Programming fundamentals

Lecture 10: Constant variables and Arrays

Recap

- Loops
- Pattern designing through loops

Agenda

- Constant variables
- Arrays

Recap

- We've already learnt that to store anything we need a memory location
- Then, we explored various types of variables that define the type of data to be stored in memory. E.g. **int variable** indicates that memory location hold by this variable can store the data that has type ***INTEGERS***.
- Just to recap:

  int sum=43;

- The statement means a location on memory is allocated (with some address e.g. **0500**). **sum** is used to identify that location and it holds a value **43**

Constants

- In many scenarios, we have to deal with values that remain fixed throughout our algorithm or program.
- **E.g.** *interest rate in banking applications remain fixed. We may accidently change this variable's value. To avoid that error scenario, in C++ we have* ***constants*** *that deal with this issue.*
- Constants refer to as fixed values, unlike variables whose value can be altered, constants - as the name implies does not change, they remain constant.
- Constant must have to be initialized at the time of creating it, and new values cannot be assigned later to it.
- There are two other different ways to define constants in C++. These are:
  - By using *const* keyword
  - By using *#define* preprocessor

Using const keyword

- Const type const_name;
- It is also possible to put const either before or after the type.

int const SIDE = 50;

or

const int SIDE = 50;

```
#include <iostream>
using namespace std;
int main()
{
    const int SIDE = 50;
    int area;
    area = SIDE*SIDE;
    cout<<"The area of the
    square with side: " << SIDE
    <<" is: " << area << endl;
    return 0;
}
```

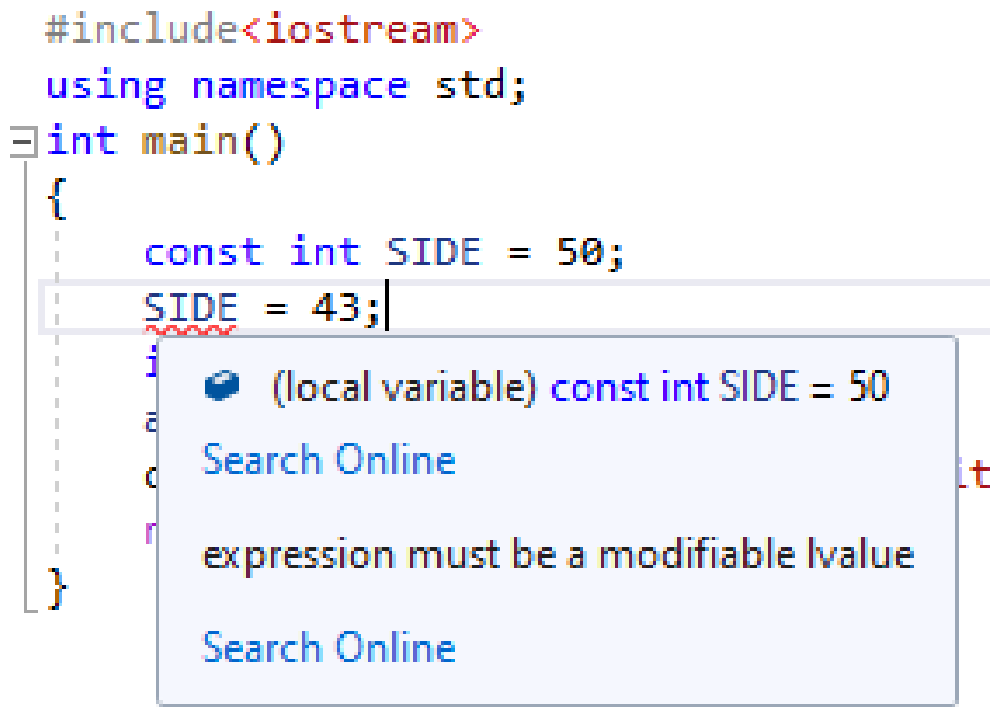
#define

- **#define** keyboard is used to define constant by specifying its name and value, separated by spaces:
- #define KILLBONUS 5000
- The constant does not have a type such as int or char.
- The #define directive enables a simple text substitution that replaces every instance of KILLBONUS in the code with 5000.
- The compiler sees only the end result.

```
#include <iostream>
using namespace std;
#define VAL1 20
#define VAL2 6
#define Newline '\n’
int main()
{
    int tot;
    tot = VAL1 * VAL2;
    cout << tot; cout << Newline;
}
```

Extra notes

- Constant variables MUST always be initialized (defined) on declaration time.
- If you try to redefine (re-assign) a constant variable AFTER the code line with declaration, compiler will generate error
- E.g. in the given program, an integer variable named **SIDE** is declared as a constant variable. In second line, SIDE is again assigned value ***43***. As this is not allowed therefore error will be generated.



Contd..

- Constant variables MUST always be defined during declaration.
- If we skip definition, then error will be generated like shown in the given example.

```
#include<iostream>
using namespace std;
int main()
{
    const int SIDE;
    int area;
    area = SIDE * S
    cout << "The ar
    return 0;
}
```

const variable "SIDE" requires an initializer

Search Online

Arrays

- When we studied variables, we have seen that a variable is allocated one memory cell location when it is defined.
- Variable is just one cell of memory (with some specific address) whereas array is a collection of consecutive memory cells.

