	TOPIC	С	CPP	JAVA	C#.NET	VB.NET
	8 bit (byte)	signed char unsigned char	signed char unsigned char	byte	byte sbyte	Byte SByte
	16 bit (short integer)	short unsigned short	short unsigned short	short char	short ushort	Short Ushort
INTEGERS	32 bit	long unsigned long	long unsigned long	int	int uint	Integer UInteger
Zi	Word size	int unsigned int	int unsigned int	N/A	N/A	N/A
	Arbitrarily precise (bignum)	N/A	N/A	java.math.BigInteger	System.Numerics.BigInteger (.NET 4.0)	System.Numerics.BigInteger (.NET 4.0)
Floating	Single Precision	float	float	float	float	Single
Floa	Double Precision	double	double	double	double	Double
	Character	char	char	char	char	Char
0	String	N/A	string	String	String	String
er type	Boolean	bool	bool	boolean	bool	Boolean
Other variable type	Enumeration	<pre>enum name {item1, item2,};</pre>	<pre>enum name {item1, item2,};</pre>	<pre>enum name {item1, item2,}</pre>	<pre>enum name {item1, item2}</pre>	Enum name Item1 End Enum
	object Universal	void *	void *	Object	object	Object
Other type	Records Structure	<pre>struct name {type name;};</pre>	N/A	N/A	<pre>struct name {type name;};</pre>	Structure name  Dim a As type End Structure
Othe	Unions	<pre>union name {type name;};</pre>			N/A	N/A

quoted text is optional April 2010 VI.0 Copyright© 2010 ISDNTC All rights are Reserved. Author: Kamini Chaudhari, Harshal Chaudhari, Sachin Sonawane.

Sponsored by





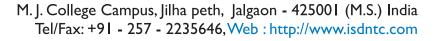














variable  constant Type synonym	<pre>Type_name name &lt;&lt; = initial_value &gt;&gt;; Enum{name = value };</pre>	<pre>Type_name name &lt;&lt; = initial_value &gt;&gt;;</pre>	Type_name name	Type name name	Dim name As Type
	<pre>Enum{name = value }:</pre>		<< = initial_value >>;	<pre>&lt;&lt; = initial_value &gt;&gt;;</pre>	<pre>&lt;&lt; = initial_value &gt;&gt;</pre>
	typedef type synonym;	<pre>const Type_name name= value; typedef type synonym;</pre>	<pre>final Type_name name= value; N/A</pre>	<pre>const Type_name name= value; using synonym = type;</pre>	Const name As Type = value Imports synonym = type
if	<pre>if(condition) {instruction;} &lt;&lt; else {instruction}&gt;&gt;</pre>	<pre>if(condition){instruction;} &lt;&lt; else {instruction;}&gt;&gt;</pre>	<pre>if(condition) {instruction;} &lt;&lt; else {instruction;}&gt;&gt;</pre>	<pre>if(condition){instruction;} &lt;&lt; else {instruction;}&gt;&gt;</pre>	<pre>If condition Then`   instruction &lt;&lt; Else   instruction &gt;&gt; End If</pre>
else if	<pre>if( condition ) {instruction;} else if( condition ) {instruction;} &lt;&lt; else {instruction;} &gt;&gt;</pre>	<pre>if( condition ) {instruction;} else if( condition ) {instruction;} &lt;&lt; else {instruction;} &gt;&gt;</pre>	<pre>if( condition ) {instruction;} else if( condition ) {instruction;} &lt;&lt; else {instruction;} &gt;&gt;</pre>	<pre>if( condition ) {instruction;} else if( condition ) {instruction;} &lt;&lt; else {instruction;} &gt;&gt;</pre>	If condition Then instruction ElseIf condition Then << Else Instruction >> End If
Select case	<pre>switch (variable) {   case case1: instruction;   &lt;<bre>  &lt;&lt; default: instruction;</bre></pre>	<pre>switch (variable) {   case case1: instruction;   &lt;<bre>  &lt;&lt; default: instruction;</bre></pre>	<pre>switch (variable) {   case case1: instruction;   &lt;<bre>  &lt;&lt; default: instruction;</bre></pre>	<pre>switch (variable) {   case case1: instruction;   &lt;<bre>  &lt;&lt; default: instruction;</bre></pre>	Select Case variable  Case case1  instruction  Case Else  instruction >>  End Select
Conditional Expression	<pre>if (Condition ? Value_True</pre>	<pre>if (Condition ? Value_True</pre>	<pre>if (Condition ? Value_True</pre>	<pre>if (Condition ? Value_True</pre>	<pre>iff (Condition , Value_True</pre>
while	<pre>while( condition ) {     instruction; }</pre>	<pre>while( condition ) {     instruction; }</pre>	<pre>while( condition ) {     instruction; }</pre>	<pre>while( condition ) {      instruction; }</pre>	do until condition     instructions Loop or While condition     instructions End While
	Select case  Conditional Expression	<pre>else if     if( condition )     {instruction;}     else if( condition )     {instruction;}     &lt;&lt; else     {instruction;} &gt;&gt;  Select case     switch (variable)     {         case case1: instruction;         &lt;                         </pre> Conditional Expression if (Condition ? Value_True	<pre>else if     if( condition )         {instruction;}         else if( condition )         {instruction;}         else if( condition )         {instruction;}         &lt;&lt; else</pre>	else if	else if

