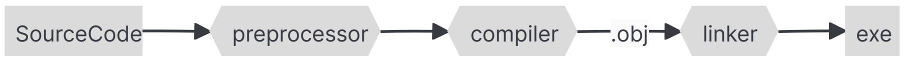
C++Notes

L1

Process of Compilation



Preprocessor

> 0. Makes a copy of the source file
> 1. `#include <stdio.h>`
>   `#define PI 3.14`
> 2. delete `//comments`

Variable declaration/definition

```
int x;     //declaration
x=2;       //value assignment

int x = 2 //definition
```

The variables in c are declared right before using them (locally);

```
int main(){
    int sum = 0;
    for(int i=1; i<10; i++){ \\i defined localy
        sum +=i;
    }
}
```

Reference Variable

> Reference variable is an alias to an existing variable

Defining a referece variable:

```
int main(){
    int x;
    int &y = x;

    y++; //it increments both x and y.
}
```

Passing Parameters

```
#include <stdio.h>

void f(int &x){
    x++;
}

int main(){
    int i = 0;
    f(i); //increment i;
    printf("%d\n",i); //2
}
```

L2

Function Overloading

Functions are defined by there:

1. Names
2. Parameter List (types)

```
#include <stdio.h>

int add(int x, int y){ return x+y;}
double add(int x, int y, double z){return x+y+z;}
int add(int x, int y, int z){ return x+y+z;}
double add(double x, double y){return x+y;}
double add(double x){return x;};

int main(){
    int x = 1, y = 1, z=1.5;
    double xd = 1.5, zd = 1.5;

    printf("%d\n", add(x,y));       //2
    printf("%d\n", add(x,y,z));     //3
    printf("%g\n", add(x,y,zd));    //3.5 it find the function with double
    printf("%d\n", add(xd,y,z));    //3   xd type conversion double->int
    printf("%g\n", add(xd, y));     //Error Ambiguous
    printf("%f\n", add(x));         //1.00000 type conversion int->double

    return 0;
}
```

Macros and Inline Functions

```
#define MAX(a,b)((a)>(b)?(a):(b))

int main(){
    x=1, y=0;
    e=MAX(++x, y);  //e = (++x)>(y)?(++x):(y)
}
```

Here the problem is that you are incrementin x several time because the macro is string copy-pasted by the preprocessor.
That is why we use inline functoins:

```
inline int max(int a, int b){
    if(a>b) return a;
    else return b;
}
```

> The inline kewwoard is a request to the compiler to not use stack calling for the fucntion but just to copy paste to the source code.

Default Parameters (Function Arguments)

Are parametes of the functions which have an inicial value even if not passed by the caller.

```
void f(int x, int y=10, double pi = 3.14){...}

int main(){    // x  y  pi
    f(1,2,3); // 1  2  3.0
    f(1,2);   // 1  2  3.14
    f(1)      // 1  10 3.14
    f();      //Compiler Error
    f(1,,3.145)//Compiler Error
}
```

Constants

You need to define a constant.

```
const int i; //compiler error
const int i=90 //good
```

You cant change the value of a constant.

```
const int x=10;
x++; //Compiler Error
```

You can also have constant arrays.

```
const int v[]={1,2,3};
v[1]++; //error you can only read
```

Pointers and constants

```
char a[] = "Hello";

const char *p = a; //constant char
*p1 = 'k'; //Compiler Error
p1++;

char const *p = a; //constant pointer
*p1 = 'k';
p++;      //Compilor Error

const char const *p = a; //constant pointer and char
*p1 = 'k'; //Compilor Error
p++;       //Compilor Error
```

Constant Reference
You can define it 2 ways.

```
int const &i = j; //Theay are equall
const int &i = j;
```

L3

cpp as an OO language

1. Encapsulation (atributes + methods)
2. Data Hiding (information hiding)

3. Inheritance (Polymorphism)

Encapsulation Class

Class def'n

> Is a templeate which tells how to extensiate enteties.
> It is a user-defined data type which has:
>
> 1. data members (attributes)
> 2. member functions (methods) (operations which can be performed to the data)

Class creation

```
class Student{...};

int main(){
    Student A("...","...","...");
}
```

Look at a class in c

```
struct Point{
    int x;
    int y;
};

void draw(Point &p){
    api_putpixel(p.x,p.y);
}

int main(){
    struct Point P;
    p.x=10; p.y=20;
    draw(p1);
    return 0;
};
```

There is:

1. No connection
2. If we have `struct Circle{}; struct Rectangle{};` than for every struct we need a fucntion with a different name. Is ambiguous.

