

**Modul Panduan
Raptor - Flowchart Interpreter
Yang Tidak Dipublikasikan**



Yusuf Lestanto, ST., MSc., MBA



**PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK DAN ILMU KOMPUTER
UNIVERSITAS BAKRIE
Semester Ganjil 2023/2024**

LEMBAR PENGESAHAN MODUL PANDUAN YANG TIDAK DIPUBLIKASIKAN

Judul : Modul Panduan: Raptor - Flowchart Interpreter
Penulis
a. Nama lengkap : Yusuf Lestanto, ST., MSc., MBA.
b. Jenis kelamin : Laki-laki
c. NIDN : 0302057105
d. Bidang Keahlian : Teknik Informatika
e. Program Studi : Teknik Informatika
f. E-mail : yusuf.lestanto@bakrie.ac.id
Jangka waktu penulisan : 14 September 2023 - 12 Oktober 2023

Jakarta, 15 Desember 2023

Menyetujui



Ardiansyah, S.TP., M.Si., Ph.D.
NIDN: 0318107501

Penulis



Yusuf Lestanto, S.T., MSc., MBA
NIDN: 0302057105

Guidance Module: Raptor - Flowchart Interpreter

Yusuf Lestanto

1 Introduction

Raptor is a free and open-source software tool that is available for Windows, macOS, and Linux. It includes a flowchart editor, an interpreter, and a debugger.

To use a raptor interpreter, students simply create a Raptor flowchart and then click the "Run" button. The interpreter will then execute the flowchart and display the results. Students can also use the debugger to step through their code line-by-line and identify any errors.

Raptor interpreters can be a valuable tool for students who are learning programming. By providing students with a visual way to express their algorithms and immediate feedback on their code, they can help students to learn programming concepts more quickly and effectively.

1.1 Integrated Development Environment (IDE)

An integrated development environment (IDE) for a Raptor interpreter is a software tool that provides a comprehensive environment for developing and debugging Raptor programs. It typically includes a flowchart editor, an interpreter, a debugger, and other tools.

An IDE for a Raptor interpreter can provide a number of benefits to students and programmers, including:

- It can help users to write and edit Raptor code more efficiently.
- It can provide immediate feedback on code errors, helping users to identify and fix them quickly.
- It can help users to understand the logic of their code by providing a visual representation of the code flow.
- It can help users to develop their programming skills by providing a variety of tools and resources.

IDE for Raptor interpreter can be a valuable tool for students and programmers who are developing Raptor programs. By providing a comprehensive environment for developing, debugging, and understanding Raptor code, they can help users to write better code and learn faster.

Here are some ideas for features that could be included in an IDE for a Raptor interpreter:

- A flowchart editor with syntax highlighting and auto-completion.
- An interpreter that can execute Raptor flowcharts and display the results in a variety of ways, such as a console window, a graphical interface, or a text file.

- A debugger that allows users to step through their code line-by-line and identify any errors.
- A code editor with syntax highlighting, auto-completion, and error checking.
- A built-in help system that provides documentation on the Raptor language and the IDE.
- The ability to generate code in a variety of programming languages, such as C, C++, Java, and Python.

1.2 Raptor interpreter

The Figure 1 shows the initial display when the raptor interpreter program is executed. There are a start symbol as the beginning of the program and an end symbol as the end of the program.

