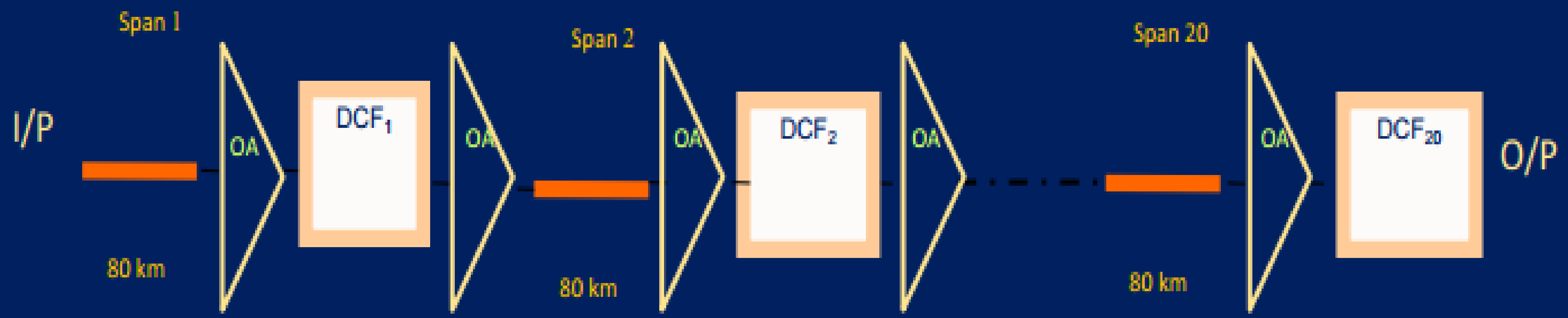
Programming Fundamentals- I

Rao Muhammad Umer
Lecturer,
Web: raoumer.github.io
Department of Computer Science & IT,
The University of Lahore.

Administrative Stuff

- Due Date for Assignment # 2 : PF – I Basic Concepts
  - **4/12/2016**

20-Spans System



Loops

For Loop

Printing the counting from 1 to 10

```
#include<iostream>
using namespace std;
int main()
{
    int i;
    for (i = 1; i <= 10; i++)
        cout << “i= \n” << i;
    getch();
    return 0;
}
```

The for Statement

- The most important looping structure in C/C++.
- Generic Form:

```
        for (initial;  condition;  increment )
                statement
```

- initial, condition, and increment are C++ expressions.
- For loops are executed as follows:
  1. Initial is evaluated. Usually an assignment statement.
  2. Condition is evaluated. Usually a relational expression.
  3. If condition is false (i.e. 0), fall out of the loop (go to step 6.)
  4. If condition is true (i.e. non zero), execute statement
  5. Execute increment and go back to step 2.
  6. Next statement

The for Statement

For statement examples

```
#include <iostream>
using namespace std;
int main () {
    int count;
    /* 1. simple counted for loop */
    for (count =1; count <=20;
    count++)
    cout << "\n", count;
    /* 2. counting backwards */
    for (count = 100; count >0;
    count--)
    cout << "count= ", count;
    /* 3. for loop counting by 5's */
    for (count=0; count<1000;
    count += 5)
    cout << "count= ", count;
    /* 4. initialization outside of
    loop */
    count = 1;
    for ( ; count < 1000; count++)
    cout << " \n", count;
    getch();
    return 0;
}
For ( ; ; )
{
}
```

The for Statement

```
#include <iostream>
int main ()
{
	int count;
	int x, y;
	/* 5. compound statements for initialization and increment */
	for (x=0, y=100; x<y; x++, y--)
	{
			cout << "\n" << x << y;
	}
	getch();
	return 0;
}
```

The for Structure: Observations

- Arithmetic expressions

  Initialization, loop-continuation, and increment can contain arithmetic expressions. If x equals 2 and y equals 10

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

While Loop

Printing the counting from 1 to 10

```
#include <iostream>
using namespace std;
int main()
{
      int i= 1;
      while (i<= 10)
            {
            cout << "i=\n“ << i;
            i++;
            }
      getch();
      return 0;
}
```

