Computer Practicum 1

Introduction to C

Vida Groznik

Source: cprogramming.com, tutorialspoint.com, learn-c.org, fresh2refresh.com

case control statements

The statements which are used to execute only specific block of statements in a series of blocks are called case control statements.

There are 4 types of case control statements in C language. They are,

1. switch
2. break
3. continue
4. goto

switch statement

Switch case statements are used to execute only specific case statements based on the switch expression.

```
switch (expression)
{
     case label1:   statements;
                          break;
     case label2:   statements;
                          break;
     case label3:   statements;
                          break;
     default:      statements;
                          break;
}
```

Output:

```
Value is 3
```

```
1   #include <stdio.h>
2
3   int main ()
4   {
5      int value = 3;
6      switch(value)
7      {
8        case 1:
9        printf("Value is 1 \n" );
10       break;
11
12       case 2:
13       printf("Value is 2 \n" );
14       break;
15
16       case 3:
17       printf("Value is 3 \n" );
18       break;
19
20       case 4:
21       printf("Value is 4 \n" );
22       break;
23
24       default :
25       printf("Value is other than 1,2,3,4 \n" );
26     }
27     return 0;
28  }
```

break statement

Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.

Syntax: `break;`

```c
#include <stdio.h>
int main()
{
   int i;

   for(i=0;i<10;i++)
   {
     if(i==5)
     {
        printf("\nStop loop when i = 5");
        break;
     }
     printf("%d ",i);
   }
}
```

Output:

```
0 1 2 3 4
Stop loop when i = 5
```

continue statement

Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.

Syntax: `continue;`

```c
#include <stdio.h>
int main()
{
   int i;
   for(i=0;i<10;i++)
   {
     if(i==5 || i==6)
     {
       printf("\nSkipping %d from" \
       "display using " \
       "continue statement \n",i);
       continue;
     }
     printf("%d ",i);
   }
}
```

Output:

```
0 1 2 3 4
Skipping 5 from display using continue statement
Skipping 6 from display using continue statement
7 8 9
```

goto statement

goto statement is used to transfer the normal flow of a program to the specified label in the program.

```
{
    ......
    goto LABEL;
    ......
    ......
    LABEL:
    statements;
}
```

```
#include <stdio.h>
int main()
{
   int i;
   for(i=0;i<10;i++)
     {
   if(i==5)
   {
     printf("\nWe are using goto statement
when i = 5");
     goto HAI;
   }
   printf("%d ",i);
}

HAI : printf("\nNow, we are inside label
name \"hai\" \n");
}
```

Output:

```
0 1 2 3 4
We are using goto statement when i = 5
Now, we are inside label name „hai“
```

Constants

C Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.

Constants refer to fixed values. They are also called as literals

Constants may be belonging to any of the data type.

We can define constants in a C program in the following ways.

- By “const” keyword

```
const data_type variable_name
```

- By “#define” preprocessor directive

Constants (2)

```c
#include <stdio.h>
void main()
{
const int height = 100;
const float number = 3.14;
const char letter = 'A';
const char letter_sequence[10] = "ABC";
const char backslash_char = '\?';
printf("value of height :%d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n",
letter_sequence);
printf("value of backslash_char : %c \n",
backslash_char);
}
```

```c
#include <stdio.h>
#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'
void main()
{
printf("value of height : %d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s
\n",letter_sequence);
printf("value of backslash_char : %c
\n",backslash_char);
}
```

Array

Array is a collection of variables belonging to the **same data type**. You can store group of data of same data type in an array.

- Array can belong to any of the data types.
- Array size must be a **constant** value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

- There are 2 types of C arrays. They are,
- **One dimensional** array
- **Multi dimensional** array
  - Two dimensional array
  - Three dimensional array
  - four dimensional array etc…

One dimensional array

Array declaration syntax:

```
data_type arr_name [arr_size];
```

Array initialization syntax:

```
data_type arr_name [arr_size]=(value1,
        value2, value3,….);
```

Array accessing syntax:

```
arr_name[index];
```

Integer array example:

```
int age [5];
int age[5]={0, 1, 2, 3, 4};

age[0]; /*0 is accessed*/
age[1]; /*1 is accessed*/
age[2]; /*2 is accessed*/
```

Character array example:

```
char str[10];
char str[10]={‘H’,‘a’,‘i’};
(or)
char str[0] = ‘H’;
char str[1] = ‘a’;
char str[2] = ‘i’;

str[0]; /*H is accessed*/
str[1]; /*a is accessed*/
str[2]; /*i is accessed*/
```