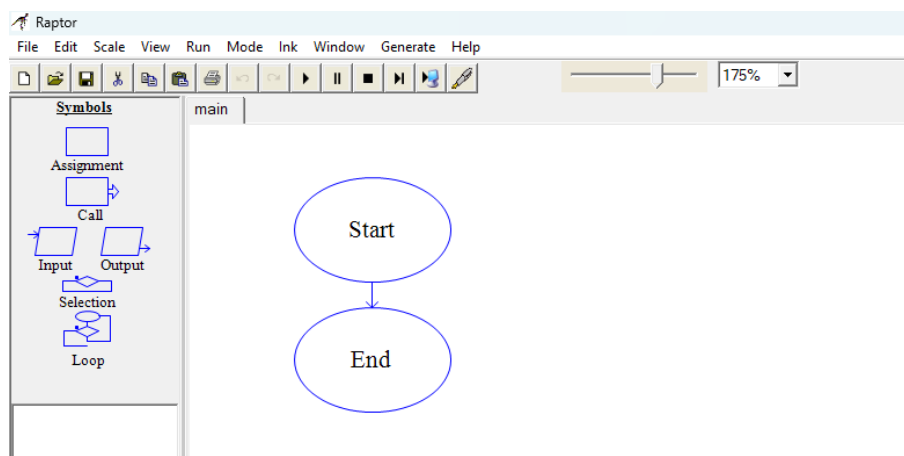


Figure 1: Start window

1.3 Symbols in Raptor

On the left side of the raptor window there is a list of symbols (see Figure 1). There are six symbols in the Raptor interpreter:

1. Assignment symbol

This symbol is used to assign a value to a variable. For example, the assignment "x ← 5" would assign the value 5 to the variable x.

2. Input symbol

This symbol is used to prompt the user for input. For example, the input symbol "Enter a number:" would prompt the user to enter a number.

3. Output symbol

This symbol is used to display output to the console. For example, the output symbol "Display the sum of 2 and 3:" would display the sum of 2 and 3 to the console.

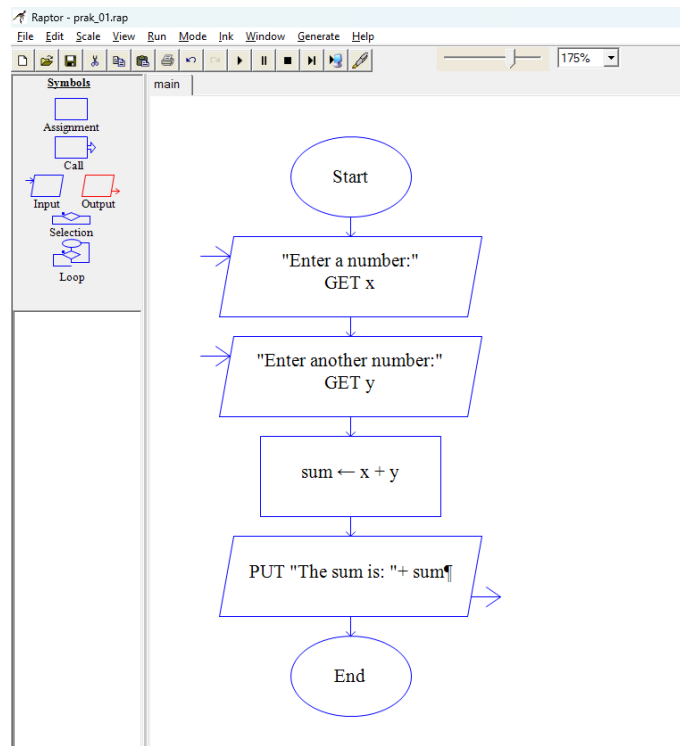


Figure 2: Sum of two numbers

4. Selection symbol

This symbol is used to make decisions. For example, the selection symbol "If x is greater than 5, display 'x is greater than 5.'" would display the message "x is greater than 5." to the console if the value of x is greater than 5.

5. Loop symbol

This symbol is used to repeat a block of code multiple times. For example, the loop symbol "While x is less than 10, display x and increment x by 1." would display the values of x from 0 to 9 to the console.

6. Call symbol

This symbol is used to call a procedure or function. For example, the call symbol "Call myProcedure()" would call the procedure myProcedure().

Those symbols can be used to create a variety of programs, including programs that perform calculations, make decisions, and repeat tasks.

Figure 2 is an example of a simple Raptor program that calculates the sum of two numbers. This program uses the input symbol to prompt the user for two numbers. It then uses the assignment symbol to store the user's input in variables. Next, it uses the calculate symbol to calculate the sum of the two numbers and store it in a variable. Finally, it uses the output symbol to display the sum to the console. By understanding the symbols in the Raptor interpreter, you can create a variety of programs to solve different problems.

