vinodsrivastava.wordpress.com

Question Bank Term-1

Q1 Write Header File for following Function/object

Function/object presents in header file	Header file required
cin, cout, endl,	iostream.h,
gets(), puts(),	stdio.h
M_PI, abs(), labs(), fabs(), cabs(), sqrt(), pow(), pow10(), exp(), log(), log10(), sin(), cos(), tan(), fmod()	math.h
toupper(), tolower(), isupper(), islower(), isalpha(), isdigit(), isalnum()	ctype.h
strcpy(), strlen(), strupr(), strlwr(), strrev(), strcat(), strcmp(), stricmp(), strcmpi()	string.h
getch()	conio.h

Q2 What is Variables ?

A variable is name given to a memory location to store value in the computer's main storage. The value assigned to the variable name may change (vary) as the program is executed.

Q 3 Write four Rules for naming a C++ variable (identifier)

1. Variable name should start with an alphabet (letter) or an underscore.
2. Variable name may contain more than one character. Second characters onwards we may use only alphabets or digit or underscore.
3. No special characters are allowed in a variable name except underscore.
4. A variable name in C++ is case sensitive. Uppercase and lowercase letters are distinct. A variable name cannot be a keyword.
5. A variable name cannot be a keyword.

Q4. Identify three incorrect identifier names and explain why, from the list given below:

long, AD_No, INT, comp-sc, CAL29, 2ndfloor, price, cell#

Ans:

1. long – It is a keyword
2. comp-sc – It uses a special character other than underscore (-)
3. cell# - It uses a special character other than underscore (#)

Q 5. Give the memory allocation for following Data Type char, int, float, double

Date Type	Storage (Memory Allocation)	Range of values
1. **char**	1 byte or 8 bits	-128 to 127
2. **int**	4 bytes or 32 bits	-2147483648 to 2147483647
3. **float**	4 bytes or 32 bits	$3.4*10^{-38}$ to $3.4*10^{38}$
4. **double**	8 bytes or 64 bits	$1.7*10^{-308}$ to $1.7*10^{308}$

Q6. What is Type Casting explain different way to do typecasting in C++

Typecasting: converting data from one type to another type temporarily, inside the processor (CPU). Examples of Type casting are given below:

```
#include<iostream.h>
void main()
{
    int m, n;
    cout<<"Input 2 integers? ";
    cin>>m>>n;
    double r1=double(m)/n;
    double r2=(double)m/n;
    cout<<r1<<','<<r2<<endl;
}
```

Q7. What are type Modifer. Name type modifier in C++ also identify data type(s) which support these modifier or does not support any

Type Modifier : Type modifiers are used to change default type of the built-in data types. Type modifiers supported by C++ are **long**, **short**, **signed** and **unsigned**.

- Data type **void** and **float** does not support any type modifiers.
- Data type **int** supports all the four type modifiers.
- Data type **char** supports **signed** and **unsigned**.
- Data type **double** supports **long**.
- Data type is assumed to be **int**, if only type modifiers are used to create a variable.

Q8. Define Token? Give some example of Token

Token : Building block of a program is called a token. It is also called program element. Tokens of a C++ program can be classified as Keyword, Identifier, Constant, Operator, String and Comment.

Q9. What are identifier how they are different from Keyword

Keyword:	**Built-in Identifier**
No header File Required	Built-in identifier we **need appropriate header** file.
Cannot be redefined	Can be redefined

Q10. What are operator? What are different type of operators. Give example(s) of each

Operator: Operators are used in C++ to carry out various functions. An operator in C++ can be **unary**, **binary** and **ternary**.

Unary operator: An operator that needs **one operand**. **Examples:** Unary -, unary +, ++, -- and !. There are more unary operators, but they will be discussed later.

Binary operator: An operator that needs **two operands**. **Example**: Binary +, Binary -, *, /, %, C++ short hand operators, logical operators, && and ||. More binary operators will be discussed later.

