CHRIST THE KING BOYS MATRIC HR. SEC. SCHOOL, KUMBAKONAM – 612 001

LIST OF SYNTAXES

NAME	SYNTAXES
Variable Declaration	datatype var1,var2,….var-n; int a,b;

Reference Variable	datatype &reference variable =original variable;
	int a=10;
	int &b=a;

setw	setw(number of characters)
	int a=2,b=3;
	cout<<setw(4)<<a<<setw(10)<<b;

setprecision	setprecision(number of digits)
	float sum=1200.123;
	cout<<setprecision(5)<<sum;

Explicit Type Conversion	typename expression;
If Statement	if(expression/condition) True block; Statement – x;

If – Else Statement	if(expression/condition)
	{
	  True block;
	}
	else
	{
	  False block;
	}

Nested If Statement – If nested inside if part	if (expression – 1)
	{
	  if (expression – 2)
	  {
	  True_Part_Statements;
	  }
	  else
	  {
	  False_Part_Statements;
	  }

	}
	else
	{
	  body of else part;
	}

Nested If Statement – If nested inside else part	if (expression-1)
	{
	body of true part;
	}
	else
	{
	  if (expression)
	  {
	  True_Part_Statements;
	  }
	  else
	  {
	  False_Part_Statements;
	  }
	}

If Nested inside both if and else part	if (expression)
	{
	  if (expression)
	  {
	  True_Part_Statements;
	  }
	  else
	  {
	  False_Part_Statements;
	  }
	}
	else
	{
	  if (expression)
	  {

	True_Part_Statements;
	}
	else
	{
	False_Part_Statements;
	}
	}

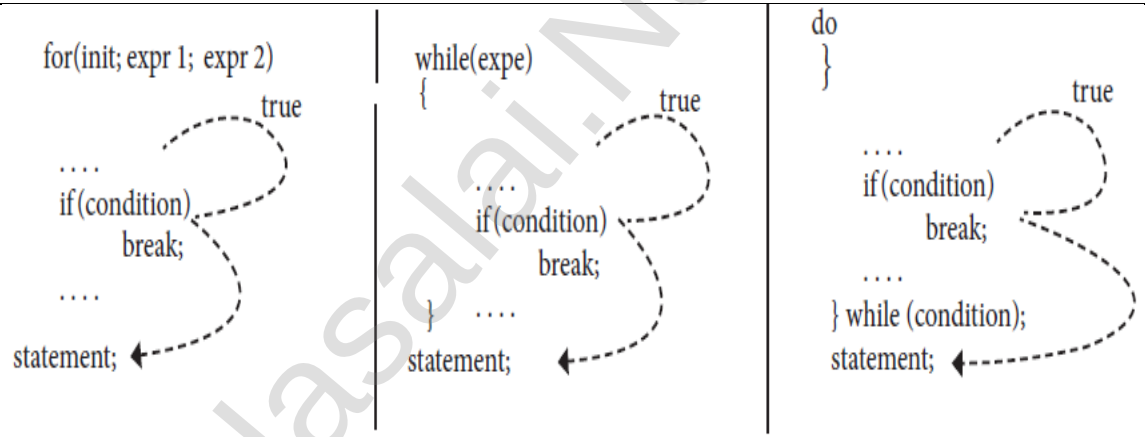
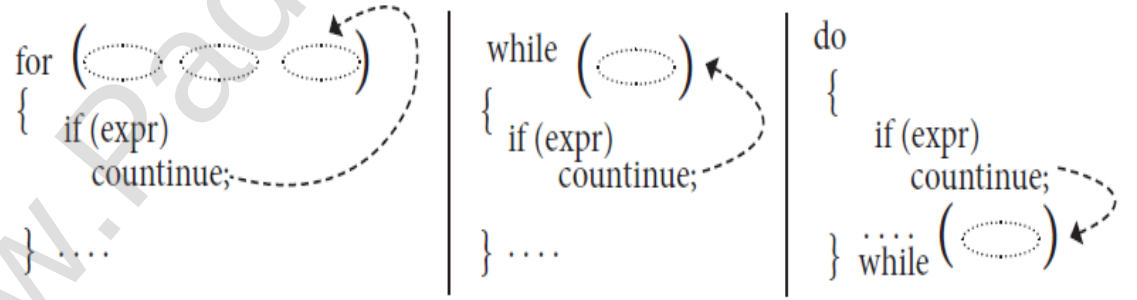
Switch Statement	switch (expression)
	{
	case constant1:
	statement – 1;
	break;
	case constant2:
	statement – 2;
	break;
	case constant3:
	statement – 3;
	break;
	case constant4:
	statement – 4;
	break;
	default:
	statement – x;
	}

