COMPUTER ENGINEERING

Laboratory
Multimedia and Internet of Things
Computer Engineering Department
Institut Teknologi Sepuluh Nopember

Basic Programming Practicum

String Array Branching Loop

2024

1 Goals

- Students are familiar with and able to use logical and comparison expressions in the C programming language.
- Students are familiar with and able to use logical and comparison expressions in the C programming language.
- Students can recognize and use while loops in the C language.
- Students can recognize and use do-while loops in the C language.
- Students can recognize and use for loops in the C language.
- Students can recognize and use for loops in the C language.
- Students can recognize and use for loops in the C language.
- Students can recognize and use strings.

2 Logical and Comparasion Expressions

2.1 Comparasion Expressions

The following are the operators used in comparison expressions.

Table 1: Comparasion Operator

Operator	Name	Expression Example
!=	Not Equal To	x != y
>	Greater Than	x >y
==	Equal To	x == y
<	Less Than	x <y
>=	Greater Than Equal To	x >= y
<=	Less Than Equal To	x <= y

A Comparison Expression will return boolean value `true` or `false` which is also represented with the value of 0 or 1. As example:

```
printf("%d",0>1); // Print 0 to the screen
printf("%d",0<1); // Print 1 to the screen
```

2.2 Logical Expression

The following are the logical operators used on a Logical Expression

Table 2: Logical Expression

Operator	Name	Expression Example
&&	AND	$x < 5 \;\&\&\; x < 10$
||	OR	$x < 5 \;\|\|\; x < 4$
!	NOT	$!(x < 5 \&\& x < 10)$

Like comparison expression, logical expression will return boolean values.

3 Branch

3.1 If Statement

`if` statement is used to decide which block of code to be executed if the condition is true.

```
// Block code before if
if (Condition)
{
 // Block of code that will be executed if the condition is true
}
// Block code after if
```

As example, look at the following code

Listing 1: If Statement Example

```
1  include <stdio.h>
2
3  int main()
4  {
5    //Variable Declaration
6    int myMoney,breadPrice;
7    myMoney = 5000;
8    breadPrice = 10000;
9
10   if (myMoney>=breadPrice)
11   {
12       printf("I can buy that bread\n");
13   }
14   printf("hehe");
15   return 0;
16 }
```

Output of the program

```
hehe
```

If line 7 changed to `myMoney=10000`, the outputs of the program would be

```
I can buy that bread
hehe
```

3.2 If-else Statement

`Else` statement is used to decide the block of code to be executed if the condition is false.

```
// Block code before if
if (Condition)
{
 // Block of code that will be executed if the condition is true
} else
{
```

```
 // Block of code that will be executed if the condition is false
}
// Bloc code after if-else
```

The following is an example of using if-else statement:

Listing 2: If-else example

```
1  include <stdio.h>
2
3  int main()
4  {
5    //Varible declaration
6    int myMoney,breadPrice;
7    myMoney = 5000;
8    breadPrice = 10000;
9
10   if (myMoney>=breadPrice)
11   {
12       printf("I can buy that bread\n");
13   }
14   else
15   {
16         printf("I can't buy that bread\n");
17   }
18   printf("hehe");
19   return 0;
20 }
```

Below is the output of that program

```
I can buy that bread
hehe
```

If line 7 changed to `myMoney=10000`, the outputs of the program would be

```
I can't buy that bread
hehe
```

3.3 Pernyataan if-else if

The `else if` statement is used to run a block of code when the condition in `if` or the previous `else if` is false.

```
 // Code block before the if statement
if (Condition1)
{
    /* Code block to be executed if Condition 1
    is true */
}
else if (Condition2)
{
    /* Code block to be executed if Condition 1 is false
```

```
    and Condition 2 is true */
}
else if (Condition3)
{
    /* Code block to be executed when
    Condition 1 and Condition 2 are false, and
    Condition 3 is true */
}
...
else if (ConditionN)
{
    /* Code block to be executed when
    Condition 1 to Condition N-1 are false, and
    Condition N is true */
}
else
{
    /* Code block to be executed when
    Condition 1 to Condition N are false */
}
// Code block after the if statement
```