Sponsored by

## Khandesh College Education Society's

















	TOPIC	С	СРР	JAVA	C#.NET	VB.NET
	Do while	<pre>do {     instruction; } while(condition);</pre>	<pre>do {     instruction; } while(condition);</pre>	<pre>do {     instruction; } while( condition );</pre>	<pre>do {     instruction; } while(condition);</pre>	do     instructions Loop While condition or do     instructions Loop Until Condition
	for	<pre>for(&lt;<type>&gt;i=first; i<last; i++)="" instruction;="" pre="" {="" }<=""></last;></type></pre>	<pre>for(&lt;<type>&gt;i=first; i<last; i++)="" instruction;="" pre="" {="" }<=""></last;></type></pre>	<pre>for (&lt;<type>&gt;i=first; i<last; i++)="" instruction;="" pre="" {="" }<=""></last;></type></pre>	<pre>for(&lt;<type>&gt;i=first; i<last; i++)="" instruction;="" pre="" {="" }<=""></last;></type></pre>	<pre>For i =first to last &lt;<step 1="">&gt;     instruction Next I</step></pre>
Declaration	foreach	N/A	<pre>for_each(start, end, function)</pre>	<pre>for(type item:set) {     instruction; }</pre>	<pre>foreach (type item in set) {     instruction; }</pre>	for each item as Type in set instruction Next item
_	throw		throw exception;	throw exception;	throw exception;	throw exception
	handler		<pre>try {instructions } catch «(exception)» {instructions } &lt;&lt; finally</pre>	<pre>try {instructions } catch &lt;&lt;(exception)&gt;&gt; {instructions } &lt;&lt; finally      {instructions }&gt;&gt;</pre>	<pre>try {instructions } catch &lt;&lt;(exception)&gt;&gt; {instructions } &lt;&lt; finally      {instructions }&gt;&gt;</pre>	try instruction Catch ex As Exception instruction Finally instruction End Try
	assertion	<pre>assert( condition);</pre>	assert( condition);	<pre>assert( condition);</pre>	System.Diagnostics. Debug.Assert(Condition);	System.Diagnostics. Debug.Assert(Condition);
	Exit block	break;	break;	break;	break;	Exit block
Control flow	continue	continue;	continue;	continue;	continue;	continue block
	Label	label:	label:	label:	label:	label:
	GOTO	<pre>goto label;</pre>	<pre>goto label;</pre>	goto label;	<pre>goto label;</pre>	Goto label