For Loop	for (initializations; test expression ; update expression)
	{
	body of the Loop;
	}

While Loop	while (Test Expression)
	{
	body of the Loop;
	}
	Statement – X;

Do – While Loop	do {

	body of the Loop;
	}
	while (Test Expression);

Goto Statement	goto label; ---------- ---------- label:
Break Statement	
Continue Statement	

Nested Loop	for (initialization(s); test-expression; update expression(s))
	{
	for (initialization(s); test-expression; update expression(s)
	{
	statement(s);
	}
	statement(s);

	}

isalnum()	isalnum (char c)
	int p = isalnum('5'); //the value of p is non – zero value.
	int r = isalnum('$'); //the value of r is zero.

isalpha()	isalpha(char c);
	int n = isalpha('4'); //the value of n is zero.
	int m = isalpha('a'); // the value of m is non – zero value.

isdigit()	isdigit(char c);
	int n = isdigit('4'); //the value of n is non – zero value.
	int m = isdigit('a'); // the value of m is zero.

islower()	islower(char c);
	int n = islower('A'); //the value of n is zero.
	int r = islower('a'); //the value of r is non – zero value.

isupper()	isupper(char c);
	int n = isupper('a'); //the value of n is zero.
	int r = isupper('A'); //the value of r is non – zero value.

toupper()	toupper(char c); char c = toupper('a'); //the character A is assigned to c
tolower()	tolower(char c); char c = tolower('B'); //the character b is assigned to c
strcpy()	strcpy(targetstring,sourcestring);
strcmp()	strcmp(string1,string2);
strcat()	strcat(targetstring,sourcestring);
strupr()	strupr(str);
strlwr()	strlwr(str);
cos()	cos(x);
sin()	sin(x);
sqrt()	sqrt(x);
pow()	pow(base,exponent);
Constant Arguments	return_type function_name (const data_type variable=value)
Inline Function	inline returntype functionname(datatype parameter1, … datatype parameter-n)
Return Statement	return expression/variable;
Declaration of One – Dimensional Array	data_type array_name [array_size];
Initialization of One – Dimensional Array	data_type array_name [size] = {value-1,value-2,…… ,value-n};
Declaration of One – Dimensional char Array	char array_name[size];

Initialization of One – Dimensional char Array	char array_name[size]={list of characters separated by comma or a string};
Declaration of Two – Dimensional Array	data_type array_name[row-size][col-size];
Declaration of Structure	struct structure_name { type member_name1; type member_name2; } reference_name;

Declaration of Class	class class-name
	{
	private:
	data member declaration;
	member function declaration;
	protected:
	data member declaration;
	member function declaration;
	public:
	data member declaration;
	member function declaration;
	};

Outline Member Function	return_type class_name :: function_name (parameter list)
	{
	function definition
	}

Accessing Class Members	Object_name . function_name(actual parameter);

Operator Overloading Function	returntype classname :: operator operator Symbol (argument list)
	{
	//Function body
	}

Defining the Derived Class	class derived_class_name : visibility_mode base_class_name
	{
	// members of derivedclass
	};