Page :
Date: / /

About c++ language

- found 1929 by Bayarne Stroustrup
- Extention of c language
- fast oop language
- more control over system & resources
- Best memory management
- High performance

major changes in 2011, 2014, 2017

low level: Near to hardware
Ex: Binary, Assembly

high level: Human understandable.
Ex: c++, Java

scope of variable

- local
- global

can global & local variable have same name?

Yes : First choice of compiler is local variable if local is not present then print global variable.

if both have same Name you want to print global then use cout<<::c;

Page :
Date: / /

Data Types in CPP

1) Built in : int , float
2) User Define - Enum , structure , union
3) Derived - Array , function , pointers

Header files

1) compiler के साथ आती.
2) user define - जो file user बनाता है OR उसे program में include कि जाती है.

Reference variable

means give Reference of variable & create copy.

```
int i=10;
int & e=i ; // e is Reference of i
```

Type casting in CPP

```
int a=10; float b=2.5;
cout << (int) b;
cout << int (b);
```

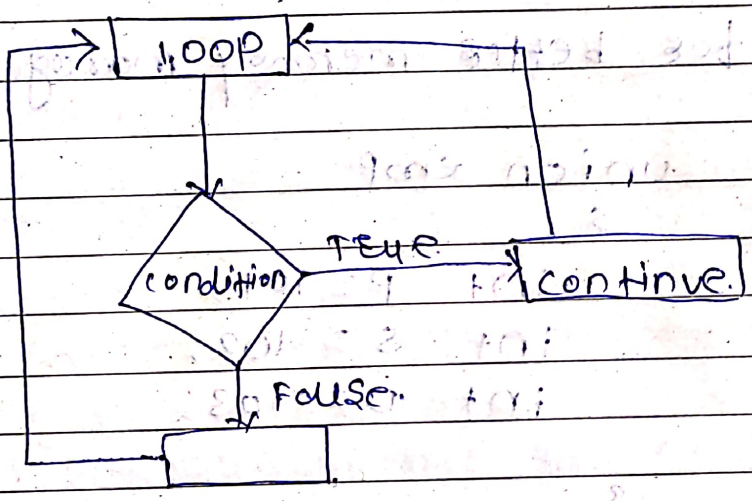
} convert int & int.

Page :
Date: / /

Break & continue statement

Break means Break program in certain condition occurs.

continue means if certain condition occurs then continue the next step or continue execution from next step




```
for( i=0 ; i<=10 ; i++)
{ if( i==6)
    continue;
  cout<< i<<endl;
}
```

output = 0, 1, 2, 3, 4, 5, 7, 8, 9, 10

if i==6 then execution continued from next step.

Page :
Date: / /

Pointers

it is variable which holds the Address of another variable.

& Address of operator
* Dereferencing operator.

union in cpp

for better memory management we use

```
union emp
{
    int P = 101;
    int S = 102;
    int D = 103;
};
```

from above declared three variables we can use only one, but if we declare separately it will use 12 Byte but union help to create 4 byte shareable memory we can use one variable any time.

in some situation union is helpful.

Page :
Date: / /

Function Prototype & Definition

#

```
int sum (int a, int b) // Prototype.
int main()
 {
    sum (n1, n2);
 }
int sum (int a, int b) // Definition.
 {
   return (a+b);
 }
```

call by value & call by Reference

call by value means you pass value to function.

call by Reference means you pass reference to functions.

```
int sum (int *a, int *b)
 {
    return ((*a) + (*b));
 }

int main()
 {
    int a=10, b=10;
    cout<< sum(&a, &b); // call by Reference.
 }
```

Page :
Date: / /

```
int sum(int a, int b)
{
    return (a+b);
}

int main()
{
    int a=10, b=10;
    sum(a,b) // call by values.
}
```

In Line Functions.

If some time function is so small we have to call that function 1000 time then we declear that function As inLine Function.

```
#
inLine int sum(int a, int b)
{
    return (a+b);
}

int main()
{
    int a=10;
    int b=20;

cout << sum(a,b);
}
```

Page :
Date: / /

in above program we decrease the function As inline means compiler can load the that function code At calling statement and make program fastes.

for the better optimization & memory management we us inline function.