Sponsored by



















	TOPIC	С	СРР	JAVA	C#.NET	VB.NET
	Calling function	foo(< <parameters>&gt;);</parameters>	<pre>foo(&lt;<parameters>&gt;);</parameters></pre>	foo(< <parameters>&gt;);</parameters>	foo(< <parameters>&gt;);</parameters>	foo(< <parameters>&gt;)</parameters>
_	Basic / Void Function	<pre>void foo(&lt;<parameters>&gt;) {instructions;}</parameters></pre>	<pre>void foo(&lt;<parameters>&gt;) {instructions;}</parameters></pre>	<pre>void foo(&lt;<pre>caparameters&gt;&gt;) {instructions;}</pre></pre>	<pre>void foo(&lt;<pre>caparameters&gt;&gt;) {instructions;}</pre></pre>	<pre>sub foo(&lt;<parameters>&gt;)     instructions End sub</parameters></pre>
function	Value returning function	<pre>type foo(&lt;<parameters>&gt;) {instructions;   return value; }</parameters></pre>	<pre>type foo(&lt;<parameters>&gt;) {instructions;   return value; }</parameters></pre>	<pre>type foo(&lt;<parameters>&gt;) {instructions;   return value; }</parameters></pre>	<pre>type foo(&lt;<parameters>&gt;) {instructions;   return value; }</parameters></pre>	Function foo(< <parameters>&gt;) As type instructions return value End Function</parameters>
	Required Main Function	<pre>&lt;<global declarations="">&gt; int main(&lt;<int *argv[]="" argc,char="">&gt;) {instructions;}</int></global></pre>	<pre>&lt;<global declarations="">&gt; int main(&lt;<int *argv[]="" argc,char="">&gt;) {instructions;}</int></global></pre>	<pre>public static void main(string   []args) {instructions;}</pre>	<pre>static void main(&lt;<string []args="" []args)="" int="" main(<<string="" or="" static="" {instructions;}="">&gt;) {instructions;}</string></pre>	<pre>sub main(&lt;<byval as="" cmdarg()="" string="">&gt;)     instructions End sub or Function main(&lt;<byval as="" cmdarg()="" string="">&gt;)     instructions End Function</byval></byval></pre>
	String to integer	<pre>Integer = atoi (string);</pre>	<pre>istringstream(string) &gt;&gt; number;</pre>	<pre>Integer=Integer.parseInt(string );</pre>	<pre>int = int.Parse(string);</pre>	<pre>Integer = CInt(string)</pre>
	String to long integer	<pre>Integer = atol (string);</pre>	<pre>istringstream(string) &gt;&gt;</pre>	<pre>long=Long.parseLong(string);</pre>	<pre>long = long.Parse(string);</pre>	long = CLng(string)
ersion	String to floating point	<pre>Integer = atof (string);</pre>	<pre>istringstream(string) &gt;&gt; number;</pre>	<pre>float=Float.parseFloat(string); double=Double.parseDouble(string);</pre>	<pre>float = float.Parse (string) double = double.Parse(ptring)</pre>	<pre>single = CSng(string) double = CDbl(string</pre>
Type conversion	Integer to string	<pre>sprintf(string,"%i",integer);</pre>	<pre>ostringstream o; o &lt;&lt; number; string = o.str();</pre>	String=Integer.toString(integer);	String=number.toString();	String = CStr (number)
	Floating point to string	<pre>sprintf(string,"%f",float);</pre>	<pre>ostringstream o; o &lt;&lt; number; string = o.str();</pre>	<pre>String=Float.toString(float); String=Double.toString(double);</pre>	String=number.toString();	String = CStr (number)

Sponsored by

## Khandesh College Education Society's



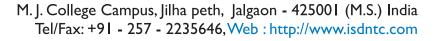














	TOPIC	С	СРР	JAVA	C#.NET	VB.NET
Standard input output	Read from	<pre>scanf(format, &amp;x); fscanf(stdin, format, &amp;x);</pre>	<pre>cin &gt;&gt; x; getline(«std::»cin, str);</pre>	<pre>x = System.in.read(); or x = new Scanner(System.in).nextInt(); or x = new Scanner(System.in).nextLine();</pre>	<pre>x = Console.Read(); or x = Console.ReadLine();</pre>	<pre>x = Console.Read() or x = Console.ReadLine()</pre>
	Write stdout	<pre>printf(format, x); fprintf(stdin, format, &amp;x);</pre>	cout << x;	<pre>System.out.print(x); System.out.printf(format,x); System.out.println(x);</pre>	<pre>Console.Write(&lt;<format,>&gt;x); or Console.WriteLine(&lt;<format,>&gt;x);</format,></format,></pre>	<pre>Console.Write(&lt;<format,>&gt;x) or Console.WriteLine(&lt;<format,>&gt;x)</format,></format,></pre>
	Write Error	<pre>fprintf(stderr,format,x);</pre>	cerr << x; clog << x;	<pre>System.err.print(x); System.err.printf(format,x); or System.err.println(x);</pre>	<pre>Console.Error.Write(&lt;<format,>&gt;x); or Console.Error.WriteLine(&lt;<format,>&gt;x);</format,></format,></pre>	Console.Error.Write(< <format,>&gt;x)  Or  Console.Error.WriteLine(&lt;<format,>&gt;x)</format,></format,>
line	Argument Values	argv[n]	argv[n]	argv[n]	argv[n]	cmdArgs(n)
land	Argument count	argc	argc	args.Length	args.Length	cmdArgs.Length
Commandline	Program Name	first Argument	first Argument	first Argument	Assembly.GetEntryAssembly(). Location;	[Assembly].GetEntryAssembly(). Location
Execution of commands	Shell Command	System("Command");	System("Command");	Runtime.exec(command); or new ProcessBuilder(command). start();		Microsoft.VisualBasic.In teraction.Shell(command <<, WindowStyle>> <<, isWaitOnReturn>>)
	Execute Program	execl(path,args); execv(path,arglist);	execl(path,args); execv(path,arglist);		System.Diagnostics.Process.Star t(path, argstring);	System.Diagnostics.Process.Star t(path, argstring)

Sponsored by















