PROGRAM CONTROL

SDN 150S – Week 3

Lecturer: Mr. Stephen Ekwe

Cape Peninsula University of Technology

creating futures

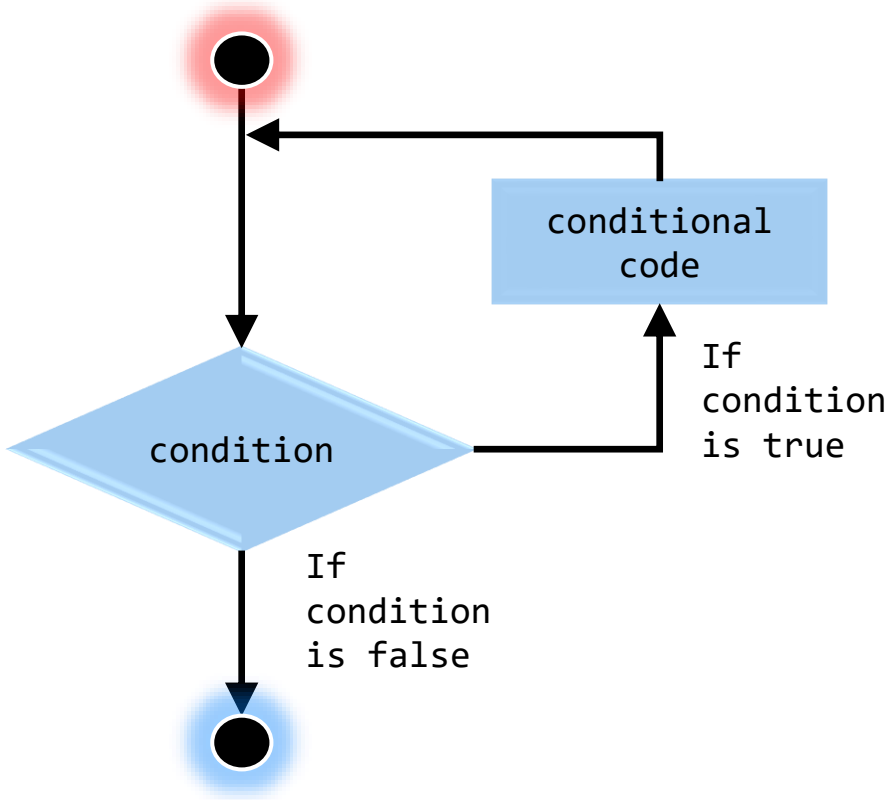
Outline

- Iterative Structures
- Loop Control Statements
- Operators

Iterative Structure

- In general, program statements are executed sequentially, however, a block of code may needs to be executed a number of times.
- Thus, programming languages provide various iterative control structures (called loops) that allow for more complicated execution paths.
- A loop repeatedly executes statements while some loop-continuation condition remains true.
- The general form of a loop statement in most of the programming languages is shown in the next slide:

Iterative Structure



Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	It is more like a while statement, except that it tests the condition at the end of the loop body.
nested loops	A switch statement allows a variable to be tested for equality against a list of values.

While loop

- A while loop in C programming repeatedly executes a target statement as long as a given condition is true.
- The syntax of a while loop in C programming language is:

```
1.  while(condition)
2.  {
3.    conditional code(s);
4.  }
```

- Here, conditional code(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value.
- The loop iterates while the condition is true. When the condition becomes false, the program control passes to the line immediately following the loop.