Below are an example of if-else statement

Listing 3: If-else example if

```
1  include <stdio.h>
2
3  int main()
4  {
5    //Varible declaration
6    int myMoney,breadPrice;
7    myMoney = 5000;
8    breadPrice = 10000;
9
10   if (myMoney>breadPrice)
11   {
12       printf("I can buy that bread\n");
13   }
14   else if(myMoney==breadPrice)
15   {
16       printf("I can buy bread, but my money will run out immediately");
17   }
18   else
19   {
20         printf("I can't buy that bread\n");
21   }
22   printf("hehe");
23   return 0;
24 }
```

Output of this program are below

```
I can't buy that bread
hehe
```

If line 7 changed to `myMoney=10000`, the output of the program would be

```
I can buy bread, but my money will run out immediately
hehe
```

If line 7 changed to `myMoney=12000`, the output of the program would be

```
I can buy that bread
hehe
```

3.4 Nested if

Nested if is when there is a conditional statements within a block of code inside the conditional statement

```
// Code block before the if statement
if (Condition1)
{
 if (Condition2)
 {
  // Do something
 }
 else
 {
  // Do other thing
 }
}
else
{
 // Do other thing
}
```

Below is an example of using nested if

Listing 4: Nested if example

```
1  include <stdio.h>
2
3  int main()
4  {
5    // Declare the variables
6    int myMoney,breadPrice,friendsMoney;
7    myMoney = 5000;
8    breadPrice = 10000;
9    friendsMoney = 42069;
10
```

```
11
12      if (myMoney>breadPrice)
13      {
14          printf("I can buy bread\n");
15      }
16      else if(myMoney==breadPrice)
17      {
18          printf("I can buy bread but I will ran out of money\n");
19      }
20      else
21      {
22          if(friendsMoney+myMoney >= breadPrice)
23          {
24              printf("I can buy bread if I borrow my friend money\n");
25          }
26          else
27          {
28                printf("I can't buy bread\n");
29          }
30      }
31      printf("hehe");
32      return 0;
33  }
```

3.5 Pre-lab Assignment

1. what is the purpose of branching in programming?
2. Apart from using if statements, branching can also be done using switch-case statements. Explain what you know about switch-case statement!
3. Try to make a program that receives 3 integer input A, B, and C. Then outputs those 3 integers to the screen sorted from largest to smallest. Do this only using conditional statements.

4 Loop

4.1 Loop while

While loop will run the code block within it repeatedly as long as the loop condition is true

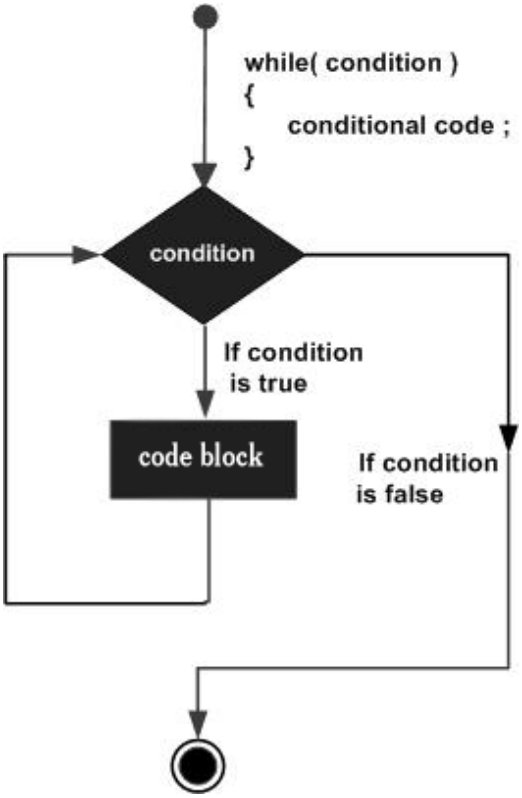

Figure 1: Flow chart loop while

Its syntax in C programming language is as follows

```
while(Condition)
{
    // Block of code that will be repeated
}
```

As an example, look at the following code

Listing 5: While implementation example

```
1  int main()
2  {
3    int myMoney,breadPrice;
4    myMoney = 10000;
5    breadPrice = 2000;
6    while(myMoney >= breadPrice)
7    {
8        printf("Buy 1 bread, my money left %d", myMoney - breadPrice);
9        myMoney -= breadPrice;
10   }
11   printf("I don't have enough money");
12   return 0;
13 }
```

Output in program 5 are below

```
Buy 1 bread, my money left 8000
Buy 1 bread, my money left 6000
Buy 1 bread, my money left 4000
```

```
Buy 1 bread, my money left 2000
Buy 1 bread, my money left 0
I don't have enough money
```

You can see the line 9 of the code causes the variable `myMoney` to have its value substracted by 2000 for every loop until `myMoney` is no longer greater than equal to `breadPrice`. The loop condition will be invalid and finaly exits the loop. Then it prints "Uang saya tidak cukup lagi", the command after the while loop statement.