2 Introduction to Module

A module is a set of statements used to perform specific tasks within the program. Using modules in various contexts, such as in software development, education, or organizational management, can offer several benefits:

1. Modularity

Modules can be used to decompose complex systems and projects into smaller manageable components. Each module can capture specific functions to facilitate the understanding, maintenance, and debugging of code or systems. This modularity increases the reuse and reading of the code.

2. Reusability

One of the main advantages of the module is that it can be reused in different parts of the project or even in completely different projects. Reusable modules save time and effort and you don't have to recreate the same functionality.

3. Scalability

Modular systems are often more flexible. Modules can be added or replaced without changing the entire system. This flexibility is essential when you need a project to evolve or expand over time.

4. Maintenance

When problems or errors occur, each module has a clear purpose and is easier to identify problems in a modular system. This simplifies debugging and reduces the risk of unwanted side effects when implementing changes.

5. Testing

Modules can be independently tested, facilitating unit tests and reducing the complexity of the whole system test. The isolation of modules for testing ensures that each component works properly before integration.

In summary, the use of modules promotes a structured, maintenance-free and scalable approach to software development and other fields. It improves the organization, reuse, simplifying debugging and testing processes.

2.1 Create a Module using Raptor Interpreter

Before making a module, the raptor mode have to be changed to **intermediate** mode, as in the Figure 3.

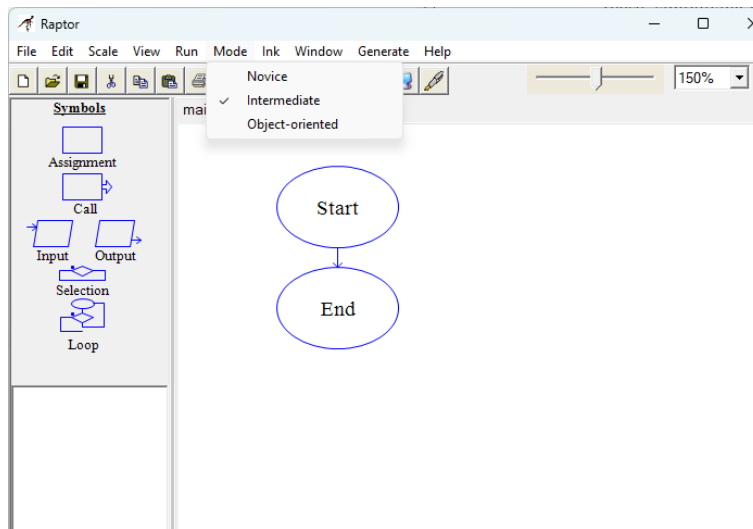


Figure 3: Change Mode to "Intermediate"

There are two types of creating modules, namely subcharts and procedures. To display the subchart and procedure options, hover the cursor and place it over the main tab. Once the cursor is above the main tab, then click the right mouse button, those options will appear as in the Figure 4.

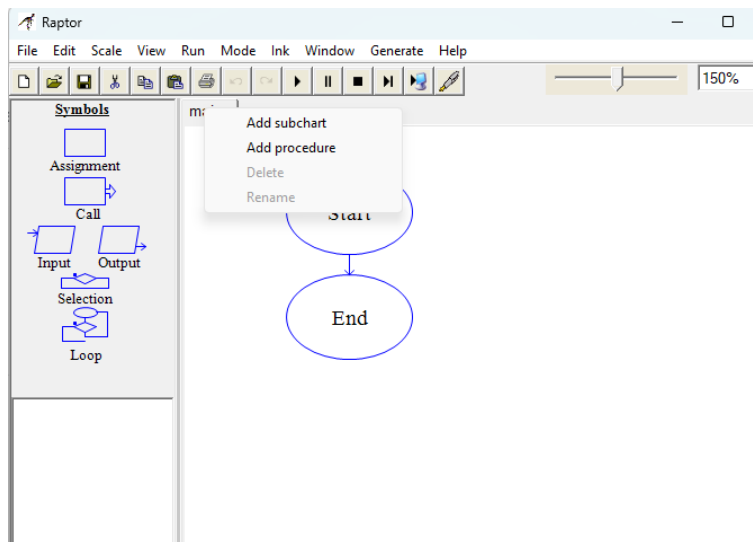


Figure 4: Subchart and procedure menus

2.1.1 Subchart

A subchart in the Raptor interpreter is a series of related programming statements that solve a small sub-problem of a larger problem. Subcharts are used to break down a complex program into smaller, more manageable pieces. This makes the program easier to write, understand, and debug. To create a subchart in Raptor (see Figure 5):