The while Statement

- Generic Form

```
while (condition)
    statement
```

- Executes as expected:

1. condition is evaluated
2. If condition is false (i.e. 0), loop is exited (go to step 5)
3. If condition is true (i.e. nonzero), statement is executed
4. Go to step 1
5. Next statement

- Note:

```
for (exp1; exp2; exp3) stmt;
```

is equivalent to

```
exp1;
while(exp2) { stmt; exp3; }
```

Do while Loop

Counting from 1 to 10

```
#include <iostream>
using namespace std;
int main()
{
    int i= 1;
    do
        {
            cout << "i= \n“ << i;
            i++;
        } while (i<= 10) ; // can also be written as while ( ++i<= 10)
    getch();
    return 0;
}
```

The do while Loop

- The do/while repetition structure
  - Similar to the while structure
  - Condition for repetition tested after the body of the loop is performed
    - All actions are performed at least once
  - Generic Format:

```
do {
        statement;
} while ( condition );
```

The do while Loop

- Standard repeat until loop
- Like a while loop, but with condition test at bottom.
- Always executes at least once.
- The semantics of do...while:
    1. Execute statement
    2. Evaluate condition
    3. If condition is true go to step 1
    4. Next statement

Comparison of for, while, and do-while loops

Loops

```
for (i = 0; i < 12; i++)
   {
     dowork();
   }
while ( i < 12)
   {
     dowork();
     i++;
   }
do
   {
     dowork();
     i++;
   } while (i < 12); // use of semicolon
```

for loop

- Preferably used where exact number of iterations are known in prior.

while loop

- Preferably used where exact number of iterations are not known in prior.
- Infinite iteration

do while loop

- Preferably used where loop should must run atleast once.

Arrays

Introduction to Arrays

```
int marks = 70;
```

```
int marks[6] = {36,78,29,36,7,99};
```

Introduction to Arrays

```
#include<iostream>
int main()
{
        int marks = 70; // single value
        cout << “Marks are“ << marks;
        /* single value will be printed */
        getch();
        return 0;
}
```

Arrays

```
#include<iostream>
int main()
{
        int marks[6]={36,78,29,89,7,99}; // array
        cout << "Marks are \n“ << marks; // error
        getch();
        return 0;
}
```

How to Print Arrays?

```
#include<iostream>
int main()
{
            int marks[6] = {36,78,29,89,7,99};
            int i;
            for (i= 0; i<6; i++) //i++ means i= i+1
            {
                        cout << “Marks are\n“ << marks[i];
            }
            getch();
            return 0;
}
```

How to read values of Arrays?

```
#include<iostream>
int main()
{
        int marks[6];
        int i;
        for (i= 0; i<6;i++) //i++ means i= i+1
        {
                cout << “Please enter the marks of students\n”;
                cin >> marks[i];
        }
        getch();
        return 0;
}
```

Single-Dimensional Arrays

Generic declaration:

typename variablename[size]

- typename is any type
- variablename is any legal variable name
- size of array

- For example

  int a[10];

- Defines an array of ints with subscripts ranging from 0 to 9

  a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

- There are 10*sizeof(int) bytes of memory reserved for this array.
- You can use a[0]=10; x=a[2]; a[3]=a[2]; etc.

Initializing Arrays

- Initialization of arrays can be done by a comma separated list following its definition.
- For example:

```
int array [4] = { 100, 200, 300, 400 };
```

  - This is equivalent to:

```
int array [4];
array[0] = 100;
array[1] = 200;
array[2] = 300;
array[3] = 400;
```

- You can also let the compiler figure out the array size for you:

```
int array[] = { 100, 200, 300, 400};
```