Don'ts :

- if function is large then Donot Decleas As inline.
- in Recursion Do not use.
- Do not use with static variables.
- if function contains loop then Don't use.
- if function contains switch & goto statement then Don't use.

Friend Function

if someone wants to Access your private data then you have to give them Paermission in program if program wants to give paermision to Access these private data then that function have to decleare as Friend.

Page :
Date: / /

```
# include <iostream>
using namespace std;

class A
{ private:
    int data=10;

  public:
  friend void sum(A);
};

void sum (A K) // Access private data.
{
   K.data +=10;
   cout<< K.data;
}

int main ()
{
    A a;
  cout<<sum(a);
}
```

Friend class :

मान लो कि एक class है OR उस class से हमें बहुत सारे function Declare करने है OR सभी functions एक दूसरे class के private data को Access करना चाहते है तो उस class को ही Friend declare कर दो तो कोई भी function अगर Access करना चाहेगा तो कर पायेगा उसे बार बार Declaration देना नहीं पड़ेगा.

```
#include <iostream>
using namespace std;

class B; // forward Declaration.

class A
{
    private:
    int data=10;

    friend class B; // friend class
                       Declaration.
}

class B
{   private:
    int length=10;

    public:
    void add (
        A a; // object of A.
    cout << A.data + length << endl }};
```

Page :
Date: / /

```
int main()
{
    B b;
    b.add();
    seven(0);

}
```

Features of OOP

- class
- object
- Abstraction
- Encapsulation
- inheritance
- polymorphism
- message passing
- Reusability

This Pointer

This pointer पॉइंट करती है उस object पर जिसने की उस member function को call किया.

Static member function Don't have this pointer.

Page :
Date: / /

#

```
class A
{
  int a;
 public:
  void setData(int a)
  {
   this->a = a;
  }

  void getData(){
   cout<<a<<end;
  }

};
int main()
{
  A a1;
 a1.setData(4);
 a1.getData();
}
```

if we not decrease this in above function then it will show us garbage value of double variable a.

if we set a=a then it will give peerfamce to local variable first.

Page :
Date: / /

Access modifiers:

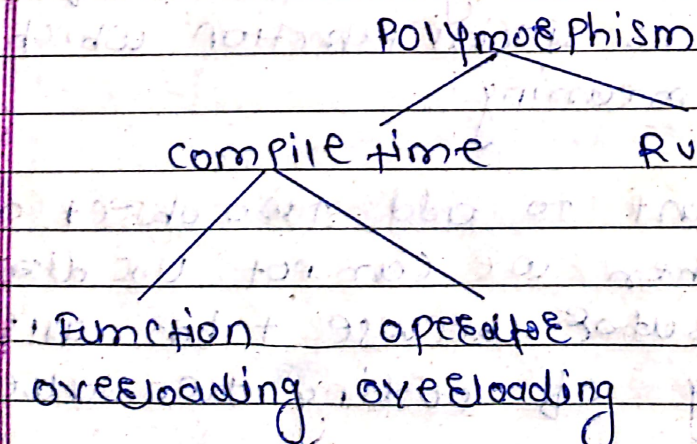
private = No one can Access
public
protected

private: No one can Access only the member function of that class can Access.

public: Anyone can Access.

protected: same as private but protected members can accessable or inherited in derived class.

☆ Poly morphism



Page :
Date: / /

Operator overloading:

- Operator overloading allows C++ operator to have user defined meaning on user defined types or classes.
- This means C++ has ability to provied the operator with special meaning for a data type This ability is known as operator overloading.
- give the additional meaning to all ready defined operators.
- The purpos of operator overloading is the provied special meaning of an operator for a user define data type.
- we can not overload (.) (::) (?:) (sizeof) operators.
- operator overloading is same like function but it is special function which has special meaning.
- if we want to add two object of class then we can not use directly (+) operator because + has already meaning of adding two values.

if we want to use (+) directly ~~then we use~~ to perform addition of two user define datatypes like object.