1. Right-click on the "main" subchart tab and select "Add subchart".

2. Give the subchart a meaningful name.
3. Click "Create".

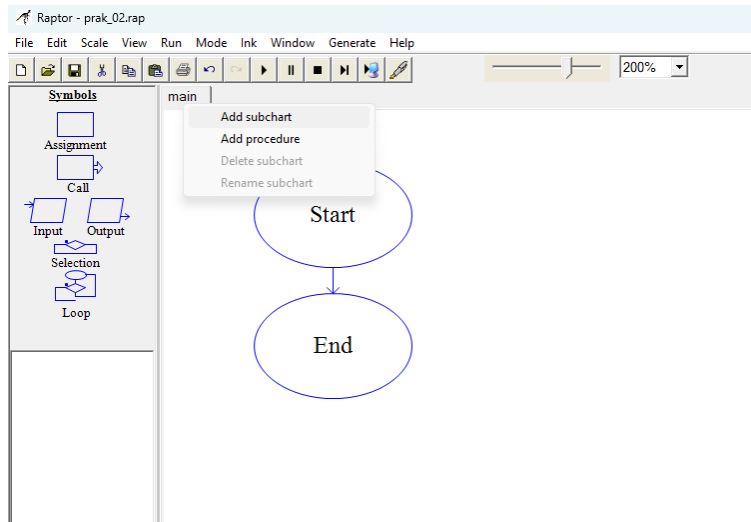


Figure 5: Add Subchart

2.1.2 Procedure

Procedure is similar with subchart. The difference lies in that procedure requires passing arguments while subchart does not (see Figure 6). To add a procedure in the Raptor interpreter, follow these steps:

1. Click the Procedures tab on the left-hand side of the Raptor window.
2. Click the Add Procedure button.
3. Enter a name for the procedure in the Procedure Name field.
4. Click the OK button.

The default argument option is as input, but that could be set as output by checking the output option. Additionally, a single argument can be declared as both input and output.

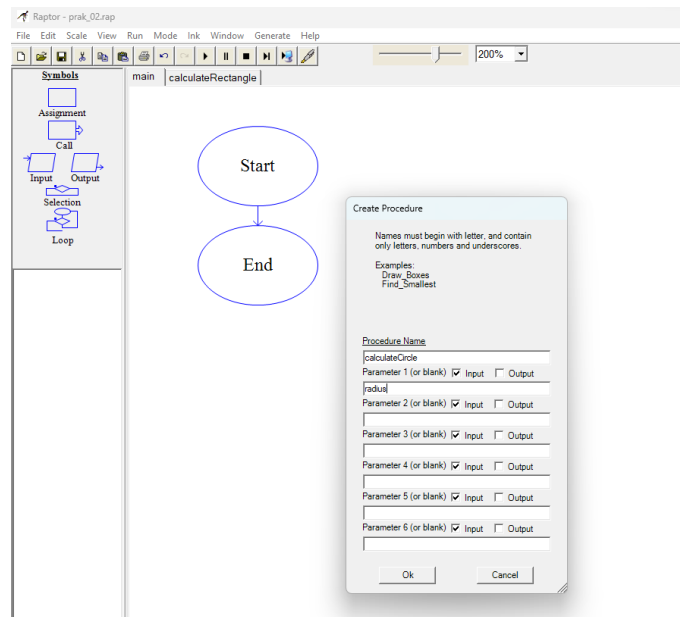


Figure 6: Add Procedure

3 Decision Structure

Decision structures (also known as selection structures) enable programs to take action only under certain conditions. Certain conditions relate to boolean logic. The Boolean expression tested by a **If-Then** statement is formed with a relational operator. A relational operator determines whether there is a specific relationship between two values. For example, the greater than operator ($>$) determines whether one value is larger than another. An operator equal to ($==$) determines whether two values are equal. Table 1 lists relational operators commonly used in most programming languages.