While loop: Example

```c
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 10;
/* while loop execution */
while( a < 20 )
{
printf("value of a: %d\n", a);
a++;
}
return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

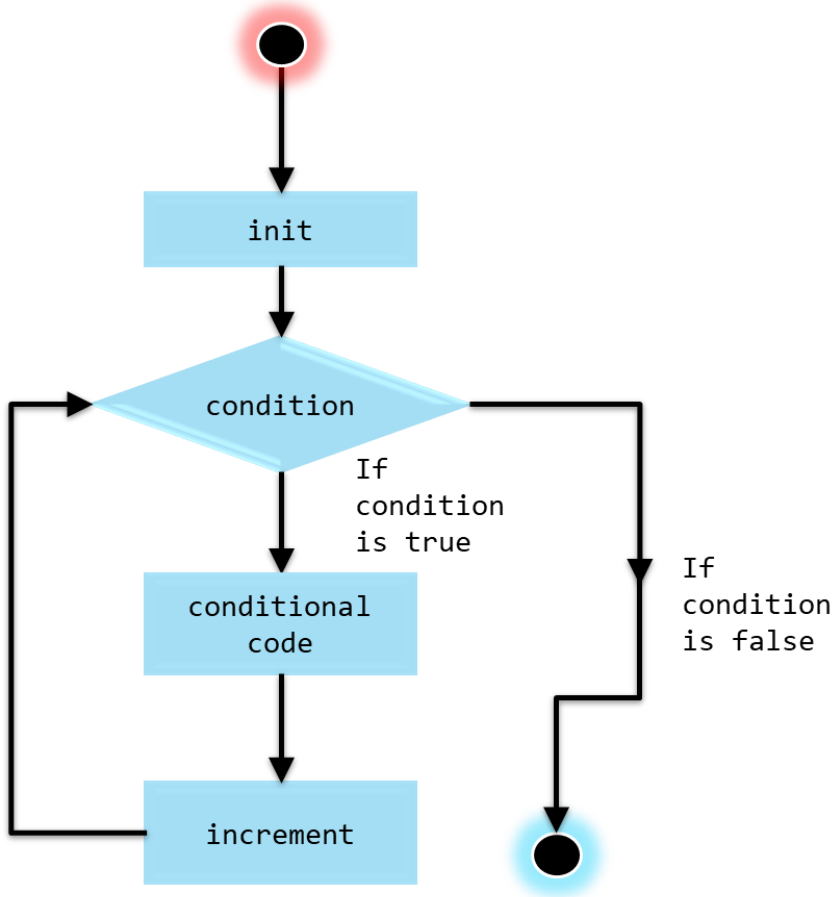
For loop

- A for loop is an iterative control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- The syntax for a for loop is given below:

```
1. for (init; condition; increment)
2. {
3.    conditional code(s);
4. }
```

For loop

- The init step is executed only once to declare and initialize any loop control variables.
- The condition is evaluated and the conditional code is executed if true, else the flow of control jumps to the next statement just after the for loop.
- After execution, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables.
- The condition is now evaluated again, and the process repeats itself (execute conditional code, increment step, and check condition again), terminating only if the condition becomes false.



For loop: Example

```c
#include <stdio.h>
int main ()
{
/* for loop execution */
for( int a = 10; a < 20; a = a + 1 )
{
printf("value of a: %d\n", a);
}
return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Do...while loop

- Unlike for loop and while loops, which test the loop condition at the top of the loop, the do...while loop in C programming checks its condition at the bottom of the loop.
- A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.
- The syntax of a do...while loop in C programming language is:

```
1.  do
2.  {
3.   Conditional code(s);
4.  }while( condition );
```

- Notice that the conditional code(s) executes at least once before the condition is tested.

Do...while loop: Example

```c
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 10;
/* do loop execution */
do
{
printf("value of a: %d\n", a);
a = a + 1;
}while( a < 20 );
return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Nested loops

- C programming allows the use of nested control structures. Hence, the following syntax describes the nested loop for a while loop, for loop, and do...while loop statement respectively.
- The syntax for each nested loop statement:

```
while(condition)
{
   while(condition)
   {
     conditional code(s);
   }
   conditional codes(s);
}
```

```
   for (init; condition; increment)
   {
       for (init; condition; increment)
       {
         conditional code(s);
       }
       conditional code(s);
   }
```

```
do
{
     conditional code(s);
     do
     {
       conditional code(s);
     }while( condition );
}while( condition );
```

Nested loops: Example

- Using a nested while loop create a 2d array of integers with n rows and m columns:

```c
#include <stdio.h>
int main()
{
    int rows, columns; // variable declaration
    int k=1, i=1; // variable initialization
    printf("Enter the number of rows :");  // input the number of rows.
    scanf("%d",&rows);
    printf("\nEnter the number of columns :"); // input the number of columns.
    scanf("%d",&columns);

    while(i<=rows) // outer loop
    {int j=1;
       while(j<=columns)  // inner loop
        {   printf("%d\t",k);  // printing the value of k.
            k++;   // increment counter
            j++;
        }
        i++;
        printf("\n");
    }
     return 0;
}
```

