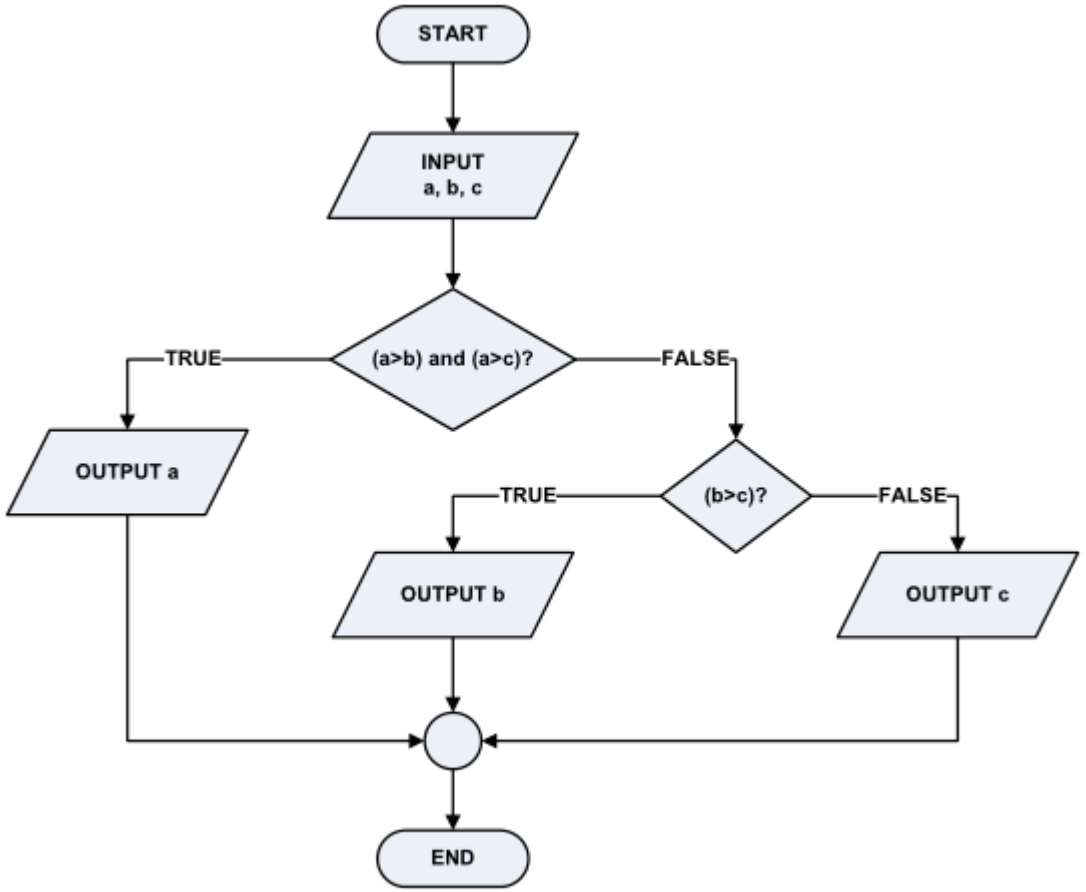
Representasi Algoritma

- Pseudo-codes

Pseudo-codes	Algoritma
• A = A + 5	• Nilai A ditambah dengan 5
• IF (A > 5) THEN WRITE (A)	• Cetak nilai A, jika nilai tersebut lebih besar dari 5
• IF (A > B) THEN WRITE (A) ELSE WRITE (B)	• Dari dua buah nilai A dan B cetak salah satu yang terbesar
• WHILE (A > 0) DO A = A - 2 END DO	• Kurangi dengan 2 nilai A terus menerus sampai nilainya lebih kecil atau sama dengan nol

Representasi Algoritma

- Flowchart



Question and Answer

Terima kasih