4.2 Do-while loop

do-while loop is very similar to while loop. The only difference is that do-while loop will execute the code block inside it once, and then checks the condition.

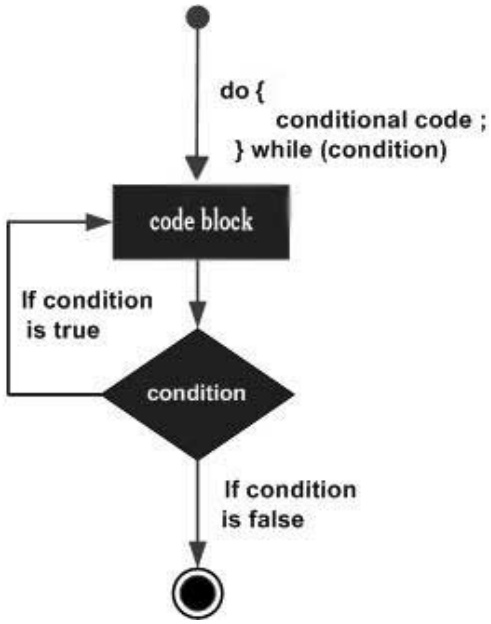

Figure 2: Do-while statement

Its syntax in C is as follows:

```
do{
    // the block of code that will be repeated
}while(Condition)
```

Look at the following example.

Listing 6: Do-while implementation example

```
1 int main()
2 {
3   int myMoney,breadPrice;
4   myMoney = 10000;
5   breadPrice = 12000;
6   do{
7       printf("Buy 1 bread, my money left %d", myMoney - breadPrice);
8       myMoney -= breadPrice;
9   }while(myMoney >= breadPrice)
```

```
10    printf("Uang saya tidak cukup lagi");
11    return 0;
12  }
```

The output of the code above in Listing 6 are

```
Buy 1 bread, my money -2000
I don't have enough money
```

The variable `myMoney` is substracted by `breadPrice` before checking the `myMoney>=breadPrice` condition. Had the code above uses while loop, the repeating block of code wouldn't have executed even once.

4.3 For loop

If you have a block of code like this:

```
InitializationStatement; // e.g.: int i = 0;
while(Condition){
    // do something
    updateStatement; // e.g.: i++
}
```

This is equal to

```
for(InitializationStatement;Condition;updateStatement){
    // do something
}
```

As example, look at the following code:

Listing 7: For implementation example

```
1  int main()
2  {
3      int i=0;
4      for(i=1;i<10;i++){
5          printf("%d ",i);
6      }
7    return 0;
8  }
```

The output of this program are

```
1 2 3 4 5 6 7 8 9
```

The following is the code if code in Listing 7 converted to its while-loop form

Listing 8: For in form of while

```
1  int main()
2  {
3      int i=0;
4      i=1;
5      while(i<10){
```

```
6           printf("%d ",i);
7           i++;
8       }
9   return 0;
10 }
```

Notes: In programming, the "break" keyword is used to exit a loop prematurely, while the "continue" keyword is used to skip the current iteration of a loop and proceed to the next iteration. These keywords are commonly used in loops to control their behavior and make the code more efficient. You can explore their usage further in programming resources and tutorials.!

4.4 Pre-lab Assignment

1. What happens if we write `break;` in a loop?
2. Try to make a program in C that calculates the factorial of a non-negative integer entered by the user using a do-while loop. Show the results.
3. Try to make a program in C language to find prime numbers between 1 and 100. Use the for loop to iterate through all numbers and the continue statement to ignore numbers that are not prime. Display all found primes.

5 Array

Array is a collection of data where each element of it has the same name(indexed) and data type. Every element in an array can be accessed using its element index.

5.1 Array 1D

One dimensional array variable can be declared by deciding the data type of the element and the number of element that is needed.

Syntax:

```
DataType variableName [arraySize];
```

1. `DataType`.
   The data type of the elements in the array, e.g. `float`, `int`, etc.
2. `variableName`
   variableName follows the variable naming convention
3. `arraySize`
   Integer more than 0. Defining the number of element an array has.