```
Enter the number of rows :3

Enter the number of columns :4
1       2       3       4
5       6       7       8
9       10      11      12
```

Loop Exercise

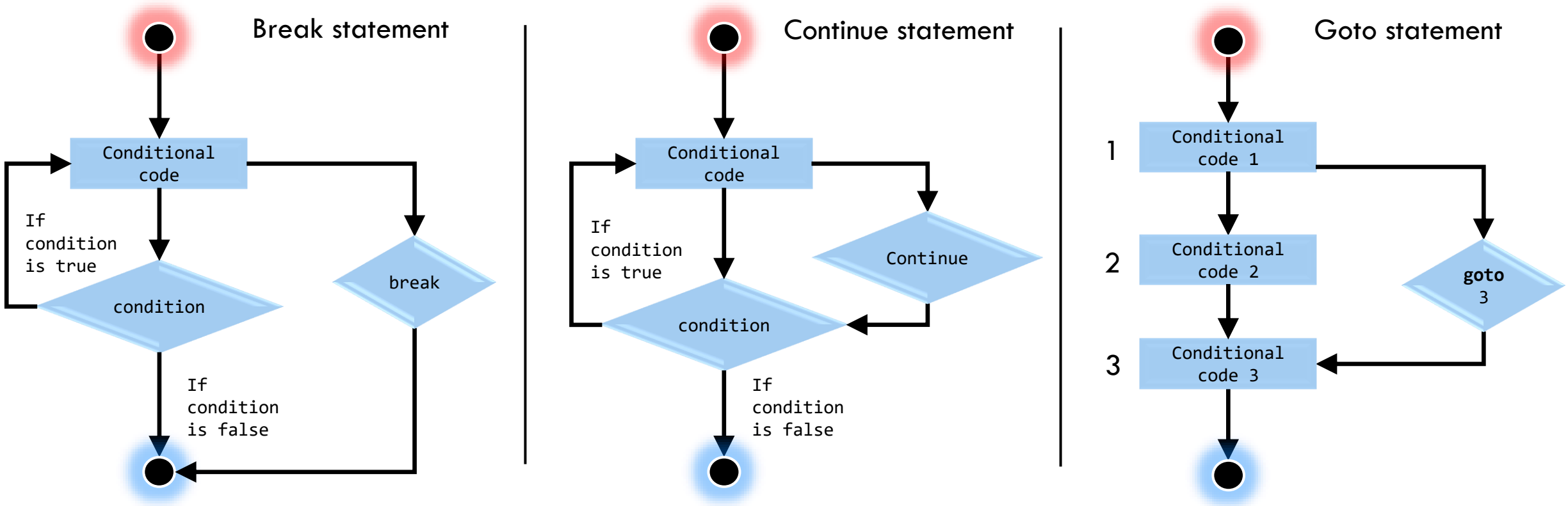
1. Write a C program to print all natural numbers from 1 to n using loop and another C program to print the numbers in reverse from n to 1using for loop.
2. Write a C program to print alphabets from a to z using for loop
3. Write a C program to input a number from user and print multiplication table of the given number.
4. Write a C program to input number from user and find sum of all even numbers between 1 to n using a for loop.
5. Using for loops, write a C program to find factorial of a number.
6. Write a C program using nested loops to print a multiplication table from 1 to 5
7. Write a C program to print all even numbers from 1 to 100.

Loop Control Statement

- Loop control statements change execution from its normal sequence. When execution sequence changes/leaves a scope, all automatic objects that were created within that scope are destroyed.
- C supports the following control statements.

Control Statement	Description
Break statement	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
Continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfers control to the labeled statement.

- *The flow diagram for the loop control statement:*



Break Statement

- The break statement in C programming has the following two usages:
  - When a break statement is encountered inside a loop, the loop is immediately **terminated** and the program control resumes at the next statement following the loop.
  - It can be used to terminate a case in the switch statement (This was covered last 2 weeks).
- If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- The syntax for a break statement in C is shown: `break;`

- The following code is an example of the break statement used as a terminator within a while loop.

