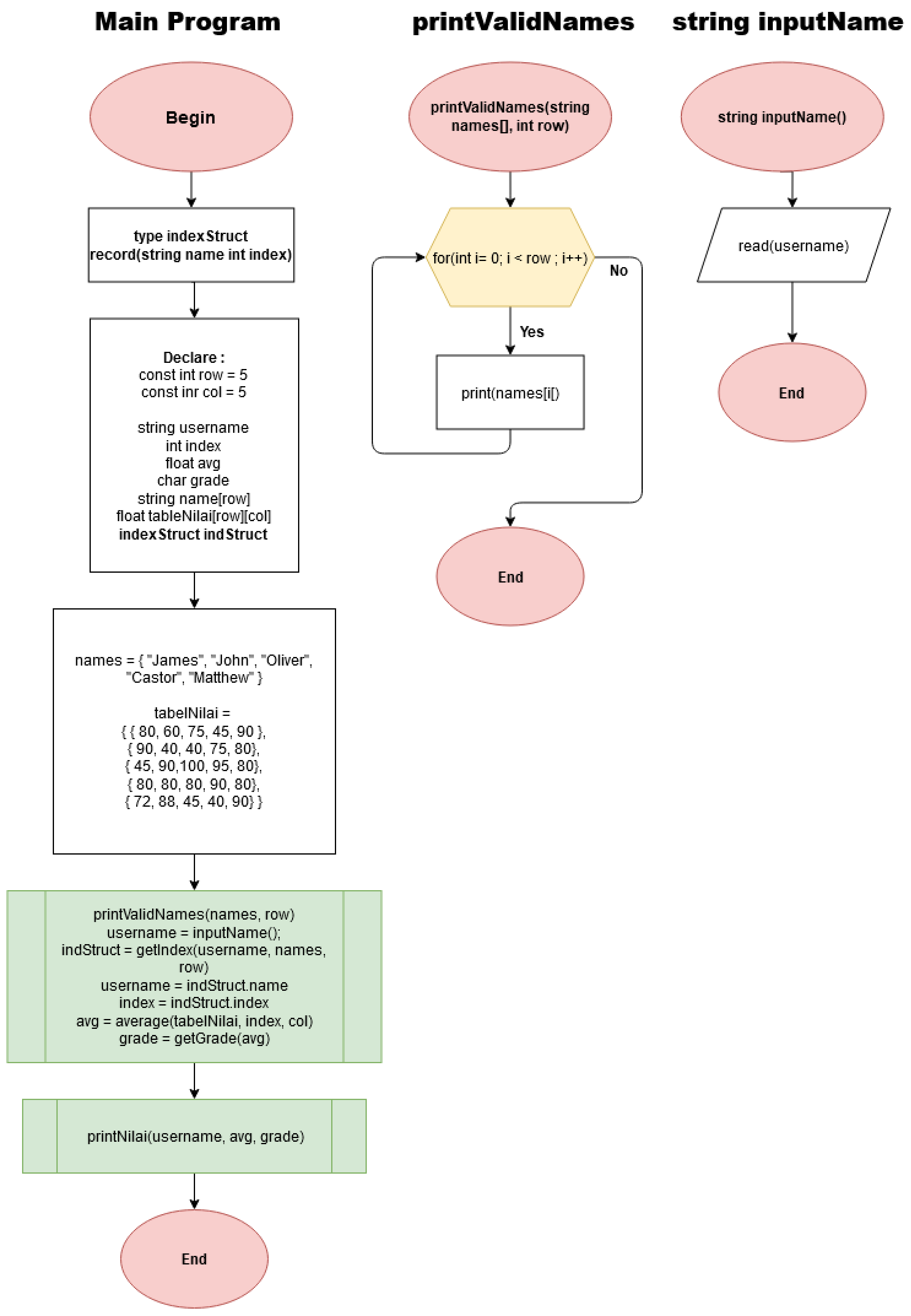
```
//*****************************************************************
****
//  © KELOMPOK 11 2020/10/17
//  ANGGOTA :
//  ~ ANDREW VIRYA VICTORIO - 32200091
//  ~ VINCENT GEORGE CHANDRA - 32200083
//  ~ MATIAS INDRA PANGESTU - 32200095
//  ~ BENEDICTUS DIKHA ARIANDA - 32200092
//  ~ CALVIN OWEN SUSANTO - 32200084
//  TEKNIK INFORMATIKA A.T 2020: 1PTI1: ALGORITMA (TIB01)
//  RABU -- 8 SKS
//  TUGAS KELOMPOK: MEMBUAT PROGRAM DENGAN 4 FUNGSI, 1 MAIN PROGRA
M
//                  DENGAN 1 DIMENSI ARRAY DAN 2 DIMENSI ARRAY
//  PEMBAHASAN: 9
//  DOSEN: CHYQUITHA DANUPUTRI, S.KOM, M.KOM
//  TEMPO: 2020/10/21 ~~ 2020/10/28
//*****************************************************************
****

/*
PROCEDURE:

printValidNames(string names[], int row)

    DECLARATION: none

    ALGORITHM:

    for (i : integer <- 0 to (row-1)) do
        print(names[i])
    endfor



FUNCTION:

inputName() : string

    DECLARATION:
    username : string
    ALGORITHM:
    read(username)
    return username

END FUNCTION



FUNCTION:

getIndex(username : string, names[5] : string, row : integer) : in
dexStruct

    DECLARATION:

    number = 0 : integer
    indStruct : indexStruct

    ALGORITHM:

    for (i <- 0 to row-1) do

        if (username == names[i]) then
            number <- i
            indStruct.name <- username
            indStruct.index <- number
            break

        else if (i == 4) then
            username <- inputName()
            indStruct <- getIndex(username, names, row)
        endif
    endfor

    return indStruct

END FUNCTION
```

```
FUNCTION:

getGrade(float avg) : char


    DECLARATION:

    grade : char


    ALGORITHM:

    if (100 >= avg && avg >= 90) then
        grade is A

    else if (90 > avg && avg >= 80) then
        grade is B

    else if (80 > avg && avg >= 70) then
        grade is C

    else if (70 > avg && avg >= 60) then
        grade is D

    else if (60 > avg && avg >= 50) then
        grade is E

    else if (50 > avg) then
        grade is F

    else
        write (Error!!)

    return grade

END FUNCTION
```