One dimensional array - example

```c
#include<stdio.h>

int main()
{
   int i;
   int arr[5] = {10,20,30,40,50};

   // declaring and Initializing array in C
   //To initialize all array elements to 0, use int
arr[5]={0};
   /* Above array can be initialized as below also
      arr[0] = 10;
      arr[1] = 20;
      arr[2] = 30;
      arr[3] = 40;
      arr[4] = 50; */

   for (i=0;i<5;i++)
   {
      // Accessing each variable
      printf("value of arr[%d] is %d \n",i,arr[i]);
   }
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

Two dimensional array

Array declaration syntax:

```
data_type arr_name [num_of_rows][num_of_column];
```

Array initialization syntax:

```
data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}};
```

Array accessing syntax:

```
arr_name[index];
```

Integer array example:

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

arr[0][0] = 1;
arr[0][1] = 2;
arr[1][0] = 3;
arr[1][1] = 4;
```

Two dimensional array - example

```c
#include<stdio.h>

int main()
{
   int i,j;
   // declaring and Initializing array
   int arr[2][2] = {10,20,30,40};
   /* Above array can be initialized as below also
      arr[0][0] = 10; // Initializing array
      arr[0][1] = 20;
      arr[1][0] = 30;
      arr[1][1] = 40; */
   for (i=0;i<2;i++)
   {
      for (j=0;j<2;j++)
      {
         // Accessing variables
         printf("value of arr[%d] [%d] : %d\n",i,j,arr[i][j]);
      }
   }
}
```

Output:

```
value of arr[0] [0] is 10
value of arr[0] [1] is 20
value of arr[1] [0] is 30
value of arr[1] [1] is 40
```

String

Strings are an array of characters ended with null character (‘\0’).

This null character indicates the end of the string.

Strings are always enclosed by **double** quotes. Whereas, character is enclosed by **single** quotes in C.

```
char string[10] = {‘f’, ’a’, ‘m’, ‘n’, ‘i’, ‘t’, ‘\0’};
```
(or)

```
char string[10] = “famnit”;
```
(or)

```
char string [] = “famnit”;
```

Difference between above declarations are, when we declare char as “`string[10]`”, 10 bytes of memory space is allocated for holding the string value.

When we declare char as “`string[]`”, memory space will be allocated as per the requirement during execution of the program.

```c
#include <stdio.h>

int main ()
{
   char string[10] = "famnit";

   printf("The string is : %s \n", string );
   return 0;
}
```

Output:

```
The string is : famnit
```

Command line arguments

`main()` function of a C program accepts arguments from command line or from other shell scripts by following commands. They are,

`argc`
`argv[]`

where,

`argc` – Number of arguments in the command line including program name
`argv[]` – This is carrying all the arguments

In real time application, it will happen to pass arguments to the main program itself. These arguments are passed to the `main()` function while executing binary file from command line.

```
./test this is a program
```

Where,

```
argc    = 5
argv[0] = “test”
argv[1] = “this”
argv[2] = “is”
argv[3] = “a”
argv[4] = “program”
argv[5] = NULL
```

Command line arguments (2)

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])   //  command line arguments
{
if(argc!=5)
{
   printf("Arguments passed through command line " \
          "not equal to 5");
   return 1;
}

   printf("\n Program name  : %s \n", argv[0]);
   printf("1st arg  : %s \n", argv[1]);
   printf("2nd arg  : %s \n", argv[2]);
   printf("3rd arg  : %s \n", argv[3]);
   printf("4th arg  : %s \n", argv[4]);
   printf("5th arg  : %s \n", argv[5]);

return 0;
}
```

Output:

```
Program name : test
1st arg : this
2nd arg : is
3rd arg : a
4th arg : program
5th arg : (null)
```

Exercise 1

Write a program which has student‘s grades stored in an array. Enter 3 grades in an array and calculate the average.

Exercise 2

Enter 10 numbers in an array and print out the eighth number.

Exercise 3

Let us try to find out the average marks of a group of five students for two subjects, Mathematics and Programming. To do this, we use a two-dimensional array called `grades`. The marks corresponding to Mathematics would be stored in the first row (`grades[0]`), whereas those corresponding to Programming would be stored in the second row (`grades[1]`). Complete the following steps:

- Declare grades as a two-dimensional array of integers
- Compute the average marks obtained in each subject

Exercise 4

Write a programme which has your name and your age saved in respective variables. Print out your name and your age in the same sentence.

Print out the length of your name which is stored in a string variable.

Exercise 5

Multiply and sum all items (array[0] to array[9], inclusive), of the variable array.

Exercise 6

Ask a user to enter a string. (use gets() function)
and check whether a tring is a palindrome.