Table 1: Relational Operators [1]

Operator	Meaning
$==$	Equal to
$!=$	Not equal to
$>$	Greater than
$<$	Less than
$>=$	Greater than or equal to
$<=$	Less than or equal to

This expression ($length > width$) determines whether the length value is greater than the width value. If the length exceeds the width, the expression value is **true**. On the other hand, the value of the expression is **false**. Since the expression is true or false only, it is a **Boolean** expression.

3.1 Implementation of If...Then... Else... statements

The Raptor interpreter implements decision structures using two basic components:

- **If/Then** statement

This statement is used to execute a block of code if a condition is true.

- **Else** statement

This statement is used to execute a block of code if a condition is false.

These two statements can be combined to create nested decision structures, which allow for more complex decision-making. To implement a decision structure in Raptor, you simply drag and drop the appropriate components onto the flowchart. In Figure 7, to create a simple If/Then statement, you would drag the If/Then component onto the flowchart and then connect it to the two blocks of code that you want to execute. The condition for the If/Then statement would be placed in the diamond-shaped box (**Condition > 0**) at the top of the component.

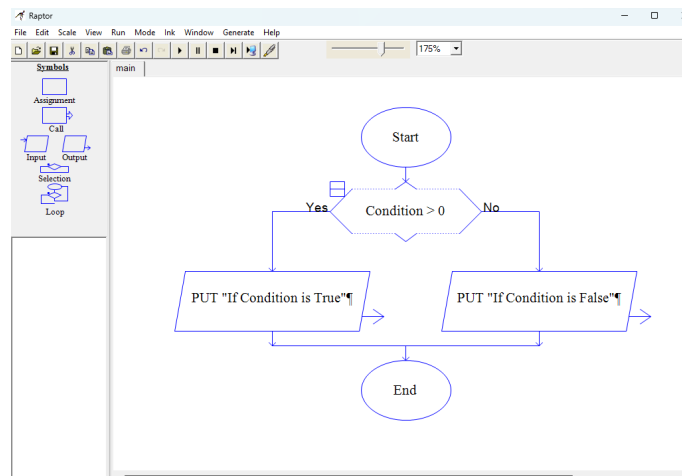


Figure 7: If...Then... Else Statements

3.2 Implementation of If... ElseIf... Else... statements

To create a nested decision structure, you would simply drag and drop additional If/Then and Else statements onto the flowchart and connect them together. In Figure 8, the flowchart shows a nested decision structure that checks to see if a number is greater than five (5) or not. This condition will be executed if the input number greater than zero.