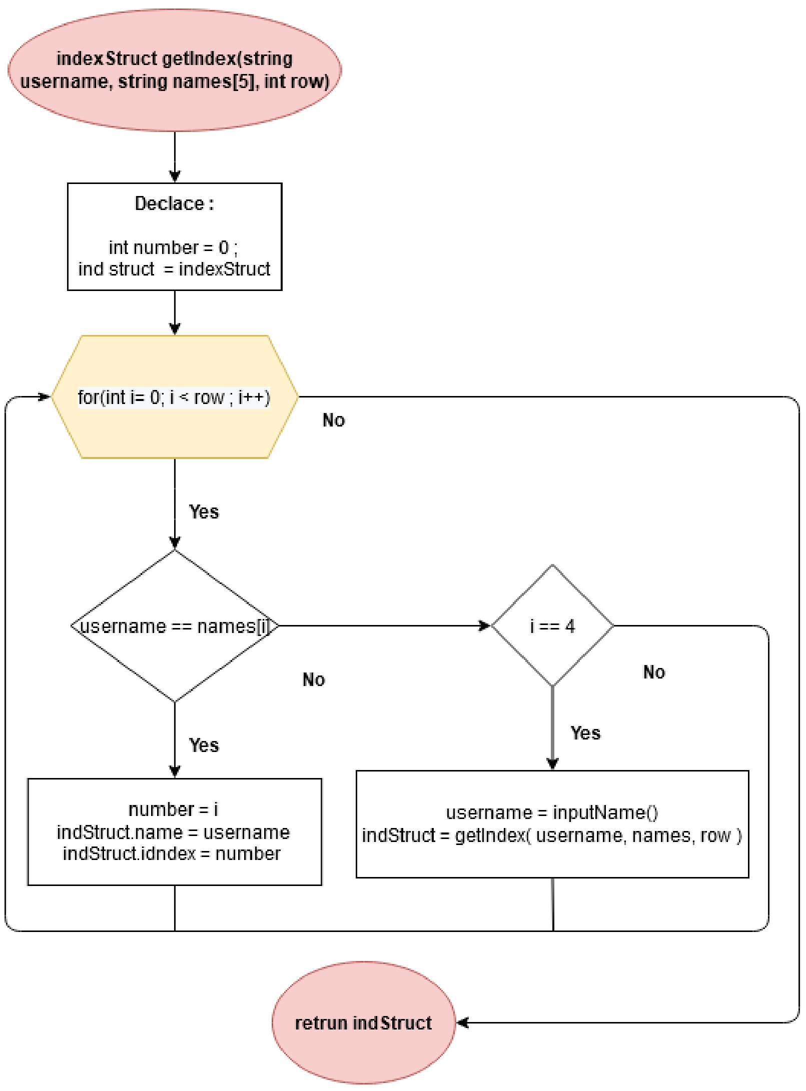
```
FUNCTION:

average(tabelNilai[][5] : float, index : integer, col : integer) :
 float

    DECLARATION:

    avg: float
    jumlah = 0 : float

    ALGORITHM:

    for (int i <- 0 to col-1) do:
        jumlah = jumlah + tabelNilai[index][i]
    end for

    avg = jumlah/col

    return avg

ENDFUNCTION



PROCEDURE:

printNilai(string username, float avg, char grade)

    DECLARATION: none

    ALGORITHM:

    write(username);
    write(avg);
    write(grade);
```

```
MAIN PROGRAM:

PROGRAM DatabaseNilai
{Program menampilkan nilai berdasarkan input nama user}


DECLARATION:

const row = 5;
const col = 5;

struct indexStruct record: {
                name : String
                index : Integer
                }
indStruct : indexStruct

username : String
index : Integer
avg : Float
grade : Char


names : [row] : string
tabelNilai : [row][col] : integer

names = {"James", "John", "Oliver", "Castor", "Matthew" }
tabelNilai = {{ 80, 60, 75, 45, 90 },
        { 90, 40, 40, 75, 80},
        { 45, 90, 100, 95, 80},
        { 80, 80, 80, 90, 80},
        { 72, 88, 45, 40, 90}}


ALGORITHM:

printValidNames(names, row)
username <- inputName()
indStruct <- getIndex(username, names, row)
username <- indStruct.username
index <- indStruct.index

avg <- average(tabelNilai, index, col)
```

```
grade <- getGrade(avg)
printNilai(username, avg, grade)

*/
```