Variable A
0001

Cell 1	Cell 2	Cell 3	Cell 4
0010	0011	0012	0013

Memory location

Memory location's address

- The word **consecutive** is important! Array is always consecutive. For example, following is not a valid array!

This is because after 0010, the immediate next memory location (0011) must be there as a second cell, **not 0051**

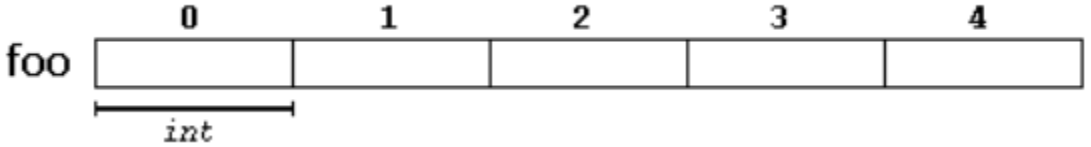
Cell 1	Cell 2	Cell 3	Cell 4
0010	0051	0012	0013

- Another important point is:
  - All the cells in array are of same data type. For example, we cannot have array like the following:

Cell 1	**Cell 2**	**Cell 3**	**Cell 4**
0010	0051	0012	0013
Int type memory location	Int type memory location	double type memory location	char type memory location

Contd..

- An array containing 5 integer values of type int called foo could be represented as:



- The elements field within square brackets [], representing the number of elements in the array, must be a constant expression
- In this case, these are values of type int.
- These elements are numbered from 0 to 4, being 0 the first and 4 the last
- In C++, the first element in an array is always numbered with a zero (not a one), no matter its length.
- Like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

type name [elements]; int foo [5];

Integer arrays: Declaration

- For now, we will focus on **integer type arrays**. In future, we will see that we can define other type of arrays also i.e. char array, double array.
- To declare an array the basic format is:

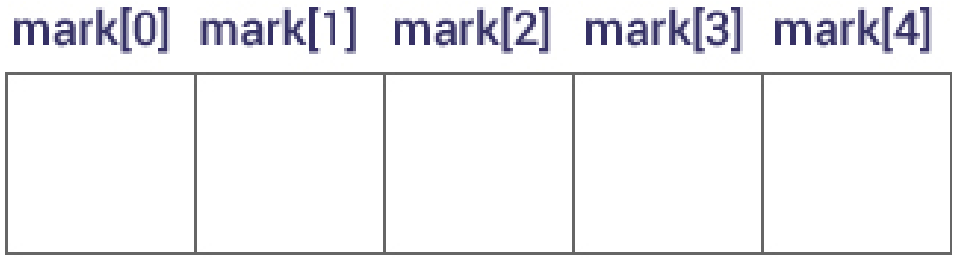
Data type variable name [size of array]

e.g. int age[5];

- Just as we name variables, we can give a meaningful name to an array. In this example an array is created that has the capacity of five. In other words, we can store the age of five persons in it.
- **Declare an array:** int age [5];
- **Declare and definition:** int age [5] = { 16, 2, 77, 40, 11 }; To define at the same time when we declare, we need to specify values in comma separated format and put them into curly brackets.

Access elements of array

- You can access elements of an array by using indices.
- Suppose you declared an array mark as above. The first element is mark[0], second element is mark[1] and so on.



Storing data in an array + print an array

- Just as in case of variable, to access the location we use variable name for correspondence.
- In array, to access a particular element of an array we need to specify **array index. Index** is an integer assigned to each cell of array. It starts from 0 and goes to n-1. That is, to access third location we need to write **age[2].** As the index is starting from 0 so index number will always be one less than the location number. (i.e. 3-1=2 in this case.) Last element's index is always n-1.
- Rest of the procedure is the same as we have done with variable. The only difference is that for arrays, we need to access only one element at a time.

Variable vs arrays

```
int age[3];
age[0]=22;
age[1]=20;
age[2]=45;
cout<< age[0];
cout<< age[1];
cout<< age[2];
```

age			
	22	20	45
	0	1	2

```
int age1, age2, age3;
age1 =22;
age2 =20;
age3 =45;
cout<< age1;
cout<< age2;
cout<< age3;
```

Practice program

- Write a program to take marks of four students as an input and then display their sum. Use array to store marks.

```
#include<iostream>
using namespace std;
int main()
{
    int marks[4]; int sum = 0;
    cout << "Enter the marks of first student: " <<
    endl;
    cin >> marks[0];
    cout << "Enter the marks of second student: " <<
    endl;
    cin >> marks[1];
    cout << "Enter the marks of third student: " <<
    endl;
    cin >> marks[2];
    cout << "Enter the marks of fourth student: " <<
    endl;
    cin >> marks[3];
    sum = marks[0] + marks[1] + marks[2] + marks[3];
    cout << "SUM: " << sum<< endl;
return 0;
}
```

Recommended reads

- Walter Savitch, Problem Solving with C++ The Object of Programming
  - Chapter 7, page 378, section 7.1