


Soal Praktikum <i>Practicum Case</i>	
COMP6360004 Algorithm and Programming	
Teknik Informatika <i>Computer Science</i>	E222-COMP6360004-EP01-05
Periode Berlaku Semester Ganjil 2022/2023 Valid on Odd Semester Year 2022/2023	Revisi 00 <i>Revision 00</i>

Learning Outcomes

- Apply syntax and functions in C language in problem solving
- Construct a program using C language in problem solving

Topic

- Session 05 - Pointers and Array

Sub Topics

- Array
- Multidimensional Array
- Strings concept in C using Array of Char
- String manipulation
- Pointer concept
- Create program using 1D Array

Tutorial/Panduan

Arrays are data structures consisting of related data items of the same type. Arrays are “static” entities in that they remain the same size throughout program execution. An array is a group of memory locations related by the fact that they all have the same name and the same type. To refer to a particular location or element in the array, specify the name of the array and the position number of the particular element in the array. The first element in every array is the zeroth element. Thus, the first element of array *c* is referred to as *c*[0], the second element of array *c* is referred to as *c*[1], the seventh element of array *c* is referred to as *c*[6], and, in general, the *i*th element of array *c* is referred to as *c*[*i* - 1]. Array names, like other variable names, can contain only letters, digits and underscores. Array names cannot begin with a digit. The position number contained within square brackets is more formally called a subscript. A subscript must be an integer or an integer expression. The brackets used to enclose the subscript of an array are actually considered to be an operator in C. They have the same level of precedence as the function call operator. Arrays occupy space in memory. You specify the type of each element and the number of elements in the array so that the computer may reserve the appropriate amount of memory. An array of type *char* can be used to store a character string.

An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it. `int data[100];`

How to declare an array? `dataType arrayName[arraySize];`

For example, `float mark[5];`

we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values. It's important to note that the size and type of an array cannot be changed once it is declared.

To access array elements, you can access elements of an array by indices.

Suppose you declared an array mark as above.

The first element is mark[0],

the second element is mark[1] and so on.

mark[0] mark[1] mark[2] mark[3] mark[4]

--	--	--	--	--

- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]
- Suppose the starting address of mark[0] is 2120d. Then, the address of the mark[1] will be 2124d. Similarly, the address of mark[2] will be 2128d and so on because the size of a float is 4 bytes.

How to initialize an array? It is possible to initialize an array during declaration.

For example,

`int mark[5] = {19, 10, 8, 17, 9};`

mark[0] mark[1] mark[2] mark[3] mark[4]

19	10	8	17	9
----	----	---	----	---

You can also initialize an array like this

`int mark[] = {19, 10, 8, 17, 9};`

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9

Example of C program using Array in Input/Output

```

1 // Program to take 5 values from the user and store them in an array
2 // Print the elements stored in the array
3 #include <stdio.h>
4
5 int main() {
6     int values[5];
7
8     printf("Enter 5 integers: ");
9
10    // taking input and storing it in an array
11    for(int i = 0; i < 5; ++i) {
12        scanf("%d", &values[i]);
13    }
14
15    printf("Displaying integers: ");
16
17    // printing elements of an array
18    for(int i = 0; i < 5; ++i) {
19        printf("%d\n", values[i]);
20    }
21    return 0;
22 }
```

Output

```

Enter 5 integers: 1
-3
34
0
3
Displaying integers: 1
-3
34
0
3
```

These arrays are called one-dimensional arrays.

In C programming, you can create an array of arrays.

These arrays are known as multidimensional arrays.

For example, `float x[3][4];`

x is a two-dimensional (2D) array that can hold 12 elements.

Imagine array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Similarly, you can declare a three-dimensional (3d) array. For example, `float y[2][4][3];`

The array y can hold 24 elements.

Here is how you can initialize two-dimensional arrays:

```
// Different ways to initialize two-dimensional array
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

An example of program using 2D arrays:

```
1  #include <stdio.h>
2
3  int main()
4  { int arr[2][3] = {12, 20, 34, 46, 11, 18};
5    int i, j;
6
7    for(i=0; i<2; i++)
8    { printf("\n");
9
10     for(j=0; j<3; j++)
11     printf(" %d", arr[i][j]);
12   }
13   return 0;
14 }
```

```
12 20 34
46 11 18
```

In C programming, a **string** is a sequence of characters terminated with a null character `\0`.

For example: `char c[] = "c string";`

c		s	t	r	i	n	g	\0
---	--	---	---	---	---	---	---	----

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character `\0` at the end by default.

Here's how you can declare strings, for example: `char s[5];`

Here, we have declared a string of 5 characters.

s[0]	s[1]	s[2]	s[3]	s[4]

You can initialize strings in a number of ways.

```
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

Here is how to assign values to strings. Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared. For example,

```
char c[100];
c = "C programming"; // Error! array type is not assignable.
```

Note: Use the **strcpy()** function to copy the string instead.

How to read string from the user? You can use the **scanf()** function to read a string. The **scanf()** function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

```
#include <stdio.h>
int main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;
}
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Even though *Dennis Ritchie* was entered in the above program, only "*Dennis*" was stored in the name string. It's because there was a **space** after Dennis. Also notice that we have used the code name instead of **&name** with **scanf()**.