Ternary operator: An operator that needs **three operands**. Ternary operator is also known as Conditional operator. Example **? :**

Q11 What are increment & decrement Operator explain with example how it can be used

Increment Operator: Increment operator (++) increments value stored in a variable by 1 (One). Increment operator works with character (**char**) type data, integer (**int**) type data and floating point (**float** and **double**) type data. Examples of Increment operators are given below:

Operator	C++ Statement	Output	Explanation
Pre ++	cout<<++x<<endl; cout<<x<<endl;	7 7	Increments x and then displays x Displays incremented values stored in x
post ++	cout<<x++<<endl; cout<<x<<endl;	6 7	Displays x and then increments x Displays incremented values stored in x

Decrement Operator: Decrement operator (--) decrements value stored in a variable by 1 (One). Decrement operator works with character (**char**) type data, integer (**int**) type data and floating point (**float** and **double**) type data. Examples of Decrement operators are given below:

Operator	C++ Statement	Output	Explanation
--	cout<<--z<<endl; cout<<z<<endl;	25 25	Decrements z and then displays z Displays decremented values stored in z
--	cout<<z--<<endl; cout<<z<<endl;	26 25	Displays z and then decrements z Displays decremented values stored in z

Note : Cascading of increment/decrement operator with << in single cout statement evaluation will start from right hand side to left hand side and display will be from left hand to right hand side

Q12 Give the Output of the code

```
void main()
{
int a=35;
cout << ++a << ',' << a++ << ',' << a++ << endl;
cout << a-- << ',' << --a << ',' << --a << endl;
getch();
}
```

Ans: 38,36,35
36,36,37

Q13 Explain Ternary Operator (Conditional Operator) with example

Ternary operator is used in place of **if-else** statement. But all **if-else** statement cannot be replaced by Ternary operator. It is called ternary operator since an expression involving ternary operator requires three (3) operands and two (2) operators. The two Ternary operator is more compact compared to **if-else** statement.

Rule: Condition? Action1: Action2

```
int x=10,y=5;
 (x>y)?cout<<"x is greter":cout<<"y is greater" ;  // OUTPUT: x is greater
```

Note Question asking to write logical expression will not contain if –else it will be purely logical expression

Q14. Write C++ logical expression (do not use C++ built-in functions):

i) To check that a character variable mychar contains only alpabets
ii) To check that an integer variable number is even no not divisible by 4
iii) To check that an integer variable marks contains a value between 300 and 500

Ans: i) mychar >= 'A' && mychar <= 'Z'
ii) number%2 == 0 && number%4 != 0
iii) marks >= 300 && marks <= 500

Note do not use if statement otherwise it will treated wrong

Q15. Give output

```
int value=10>5 && 10<5 || 10==5;
cout<<value<<endl;
```

Ans **0**

Q16. What are comment ? explain two types of comment with example

Comment: Non executable statements of a C++ program are called Comments. Comments are also known as Remarks. A Comment is completely ignored by a compiler. C++ supports two types of Comments: Single Line Comment and Multi-Line Comment.

Single line Comment:Single Line Comment starts with pair of forward slash (//) and till the end of line is considered as a Comment. Examples of Single Line Comment are given below:

```
// single line comment
// in C++ style
```

Multi-line comment: Multi-line comment start with forward slash and star (/*) and with star and forward slash (*/). Examples of Multi-Line Comment are given below:

```
/*
    multi-line comments
    comment in C style    */
```

Q17. What are compiler directive/ Pre-processor give two examples

Compiler directive: instruction given to the compiler. Compiler directive is also called Pre-processor. C++ statement is an instruction given to CPU or to the computer. It is called Pre-Processor because instruction to the compiler given before the processing starts. Every Compiler Directive begins with hash (#). Examples of Compiler Directives are : **#include:** is used to include header files
#define: is used to create C++ macros

Q18 Differentiate between Run time & Logical Error with example

Run time error	Logical error
Syntactically correct statement performs illegal operation during execution of a program is called Run-Time errors.	An error in program design or program implementation that does not prevent your program from compiling, but causes it to do something unexpected.
Example: Division by zero (0), Square root of a negative number.	Example: Variables with incorrect or unexpected values. Incorrect formulae Incorrect use of operators

Q19 Differentiate between Syntax & Run time

Syntax error	Run time error
Error committed when the grammar of the language is violated.	Syntactically correct statement performs illegal operation during execution of a program when the program encounters unexpected data is called Run-Time errors.
Syntax errors are detected at compile time.	Run time errors are detected at run time.
Example: Typographical mistakes like missing semicolon Use of undeclared variable	Example: Division by zero (0) Square root of a negative number

Q20 Mention two differences between data type float and data type double.

float Data Type	double Type
float uses 4 bytes	double uses 8 bytes.
float does not support any type modifier	double supports 'long' type modifier.