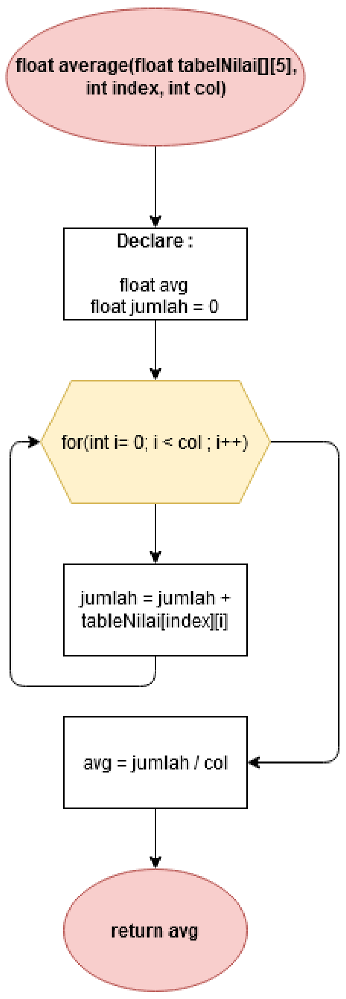

Main Program
printValidNames
string inputName
Begin
type indexStruct record(string name int index)
Declare :
const int row = 5
const inr col = 5
string username
int index
float avg
char grade
string name[row]
float tableNilai[row][col]
indexStruct indStruct
names = { "James", "John", "Oliver", "Castor", "Matthew" }
tabelNilai =
{ { 80, 60, 75, 45, 90 },
{ 90, 40, 40, 75, 80},
{ 45, 90,100, 95, 80},
{ 80, 80, 80, 90, 80},
{ 72, 88, 45, 40, 90} }
printValidNames(names, row)
username = inputName();
indStruct = getIndex(username, names, row)
username = indStruct.name
index = indStruct.index
avg = average(tabelNilai, index, col)
grade = getGrade(avg)
printNilai(username, avg, grade)
End
printValidNames(string names[], int row)
for(int i= 0; i < row ; i++)
No
Yes
print(names[i])
End
string inputName()
read(username)
End