```
scanf("%s", name);
```

This is because name is a char array, and we know that array names decay to pointers in C.

Thus, the name in **scanf()** already points to the address of the first element in the string, which is why we don't need to use **&**.

How to read a line of text?

You can use the **fgets()** function to read a line of string. And, you can use **puts()** to display the string.

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin); // read string
    printf("Name: ");
    puts(name); // display string
    return 0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Commonly Used String Functions:

strlen() - calculates the length of a string

strcpy() - copies a string to another

strcmp() - compares two strings

strcat() - concatenates two strings

• Example of String Manipulation

```
char s1[ ] = "abcdef";
char s2[ ] = "xyz";

strlen("nana");           // 4
strcmp("nana", "nana")    // result 0
strcpy(s1,s2);            // s1 = "xyz", s2 = "xyz"
strncpy(s1,s2,2);         // s1 = "xycdef", s2 = "xyz"
strncpy(s1,s2,4);         // if n>=strlen(s2) similar with
                           // strcpy()    s1 = "xyz"
strcat(s1,s2);            // s1="abcdefxyz", s2="xyz"
strncat(s1,s2,2);         // s1="abcdefxy", s2="xyz"

s1 = "Happy"
s2 = "New Year"

strcat( s1, s2 )          // s1= "Happy New Year"
strncat( s3, s1, 6 )      // s1= "Happy"
strcat( s3, s1 )          // s1= "Happy Happy New Year"
```

• Example of String Manipulation (Copy String)

```
/* Copy string */
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "Copy a string.";
    char str2[15];
    char str3[15];
    int i;

    strcpy(str2, str1);    // with strcpy()

    for (i=0; str1[i]; i++) // without strcpy()
        str3[i] = str1[i];
    str3[i] = '\0';

    /* print out str2 and str3 */
    printf("The content of str2: %s\n", str2);
    printf("The content of str3: %s\n", str3);
}
```

Pointer

A pointer contains an address of another variable that contains a value. In this sense, a variable name directly references a value, and a pointer indirectly references a value. Referencing a value through a pointer is called indirection. Pointers can be defined to point to objects of any type. Pointers should be initialized either when they're defined or in an assignment statement. A pointer may be initialized to NULL, 0 or an address. A pointer with the value NULL points to nothing. Initializing a pointer to 0 is equivalent to initializing a pointer to NULL, but NULL is preferred. The value 0 is the only integer value that can be assigned directly to a pointer variable. NULL is a symbolic constant defined in the <stddef.h> header (and several other headers).

As the pointer operator, the &, or address operator, is a unary operator that returns the address of its operand. The operand of the address operator must be a variable. The indirection operator * returns the value of the object to which its operand points.

Pointers (pointer variables) are special variables that are used to store addresses rather than values. How we can declare pointers? `int* p;` Here, we have declared a pointer p of int type.

You can also declare pointers in these ways.

```
int *p1;
int * p2;
```

Let's take another example of declaring pointers.

```
int* p1, p2;
```

Here, we have declared a pointer p1 and a normal variable p2.

How to assign addresses to pointers?

Let's take an example.

```
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the c variable.

And, the address of c is assigned to the pc pointer.

To get the value of the thing pointed by the pointers, we use the * operator. For example:

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc); // Output: 5
```

Here, the address of c is assigned to the pc pointer.

To get the value stored in that address, we used *pc.

In this example, pc is a pointer, not *pc.

You cannot and should not do something like *pc = &c;

By the way, * is called the dereference operator (when working with pointers).

It operates on a pointer and gives the value stored in that pointer.

An example of program using pointer

```
1 #include <stdio.h>
2 int main()
3 {
4     int* pc, c;
5
6     c = 22;
7     printf("Address of c: %p\n", &c);
8     printf("Value of c: %d\n\n", c); // 22
9
10    pc = &c;
11    printf("Address of pointer pc: %p\n", pc);
12    printf("Content of pointer pc: %d\n\n", *pc); // 22
13
14    c = 11;
15    printf("Address of pointer pc: %p\n", pc);
16    printf("Content of pointer pc: %d\n\n", *pc); // 11
17
18    *pc = 2;
19    printf("Address of c: %p\n", &c);
20    printf("Value of c: %d\n\n", c); // 2
21    return 0;
22 }
```

Output

```
Address of c: 2686784
Value of c: 22
```

```
Address of pointer pc: 2686784
Content of pointer pc: 22
```

```
Address of pointer pc: 2686784
Content of pointer pc: 11
```

```
Address of c: 2686784
Value of c: 2
```

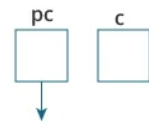
Explanation of the program:

1. `int* pc, c;`

A pointer variable and a normal variable is created.

Here, a pointer pc and a normal variable c, both of type int, is created.

Since pc and c are not initialized at initially, pointer pc points to either no address or a random address. And, variable c has an address but contains random garbage value.

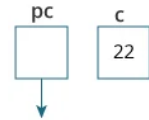


2. `c = 22;`

22 is assigned to variable c.

This assigns 22 to the variable c.

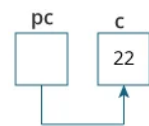
That is, 22 is stored in the memory location of variable c.



3. `pc = &c;`

Address of variable c is assigned to pointer pc.

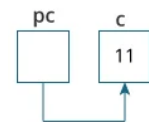
This assigns the address of variable c to the pointer pc.



4. `c = 11;`

11 is assigned to variable c.

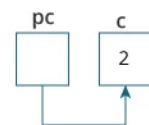
This assigns 11 to variable c.



5. `*pc = 2;`

5 is assigned to pointer variable's address.

This change the value at the memory location pointed by the pointer pc to 2.



Soal

Case

- Create a C program to find largest element in an array
- Solve all of this week's exercises that are available on <https://socs1.binus.ac.id/quiz/>

References:

Paul Deitel & Harvey Deitel. (2016). C how to program:with an introduction to C++. 08. Pearson Education. Hoboken.ISBN: 9780133976892.
<https://www.programiz.com/c-programming/c-arrays>
<https://www.programiz.com/c-programming/c-multi-dimensional-arrays>
<https://www.programiz.com/c-programming/c-strings>
<https://www.programiz.com/c-programming/c-pointers>