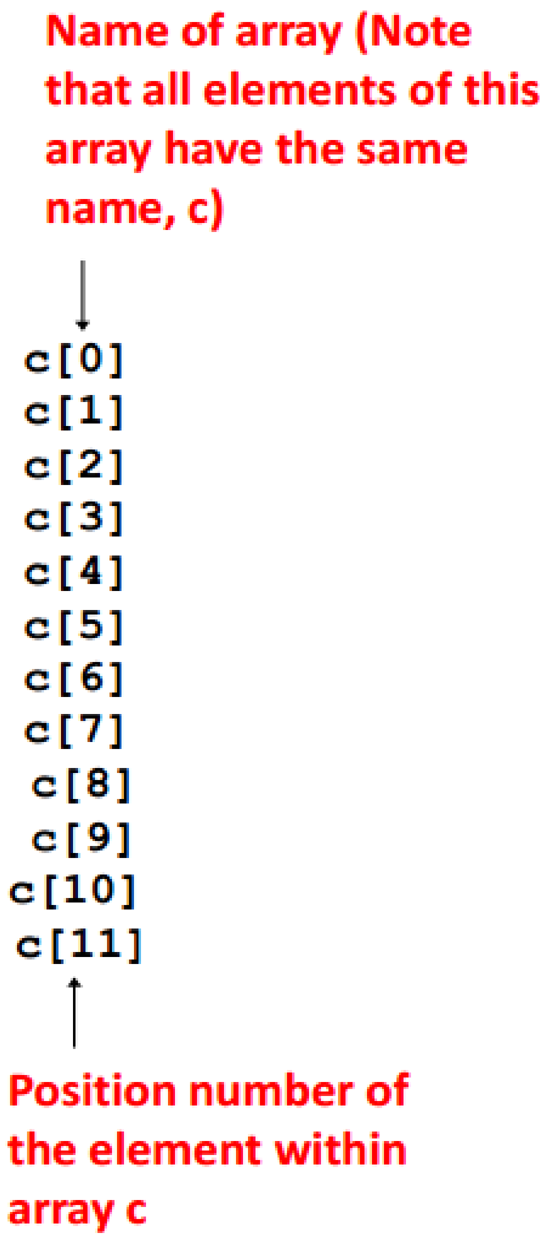
A Simple Example

```
#include <iostream>
int main() {
float expenses[12]={10.3, 9, 7.5, 4.3, 10.5, 7.5, 7.5, 8, 9.9, 10.2, 11.5, 7.8};
int count, month;
float total;
for (month=0, total=0.0; month < 12; month++)
{
		total+=expenses[month];
}
for (count=0; count < 12; count++)
cout << "Month:“ << count +1 << “Rupees:“<<expenses[count]<< endl;
cout << "Total:” << total <<“Rs”<< “Average = total/12” << total/12<<“Rs”<<endl;
return 0;
}
```

Arrays

- Array
  - Structures of related data items
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

  arrayname [*position number*]
  - First element at position 0
  - **`n`** element array named **`c`**:
    - **`c[ 0 ], c[ 1 ]...c[ n – 1 ]`**


Name of array (Note that all elements of this array have the same name, c)
c[0]
c[1]
c[2]
c[3]
c[4]
c[5]
c[6]
c[7]
c[8]
c[9]
c[10]
c[11]
Position number of the element within array c

Arrays

- Array elements are like normal variables

```
c[ 0 ] = 3;
cout << c[ 0 ];
```

- Perform operations in subscript. If x equals 3

```
c[ 5 -2 ] == c[ 3 ] == c[ x ]
```

Declaring Arrays

- When declaring arrays, specify
  - Name
  - Type of array
  - Number of elements

    ```
    arrayType arrayName[numberOfElements ];
    ```
  - Examples:

    ```
    int c[ 10 ];
    float myArray[ 32 ];
    ```
- Declaring multiple arrays of same type
  - Format similar to regular variables
  - Example:

    ```
    int b[ 100 ], x[ 27 ];
    ```

Examples Using Arrays

- • Initializers

  int n[5] = { 1, 2, 3, 4, 5 };

  – If not enough initializers, rightmost elements become 0

  int n[5] = { 1 }

- All other elements 0
  – C arrays have no bounds checking
- If size omitted, initializers determine it

  int n[ ] = { 1, 2, 3, 4, 5 };

  – 5 initializers, therefore 5 elements array