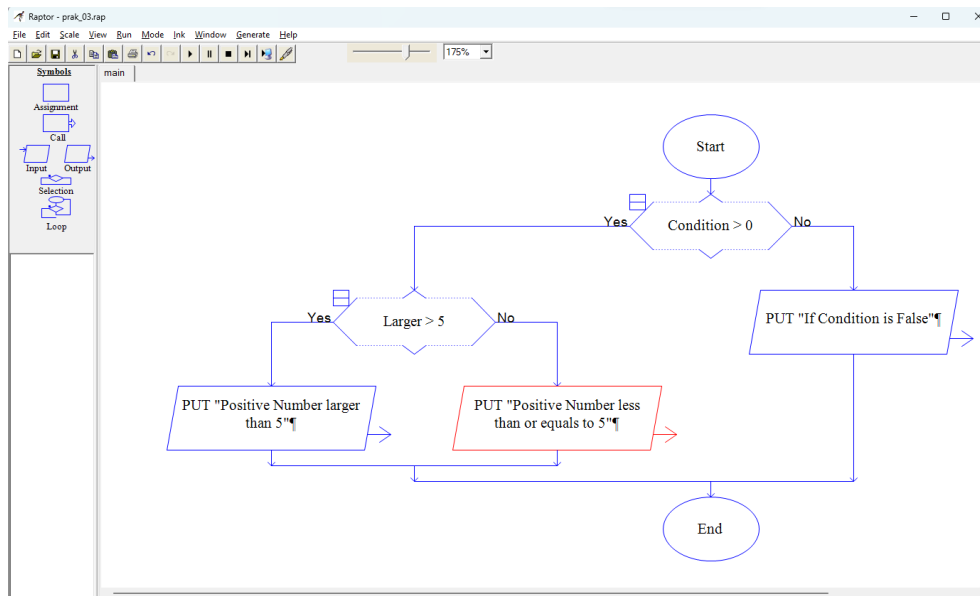


Figure 8: Nested condition structure

4 Repetition

A repetition structure causes a statement or set of statements to execute repeatedly. Repetition is a basic concept of programming. This allows programmers to write code that can be executed several times. This is essential to solve many types of problems. For example, if a programmer wants to print numbers from 1 to 100, it can use a loop for this. The for loop executes the code within it 100 times, for each number from 1 to 100.

4.1 Pretest and Posttest loop

A **pretest** loop is a type of repetition structure that evaluates a condition before executing the code inside the loop. If the condition is true, the code inside the loop is executed. The condition is then evaluated again before the next iteration of the loop. This process continues until the condition is false.

The most common example of a pretest loop is the **while/for** loop. In a while loop, the condition is evaluated at the beginning of the loop. If the condition is true, the code inside the loop is executed. The condition is then evaluated again before the next iteration of the loop. This process continues until the condition is false.

A **posttest** loop is a type of loop in which the condition is evaluated after the loop body is executed. This means that the loop body will always be executed at least once. The most common example of a posttest loop is the **do-while** loop.

Two types of repetition loops: **for** loop and **while** loop. For loops are used when the programmer knows how many times the code inside the loop needs to be executed. While loops are used when the

programmer does not know how many times the code inside the loop needs to be executed.

4.1.1 While/For Loop

A for loop is a type of control flow statement that allows programmers to execute a block of code multiple times. The syntax for a **for loop** in Raptor is as follows:

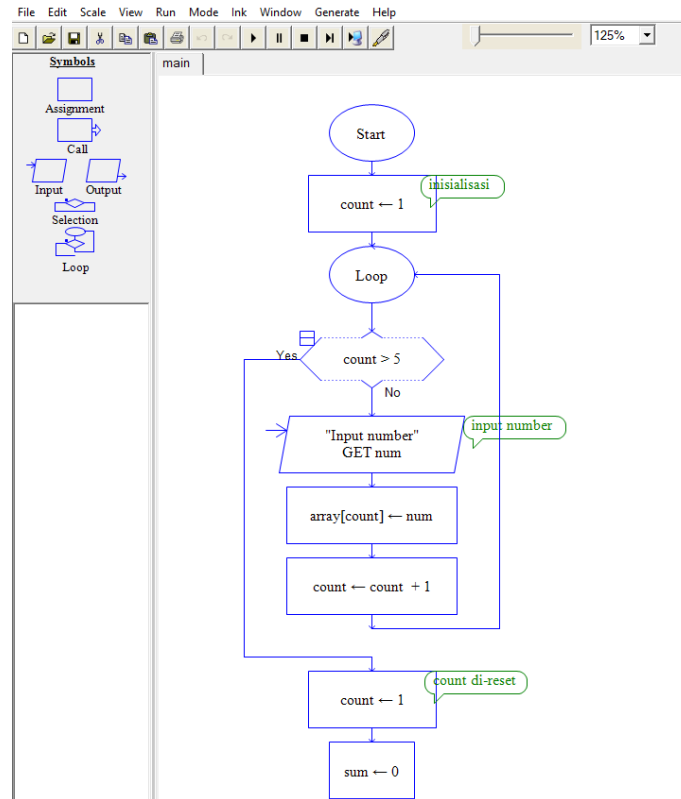


Figure 9: While/For...loop

The initialization statement is executed once before the loop begins. The condition statement is evaluated before each iteration of the loop. If the condition is true, the statements inside the loop are executed. The update statement is executed after each iteration of the loop.

The initialization statement sets the variable i to 1. The condition statement checks if i is less than or equal to 5. If the condition is true, the print statement is executed, printing the value of i . The update statement increments the value of i by 1.