Q21 Explain with example difference between Entry Level & Exit Level Loop

Entry controlled Loop	Exit Controlled Loop
Looping condition is checked at the beginning of the loop.	Looping condition is checked at the end of the loop.
Loop does not execute even once if the looping condition is false in the beginning itself.	Loop executes at least once irrespective of the looping condition.
Example: **for** (**int** x=2; x>5; x++) {cout<<x<<", "; }	Example: **int x=2;** **do** **{ cout<<x<<", "; }** **While (x>5);**

Q22. Write a complete C++ program to input name of a student (string), theory marks (out of 70), practical marks (out of 30) and weekly test marks (out of 40); calculate term total (theory + practical) and grand total (80% of term total + 50% of weekly test). Display name, theory marks, practical marks, weekly test marks, term total and grand total on the screen.

```
#include<iostream.h>
void main()
{
    char name[20];
    double theo, prac, wt;
    cout<<"Student Name? ";
    cin>>name;            //Input name without space
    cout<<"Theory marks[0-70]? ";  cin>>theo;
    cout<<"Practical marks[0-30]? "; cin>>prac;
    cout<<"Weekly Test marks[0-40]? "; cin>>wt;
    double term=theo+prac;
    double grand=0.8*term+0.5*wt;
    cout<<"Name       ="<<name<<endl;
    cout<<"Theory     ="<<theo<<endl;
    cout<<"Practical  ="<<prac<<endl;
    cout<<"Term Total ="<<term<<endl;
    cout<<"Weekly Test="<<wt<<endl;
    cout<<"Grand Total="<<grand<<endl;
}
```

Note : attention should be given for proper data type variable declaration

Q23. Write a complete C++ program to input employee name (string), basic salary; calculate house rent (40% of basic salary), dearness allowance (65% of basic salary), city allowance (15% of basic salary), gross salary (basic salary + house rent + dearness allowance + city allowance), provident fund deductions (10% of gross salary) and net salary (gross salary - provident fund deductions). Display basic salary, house rent, dearness allowance, city allowance, gross salary, provident fund deductions and net salary on the screen.

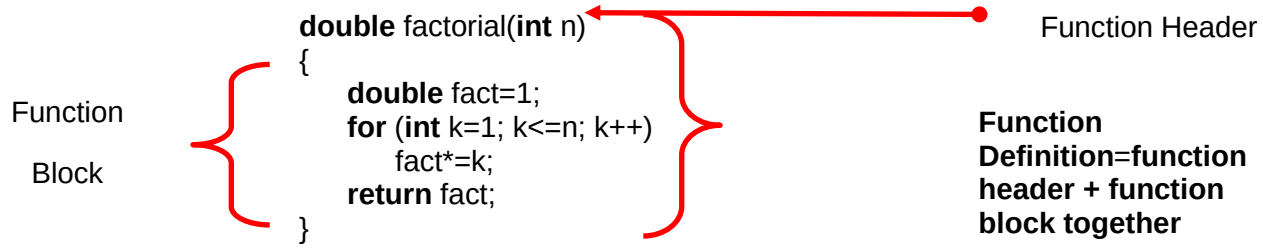
```
#include<iostream.h>
void main()
{
    char name[20];
    double basic;
    cout<<"Employee Name? ";
    cin>>name;              //Input name without space
    cout<<"Basic Salary? ";
    cin>>basic;
    double hrent=0.4*basic;
    double dallow=0.65*basic;
    double callow=0.15*basic;
    double gross=basic+hrent+dallow+callow;
    double pfund=0.1*gross;
    double net=gross-pfund;
    cout<<"Name              ="<<name<<endl;
    cout<<"Basic Salary      ="<<basic<<endl;
    cout<<"House Rent        ="<<hrent<<endl;
    cout<<"Dearness Allowance ="<<dallow<<endl;
    cout<<"City Allowance    ="<<callow<<endl;
    cout<<"Gross Salary      ="<<gross<<endl;
    cout<<"Provident Fund    ="<<pfund<<endl;
    cout<<"Net Salary        ="<<net<<endl;
}
```

***Note** Using **int** as datatype for a variable like **hrent /dallow** where value is decimal will leads to logical error

Q24. Explain with Example different part of Function

a) **Function Header**: it contains the **name** of the function, **return value** of the function and **optional** list of formal parameters, that is, it is not necessary that every function must have parameters. Function header is also called **Function Declarator**.

b) **Function Body**: it is the block after the function header. Function block contains statement that carries out action inside the function including the optional **return** statement. If the return value of a user defined function is **void**, then return statement is not required. Function Body is also called **Function Block**. Function header along with function block defines a complete function. An example of a user defined function is given below:



a) Name of the function is factorial
b) Return value of the function is **double**
c) Function has a formal parameter **int** n
d) Block after the function header is the body of the function. Function header plus function block is the function definition.