Page :
Date: / /

give special meaning to (+) operator through operator overloading.

```
class complex
{
private:
    int real, img;

public:
    complex(int r=0, int i=0)   // Parameterize constructor
    {
        real = r;
        img = i;
    }
    complex operator + (complex c)   // syntax of operator overloading.
    {
        complex temp;
        temp.real = real + c.real;
        temp.img = img + c.img;
        return temp;
    }
};

int main()
{
    complex c1(5, 6);
    complex c2(2, 5);
    complex c3;

    c3 = c1 + c2;   // This is possible in case of overloading operator.
}
```

Page:
Date: / /

```
class TestClass {
private:
    int count;
public:
    TestClass(): count(5) {}

    void operator --() {
        count = count-3;
    }

    void display() {
        cout<<"count:"<<count;
    }
};

int main()
{
    TestClass tc; //object
    --tc;         // we directly perform
    tc.display(); // here -- operation
                  // on object.
}
```

which is not possible in general but we give special meaning to (--) operator above that's why we can directly perform -- operation on object like this.

Page :
Date: / /

virtual function :

if we create pointer object of ~~Derived class~~ Base class and we point that object of base class to object of Derived class then we can only Asscess the functions and member variables of Base class.

but pointer object of base class का Address contains कर रहा होता derived class के object का था why we only Access the things of base class.

well that is the rule of c++ or Default behaviour of c++.

if we want base class pointer object point करे Derived class के object का and we are able to Access member variable and function of Derived class the use virtual function in base class.

Page :
Date: / /

```
class Base
{
private:
int var = 10;
virtual void display()
{
cout << var << endl;
}
};

class Derived : public Base
{
private: var2 = 20;
void display()
{
cout << var;
cout << var2;
}
};
int main()
{
Base *bptr;
Derived obj;

bptr = &obj;

bptr -> Display();

}
```

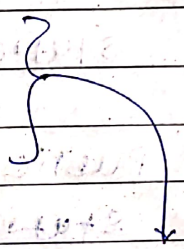
By default Behaviour

Because of virtual Function

Page :
Date: / /

Static variable:

```
#
void Fun()
{
static int num = 1;
    cout << num << endl;
    num++;
}

int main()
{
  Fun();
  Fun();
  Fun();
}
```



with static output :
1
2
3

without static
1
1
1

- once number change then in memory it remains same that's why we called it static means it update.

- if we won't Declare As static then every time run by Defult As new.

- जब तक program End नहीं होता है तब तक num मेमोरी रहेगी। After end of program num is again 1.

Page :
Date: / /

Static Function :

- static member function shared by all object of that class.
- static member function only use static data members.
- static member function called using (::) operator.

```
class Text
{
    static int count; //decrease

public:
    static getCount()
    {
        cout << count << endl;
    }
}; int Text::num=15; //initilize.

int main()
{
    Text s;
    s.getCount();
    OR
    Text::getCount();
}
```

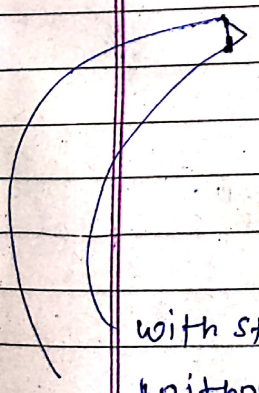
- static function only Access the static variable and static functions.
- it is shared by all object of class

Page :
Date: / /

Static object:

- Same like static member variable we can declare static object.
- The scope of static object is entire program. When program end then static object stop or release the memory.

```
class Test
{
public:
Test()
{
cout << " constructor";
}
~Test()
{
cout << " distructor";
}
};

int main()
{
if(true)
{
static Test obj;
}
cout << "end of main";
}
```



with static : constructor end of main distructor

without : constructor distructor end of main

Page :
Date: / /

Function overriding :

- if derived class defines same function as defined in it's Base class it is known as function overriding.

- overriding is needed when derived class function has to do some added or different job then the base class function.

- overriding is used to have same name function which behave differently depending upon parameters passed to them.

```
class A
{
    void print()
    {
        cout<< ok;
    }
};
class B : public A
{
    void print()
    {
        cout<<" not ok ";
    }
};
```

Page:
Date: / /

```
int main()
{
    A a;
    a.print();
    B b;
    b.print();
}
```

- in above program every object of each class called there own functions.
- if in class B there is no function like print then class A function called.

const variable

ones we declare variable As constent then we cannot change value. for entire program.