getIndex



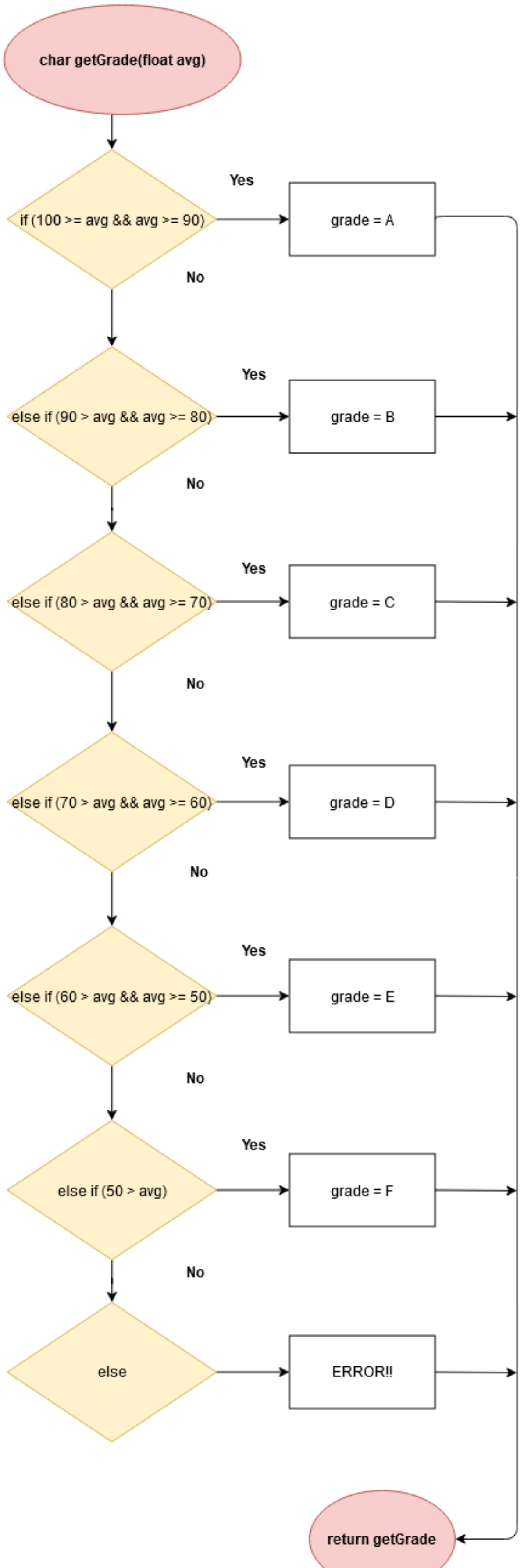
indexStruct getIndex(string username, string names[5], int row)
Declace :
int number = 0 ;
ind struct = indexStruct
for(int i= 0; i < row ; i++)
No
Yes
username == names[i]
No
i == 4
No
Yes
Yes
number = i
indStruct.name = username
indStruct.idndex = number
username = inputName()
indStruct = getIndex(username, names, row)
retrun indStruct

Average

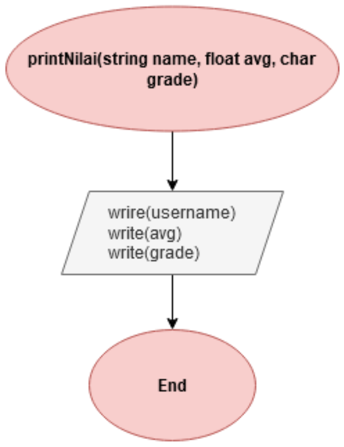


float average(float tabelNilai[][5], int index, int col)
Declare :
float avg
float jumlah = 0
for(int i= 0; i < col ; i++)
jumlah = jumlah + tableNilai[index][i]
avg = jumlah / col
return avg

getGrade


char getGrade(float avg)
if (100 >= avg && avg >= 90)
Yes
grade = A
No
else if (90 > avg && avg >= 80)
Yes
grade = B
No
else if (80 > avg && avg >= 70)
Yes
grade = C
No
else if (70 > avg && avg >= 60)
Yes
grade = D
No
else if (60 > avg && avg >= 50)
Yes
grade = E
No
else if (50 > avg)
Yes
grade = F
No
else
ERROR!!
return getGrade

printNilai


printNilai(string name, float avg, char grade)
wrire(username)
write(avg)
write(grade)
End