Q25. When is return statement necessary in a C++ function? What is the role of return statement?

Return statement is necessary in a C++ function when the return type of the function is not void but some other data type.**Return statement serves 3 purpose**

a) It terminates the function
c) Returns a value to the calling function
b) Program control returns to calling function

Q26 Explain with example correct way to invoke a function with void return type & double return type

Comparing factorial() function with return value **double** and with return value **void**.

Return value of a function is double	**Return value of a function is void**
double factorial(**int** n) { **double** fact=1; **for** (**int** k=1; k<=n; k++) fact*=k; **return** fact; }	**void** factorial(**int** n) { **double** fact=1; **for** (**int** k=1; k<=n; k++) fact*=k; cout<<fact<<endl; }
Correct function invocation	**Correct function invocation**

Var=FunctName(ActualParam);
cout<<FunctName(ActualParam);
double f1=factorial(m);
cout<<factorial(m);
	FunctName(ActualParam);
	factorial(m);
	factorial(8);
	factorial(m+3);

Q27 What happen when return type of function is void and user have given return statement;

Ans Complier will flag error

Q28 What happen when return type of function is not void and user have not given return statement;

Ans Complier will compile the program but flag warning. At run time it will stop or halt the program

Q 29. What is Function prototype Explain with example

A function declaration contains Function name, Return value of the function, Data type of optional list of formal parameters and a Semi-colon at the end. Name of the formal parameters are not important in a function declaration. But if the formal parameter names are included in the function declaration, then they are ignored by the compiler. **Function Declaration** (**Function Prototype**) are **declared before main()** and its **function definition** is given after **main()**

```
#include<iostream.h>
double factorial(int);
void main()
{  int m;
     cout<<"Input an integer? ";
     cin>>m;
     double f1=factorial(m);
     cout<<m<<"!="<<f1<<endl;  }
double factorial(int n)
{  double fact=1;
     for (int k=1; k<=n; k++)
          fact*=k;
     return fact; }
```

Blue highlighted line is the **Function Declaration** or **Function Prototype**. Pink highlighted line is the **Function Invocation**. When the compiler encounters the function declaration, it knows that somewhere in the block of the main() function, it will come across a function invocation which will match function declaration but the function definition will be after the main() function. This is another common practice to first declare the function and then define the function after the main() function. In this, function's declaration is separated from its definition and when coding large program this type of methodology is followed.

Q30. Difference between Function prototype & Function Definition with example

Function Declaration/Prototype	Function Definition
A function declaration contains Function name, Return type of the function, optional list of formal parameters, and a Semi-colon at the end.	A function definition is the complete function, that is, header and the body.
Name of the formal parameters are not compulsory in a function declaration.	Name of the formal parameters are compulsory in a function definition.

Example:

```
#include<iostream.h>
double factorial(int); //function declaration
void main()
{
     int m;
     cout<<"Input an integer? ";
     cin>>m;
     double f=factorial(m);
     cout<<m<<"!="<<f<<endl;
}
double factorial(int n)
{
     double fact=1;
     for (int k=1; k<=n; k++)
          fact*=k;
     return fact;
}
```

Function definition

Q31. Difference between Actual Parameter and Formal Parameter with proper example

Actual Parameter	Formal Parameter
• Parameter used in function invocation	• Parameter used in function definition
• Actual parameter may be a variable or an expression or a constant	• Formal parameter is always variable (or an alias)

Example:

```
#include <iostream.h>

 void first(int b)  //b is the formal parameter
{        b++;   cout<<b;   }

void main()
{    int a = 10;
     first(a);    // a is the actual parameter
     first(a+10); //a+10 is the actual parameter
}
```

Q32. Difference between Value Parameter and Reference Parameter

Value Parameter	Reference Parameter
• Copy of actual parameter	• Alias of actual parameter
• Change in value parameter does not change actual parameter	• Change in reference parameter, updates actual parameter
• Transfer of data is one way, from calling function to called function	• Transfer of data is two ways, from calling function to called function and vice-versa
• Actual parameter may either be a variable or an expression or a constant	• Actual parameter can only be a variable, it cannot be constant or expression