Syntax of Declaration:

```
const int p = 10;
```

we can not Assign any value to p throughout the program it is fix 10.

Ex pi = 3.14

Page :
Date: / /

Constent function:

if you want some variables of your passing values should be constent then declare constent function.

```
class Test
{
    int x;
    Base(int a)
    {
    x=a;
    }

    int getval() const  //Declaration constent function.
    {
      return x;
    }
};

int main()
{
    Test t(10);
    t.getval();

}
```

we cannot change value of t object x. it is constent if we Do that then Error occurs.

Page :
Date : / /

Constant Argument :

if Argument passes to function and then we can not change the value of that Argument it is same as passed.

```
Void print ( const int a)
{
   cout << a;
}
int main()
{
   print(10);
}
```

we can not change value of a. it is same.

Page :
Date: / /

Abstract class & pure virtual fun.

- To Define abstract class we have to decrare one pure virtual function.
- if create one pure virtual function in Base class we can not create object of that class but we can create pointer of that class.
- when you decrare pure virtual function then then that class becom abstract but you have to provide defination pure virtual function in derived class means you have to overried the pure virtual function.

Page :
Date: / /

#

```
using namespace std;

class A
{
public:
    virtual void display()=0;
};

class B : public A
{
public:
    void display()
    {
        cout<<"Derived"<<endl;
    }
};

int main()
{
    A *obj;
    B ob2;
    obj = &ob2;
    obj->display();

    OR

    B obj2;
    obj->display();

    return(0);
}
```

syntax

overried

- But we can not create object of base class. because that is abstract class.

if you inherit Abstract class into Derive and you won't overried definition the it will show ERROR OR that Derived class become Abstract.

Page :
Date: / /

C++ Encapsulation:

Encapsulation means bunding the data members & member function in single class. bunding data members & member function helps to Achive data hiding.

It helps us to write clean code. hide data.

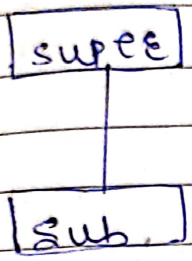
```
#
class Rect
{ private:
    int a;
  public:
  void disp()
  {
    cout<<a;
  }
};

int main()
{
  Rect a;
  a.disp();
  return(0);
}
```

in above Program Acive data hiding through private specifier and we bunde data and functions together All we called encapsulation.

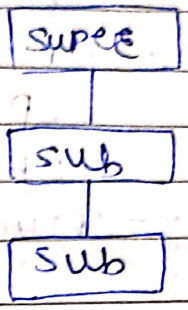
Page :
Date: / /

Inheritance:

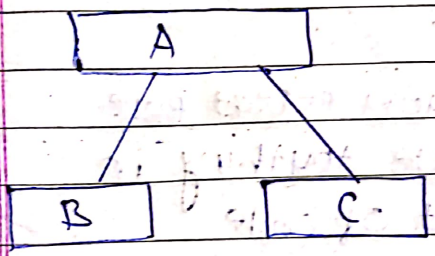
Single:



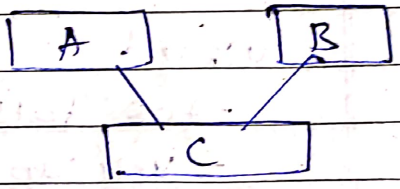
multilevel:



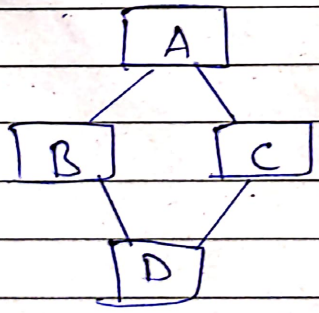
★ Hierarchical ★



★ multiple. ★



~~Cross~~



Hybrid inheritance.

Page :
Date: / /

On GitHub Read About

- Constructor
- Distructor
- copy constructor
- inheritance.
- Tempreats.

* Tempreats :
  - is A custom class

? Why?
  - for, follow NON Repeat Rule.
  - you can create Anything in small Amount of code.