Initializing one dimensional array can be done like shown below:

```
int contoh_array[5] = {4,2,0,6,9};
```

Data in an array can be accessed by using an integer that is the index of the array. Look at the code below

Listing 9: Accessing 1D array implementation

```
int main()
{
    int arr[5] = {4,2,0,6,9};
    printf("%d\n",arr[0]);
    printf("%d\n",arr[4]);
    int i = 0;
    printf("%d\n",arr[i]);
    for(i=0;i<5;i++)
        printf("%d",arr[i]);
}
```

The code in Listing 10 will give output

```
4
9
4
42069
```

5.2 Array 2D and Other Multidimensional Array

2D array is basically a 1D array of 1D array. Intuitively, you can define a 2D array like as seen below:

```
DataType variableName[arraySize1][arraySize2];
```

This also applies to multidimensional array.

```
DataType variableName[arraySize1]...[arraySizeN];
```

There will be $arraySize_1 \times arraySize_2 \times \cdots \times arraySize_n$ of elements that would be allocated to the memory after doing multidimensional array like that.

To initialize multidimensional array, you can do the following:

```
int arr[2][2] = {{1,2},{3,4}};
```

5.3 Pre-lab Assignment

1. TWrite a program that accepts input numbers 1 to 9 from the user, then inserts all the numbers into an array!
2. What would happen if an array `arr` is accessed with `arr[-1]`?
3. What would happen if an array `arr` with size 5 is accessed with `arr[5]`?
4. Look at the following code

```
for(i=0;i<10;i++){
    for(j=i;j<10;j++){
        printf("A");
```

```
        }
    }
```

How many "A" will be printed on the screen if that block of code is executed?

6 String

In general, a string is a collection of one or more characters. Specifically in the C language, a string is defined as a collection of characters terminated by a null character.`'\0'`.
For example, string `"Dasar"`, in C programming language can be represented in a collection of character `'D'`, `'a'`, `'s'`, `'a'`, `'r'`, dan `'\0'`.

6.1 String Uses

Because a string is essentially an array of characters, creating a string data type in C follows the same approach as creating an array. Here's an example:

Listing 10: Char in string implementation

```c
#include <stdio.h>

int main(void)
{
char foo[8] = {'b','e','l','a','j','a','r','\0'};
printf("Isi variabel foo adalah %s \n", foo);

return 0;
}
```

`'\0'` is one of requirement in order to create a string in C programming language. Every string need a "special" character to indicates it's end. This `'\0'` represented null characther which use in C programming compiler as an indication of the end of the string

Source code implementation `scanf` to read string:

Listing 11: String with scanf implementation

```c
#include <stdio.h>

int main() {
  // Declare variable to store input from user
  int age;
  float height;
  char name[50];

  // Request user to input their age
  printf("Enter your age: ");
  scanf("%d", &age);

  // Request user to input their height
  printf(Enter your height (in meter): ");
  scanf("%f", &height);
```

```
16
17      // Request user to enter their name
18      printf("Enter your name: ");
19      scanf("%s", name);
20
21      // Display user information
22      printf("Name: %s\n", name);
23      printf("Age: %d year old\n", age);
24      printf("Height: %.2f meter\n", height);
25
26      return 0;
27    }
```

Source code example `gets` to read string:

Listing 12: String with gets implementation

```
1  #include <stdio.h>
2
3  int main () {
4
5    char arr[100];
6    while(true)
7    {
8      gets(arr);
9
10     printf("-- %s\n", arr);
11   }
12   return 0;
13
14 }
```

String that read using scanf or gets will automatically has `null` character in the end.

6.2 String Functions

In the C programming language, there is a library created with the purpose of facilitating users in string manipulation. This library is stored in `<string.h>`, therefore, to access this library, an additional preprocessor directive is required, which is::

```
1    #include <string.h>
```

Learn other function in www.cplusplus.com.

6.3 Pre-lab Assignment

1. Create a program in C programming language that takes 2 string from the user input and decide whether those 2 string are an anagram (contains the same characters even in different order). For example "night" and "thing".

2. Explain the difference between string that is declared as an array of charater (char array) and a string that is declared as a string data types (string literal). Explain example of using both

3. Name 5 functions from the `string.h` library! explain each function!

4. To get string output, instead of using `printf()` we can also use `puts()`. Explain the advantages of using `puts()` compared to `printf()`!