4.1.2 Do...While Loop

A while loop is a type of repetition structure that allows code to be executed repeatedly based on a given Boolean condition. The while loop works by first evaluating the condition. If the condition is true, the code inside the loop is executed. The condition is then evaluated again. This process continues until the condition is false.

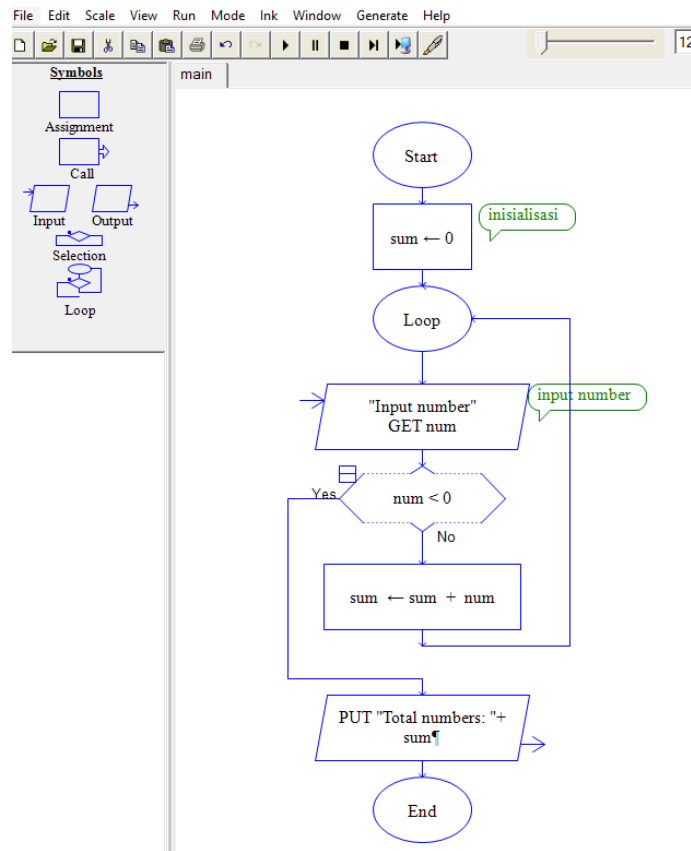


Figure 10: Do... While loop

In Figure 10, the condition is *num* less than 0. Before the first time the loop is executed, the value of *sum* is 0. Since *num* is greater than 0, the code inside the loop is executed. The value of *sum* is then added by *num*. This process continues until the value of *num* is less than 0. At that point, the loop terminates. At last, the total *sum* will be informed.

5 Function

A function in the Raptor interpreter is a reusable block of code that can be called from other parts of a program. Functions can take input parameters and return output values. Functions are a good way to organize code and make it more modular and reusable.

To create function is similar to create modules which are already explain in Section 2. Now, it's will be explained how to create random number using raptor. In this case, random number is limited to ten numbers.

5.1 Creating main program

The main part of program is shown in figure 11. Function "*generateRandom()*" will be called in main program. After calling function, program execution will jump to function part.

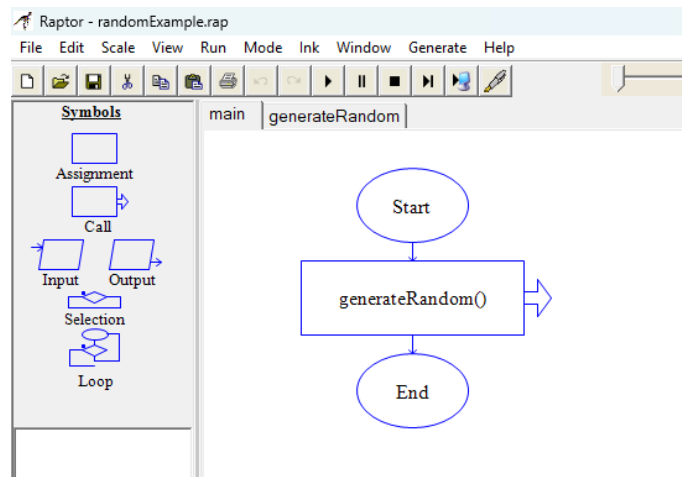


Figure 11: Main program of creating random numbers

5.2 Creating function

In Raptor Interpreter, creating function will follow the provided function template. In Figure 12, user is able to input the function name and passing arguments. For this example, there are no passing arguments in function creation. Name "generateFunction()" is given.