```c
#include <stdio.h>
int main ()
{
   /* local variable definition */
   int a = 10;
   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
      if( a > 15)
      {
         /* terminate the loop using break statement */
         break;
      }
   }
return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Continue Statement

- The continue statement in C programming works somewhat like the break statement.
- Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
- For the for loop, continue statement causes the conditional test and increment portions of the loop to execute.
- For the while and do...while loops, continue statement causes the program control to pass to the conditional tests.
- The syntax for a continue statement in C is shown: `continue;`

- The following code is an example of how the continue statement is used exist an if statement.

```
1. #include <stdio.h>
2. int main ()
3. {
4.    /* local variable definition */
5.    int a = 10;
6.    /* do loop execution */
7.    do
8.    {
9.       if( a == 15)
10.      {
11.        /* skip the iteration */
12.        a = a + 1;
13.        continue;
14.      }
15.      printf("value of a: %d\n", a);
16.      a++;
17.   }while( a < 20 );
18.return 0;
19.}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Goto Statement

- A goto statement in C programming provides an unconditional jump from the 'goto' to a another line (label) in the conditional code statement, in the same function.
- However, the use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify.
- Any program that uses a goto can be rewritten to avoid them.
- The syntax for a continue statement in C is shown:

```
1.  Goto label;
2.  …
3.  …
4.  Label statement;
```

- The following code is an example of the goto statement used jump lines of code in a do...while loop.

```c
#include <stdio.h>
int main ()
{
   /* local variable definition */
   int a = 10;
   /* do loop execution */
   LOOP:do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }
      printf("value of a: %d\n", a);
      a++;
   }while( a < 20 );
return 0;
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Infinite Loop

- A loop is said to be infinite if the upholding condition is always true.
- A for loop with empty condition is considered a good example of an infinite loop.
- This is because none of the three expressions that form the 'for' loop are required, making it an endless loop (since the empty condition will always be true).
- However, note that you can terminate an infinite loop by pressing Ctrl + C keys.

```
1. #include <stdio.h>
2. int main ()
3. {
4.    for( ; ; )
5.    {
6.       printf("This loop will run forever.\n");
7.    }
8. return 0;
9. }
```

OPERATORS

1. An operator is a symbol that tells the compiler to perform specific mathematical or logical operation between operands.
2. An operand is quantity on which an operation is to done.
3. Example (given a=1, b = 2), C = a+b. Where a and b are operands, while '+' is the operator.
4. C language is rich in built-in operators, which can be classified into the following types:
    - Arithmetic Operators
    - Relational Operators
    - Logical Operators
    - Bitwise Operators
    - Assignment Operators
    - Special Operators

Arithmetic OPERATORS

Arithmetic Operator	Description	Example
+	Adds two operands (values)	a+ b
–	Subtract second operands from first	a – b
*	Multiply two operands	a*b
/	Divide numerator by the denominator	a/b
%	Returns the remainder of the division of two operands as the result	a % b
++	Increases integer value by one. Uses single operand	a++ or ++a
--	Decreases integer value by one. Uses single operand	--b or b--

Relational OPERATORS

Relational Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(a== b) false
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(a != b) true
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(a>b) false
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(a<b) true
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(a>= b) false
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(a<=b) true

Logical OPERATORS

Logical Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(a&& b) true
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(a || b) true
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(a&&b) false

Bitwise OPERATORS

Bitwise Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b)=12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(a | b)=61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(a^b)=40, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. (in complement of twos)	(~a)=-61, i.e., 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	(a<< b)=240, i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	(a>>b)=15, i.e., 0000 1111

Assignment OPERATORS

Assignment Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = a + b will assign the sum to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	b += a is equivalent to b = b + a
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	b -= a is equivalent to b = b – a
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	b *= a is equivalent to b = b * a
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	b /= a is equivalent to b = b / a
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	b %= a is equivalent to b = b % a

Assignment OPERATORS

Assignment Operator	Description	Example
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C= C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Special OPERATORS

Special Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
?:	Conditional Expression.	If Condition is true ? then value X : otherwise value Y
[ ] (operator)	used to access array elements using indexing	arr[index], where index represents the array index starting from zero