Q33. Difference between Local & Global Variable with example

Local Variable	Global variable
Default value of a Local Variable is **garbage**	Default value of a Global Variable is **0**
A local variable is **visible inside** the block and blocks nested blow	A global variable is **visible throughout** the program – main() function and all other user defined functions
Longevity of a local variable is as long as the **block is active**	**Longevity** of a global variable is as long as the **program is active**

Example:

```
#include <iostream.h>
int a=4;    // global variable
void first()
{   int b = a;   // b is local variable
     cout<<a+b;
     a++; b--;   }
void main()
{    cout<<a;   //global a
     first(a);
     cout<<b;  //Syntax error. Local variable of a
               //function is not accessible in some other function
}
```

Q35. What is an alias? How is an alias created? Give a suitable example to create an alias.

An alias is another name given to an already existing variable. An alias is created by the following rule:

DataType**&** NewVariableName=OldVariableName;

Example:

```
int x=35;
int& y=x;
```

Q36. Name the keywords which are optional in switch-case.

break and **default** are optional keywords in switch-case.

Q37 When is a scope resolution operator necessary with a global variable?

When local & global variable having the same name and to distinguish between local or global variable we use scope resolution(**::**) before global variable.

Q38 Some USEFUL Functions

```
int countdigit(int n)
{
    int count=0;
    while (n!=0)
    {
        count++;
        n/=10;
    }
    return count;
 }
```

```
int sumofdigit(int n)
{
    int sum=0;
    while (n!=0)
    {
        sum+=n%10;
        n/=10;
    }
    return sum;
}
```

```
int reverseint(int n)
{
    int num=0;
    while (n!=0)
    {
        int digit=n%10;
        num=10*num+digit;
        n/=10;
    }
    return num;
}
```

```
int checkprime(int n)
{
    int x=2;
    int prime=1;
    while (x<n && prime==1)
        if (n%x==0)
            prime=0;
        else
            x++;
    return prime;
 }
```

```
int checkarmstrong(int n)
{
    int sum=0, temp=n;
    while (n>0)
    {
        int digit=n%10;
        sum+=digit*digit*digit;
        n/=10;
    }
    return sum==temp;
 }
```

```
int checkpalidrome(int n)
{
    int num=0, temp=n;
    while (n!=0)
    {
        int digit=n%10;
        num=10*num+digit;
        n/=10;
    }
    return temp==num;
}
```

```
int productofdigit(int n)
{
int prod=1;
while (n!=0)
{
int digit=n%10;
prod*=digit;
n/=10;
}
return prod; }
```

```
int productofdigit(int n)
{
int prod=1;
while (n!=0)
{
int digit=n%10;
if (digit!=0)
prod*=digit;
n/=10;
}
return prod;  }
```

```
// Displays & find sum of all prime Nos between 2 & n
#include<iostream.h>
void sumofprime(int n)
{
    int sum=0;
    for (int k=2; k<=n; k++)
    {
        int x=2, prime=1;
        while (x<k && prime==1)
            if (k%x==0)
                prime=0;
            else
                x++;
        if (prime==1)
        {   cout<<k<<endl;
            sum+=k;
        }
    }
    cout<<"Sum Of Prime="<<sum;
}
void main()
{  int n;
    cout<<"Input n? "; cin>>n;
    sumofprime(n); }
```

```
// Displays first n Prime Nos, starting from 2
#include<iostream.h>
void generateprime(int n)
{
    int k=2, count=0;
    while (count<n)
    {
        int x=2, prime=1;
        while (x<k && prime==1)
            if (k%x==0)
                prime=0;
            else
                x++;
        if (prime==1)
        {  cout<<k<<endl;
            count++;
        }
        k++;
    }
}
void main()
{  int n;
    cout<<"Input n? "; cin>>n;
    generateprime(n);}
```

```
// Displays Armstrong Nos between 1 and n
void generatearmstrong(int n)
{
    for (int k=1; k<=n; k++)
    {
        int sum=0, temp=k;
        while (temp>0)
        {
            int digit=temp%10;
            sum+=digit*digit*digit;
            temp/=10;
        }
        if (sum==k)
            cout<<k<<endl;
    }
}
```

```
// Displays Prime Nos between  2 and n
void generateprime(int n)
{
    for (int k=2; k<=n; k++)
    {
        int x=2, prime=1;
        while (x<k && prime==1)
        {
            if (k%x==0)
                prime=0;
            x++;
        }
        if (prime==1)
            cout<<k<<endl;
    }
}
```