Figure 12: Creating function template

5.3 Generate Random Number function

To generate random numbers in the Raptor interpreter, the provided **Random()** function can be used. This function takes no arguments and returns a random number between 0.0 and 1.0. This function

”**floor()**” will generate a random integer between 1 and 100.

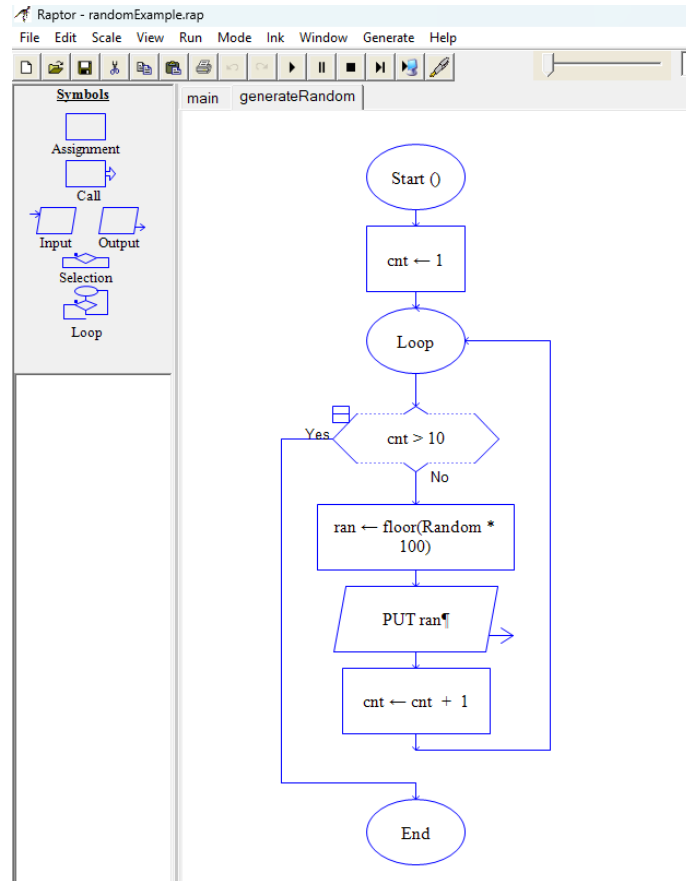


Figure 13: Generate ten random numbers

References

- [1] Gaddis, T. Starting Out with Programming Logic and Design, 4/e. (Pearson Education,2015)

SURAT TUGAS

No: 033a/ST/UBakrie/D-FTIK/II.2024

Dekan Fakultas Teknik dan Ilmu Komputer (FTIK) Universitas Bakrie dengan ini menugaskan kepada:

Nama : Yusuf Lestanto, S.T., M.Sc.. MBA.
NIDN : 0302057105
Prodi : Informatika
JJA : Asisten Ahli (AA)

Untuk melaksanakan tugas Menulis Modul Panduan **Raptor - Flowchart Interpreter** yang dilaksanakan pada semester ganjil 2023/2024 tanggal 12 Oktober 2023

Kegiatan tersebut tidak mengganggu kegiatan belajar mengajar yang telah menjadi tugas pokok yang ditetapkan oleh Universitas Bakrie.

Kepada yang bersangkutan diwajibkan untuk melaporkan hasil pengabdianya kepada Lembaga Pengabdian Masyarakat.

Demikian surat tugas ini diberikan untuk dilaksanakan sebaik-baiknya.

Jakarta, 02 Februari 2024
Fakultas Teknik dan Ilmu Komputer



UNIVERSITAS BAKRIE
Dr. Mohammad Ihsan, S.T., M.T., M.Sc.
Dekan

Tembusan:

1. Dekan FTIK
2. Kabiro SDM
3. Arsip