Now look at a class in cpp

```
class Point{
public:
    \\atributes
    int x;
    int y;
    \\methods
    void draw(){
        api_putpixel(x,y)
    }
    void setx(int nx){ \\this is a setter
        if(nx>0 && nx<640)
        x=nx;
    }
};
```

```
int main(){
    Point p1;

    p1.x=-4324324235; p1.y=14; \\not safe
    p1.draw();

    setx(50); \\safer
    p1.draw();
}
```

Setters , Data Hiding, this->x

The method `void setx()` is a setter. It helps set the value of the atributes while checkin if the input is valid.

So the user doesn't make mistakes we do not need to allow unsafe attribite value assigment. Therefore we use data hidins keywords:

1. public - visible to every element outsidet the class
2. protected - only members inside can acces (Children)
3. private - only from inside of the class

Here is how our safer class should look:

```
class Point{
private:  \\or northing
    int x;
    int y;
public:
    void draw(){
        api_putpixel(x,y)
    }
    void setx(int x){
        this->x = x;
    }
};
```

The `this` keyword has the adresses of the class.

Encapsualtion(Class) Benefits

The class with attributes and and methods is encapsulated.

> The benefits Encapsulation:
> 1. The object remains consistet (security)
> 2. Simpler usage of the class (interface)

Init Function and Contructor

Look at this code:

```
int main(){
    Point p1;
    p1.x = 20; \\y had memory garbage
    p1.draw(); \\Error may happen
}
```

We need to make shure this dosnt happen.

When we initiate a member of a class we get memory garbage. Therefore we need
to assign vales to the attributes when the class starts.

One way i susing the init function:

```
class Point{
private:
    int x;
    int y;
public:
    void Init(int x = 0, int y = 0){
        this->x = x;
        this->y = y;
    }
    void draw(){
        api_putpixel(x,y)
    }
    void setx(int x){
        this->x = x;
    }
};

int main(){
    Point p1;
    p1.Init(10,20); \\user should call
}
```

The problem with it is that the user should call the function. Therefore we need something different that calls itself every time the class is declared. We call this special(method) function a contructor.

Def'n Constructor

> A contructor is a special method that will be executed automatically always after the creation of the object.

```
class Point{
private:
    int x;
    int y;
public:
    \\same name as the \\class is a contructor
    Point(int x = 0, int y = 0){
        this->x = x;
        this->y = y;
    }
    void draw(){
        api_putpixel(x,y)
    }
    void setx(int x){
        this->x = x;
    }
};

int main(){
    Point p1(10,20);
    Point p2(10); Point p2 = 10;
    Point p1();
}
```

When you have on parameter for the contructor you can use the assign.
Int is a class so:

```
int x=50; or int x(50);
```

Class.h Class.cpp

Look at class.h:

```
#ifndef CLASS_H
#define CLASS_H

class Class{ //we dont use class name but for the purpse of example
    int x;
    double y;
    ...
public:
    Class(int x_=0, double y_=3.14);
    void method1();
    int method2(int x_);
    ...
}

#endif
```

Look at class.cpp:

```
!!!//incldue class
#include "class.h"
#include ....

Class:Class(int x_, double y_){ !!!//no default parameters
    .....
}
void Class:method1(){
    ....
}
int Class:method2(int x_){

}
```

Look at main.cpp:

```
int main{
    Class class; //you can have the same name ass the class
    Class class2(10, 8,5);

}
```

L4

Destructor

> Special method called in the release of the object
> `~MyClass()`

It is used to delete dynamically allocated memory.

Copy Constructor

> Special method called automatically when coping the object

You can access private attributes of every object of the class form another object of the class if you pass by reference. Const just doesn't let you change them.

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

class Point{
    int x;
    int y;
 public:
    Point(int x = 0, int y = 0){
        this->x = x;
        this->y = y;
    }
    Point(const Point& p){
        x = p.x;
        y = p.y;
    }
    //becuase of this method function every object of the
    //same Class can access the others private attributes
    void method(Point& p){ //(const Point& p)-> you can only read
        p.x=10;
        p.y=100;
    }
    void print(){
        cout<<"x = "<<x<<" y = "<<y<<endl;
    }
};

int main(){
    Point a(10,11);
    Point b(a);

    a.print();//x = 10 y = 11
    b.print();//x = 10 y = 11

    b.method(a);
    a.print();//x = 10 y = 100

    return 0;
}
```

Dynamic Memory Allocation

```
int main(){
    int *p = new int[3]; //p->[ | | ]
    int *a = new int; //a->[ ]
    Point *pp = new Point(10,12); //pp->[{10,12}]

    delete a;
    delete[]p;
    delete pp;
}
```

Initialization List

> Gives an opportunity to declare and assign during the creation of the object

```
class Point{
    int x,y;
    public:
```

```
    Point(int x, int y):x(x),y(y){
        ...
    }

};
```

Constants in Classes

Constant variable

```
class A{
    static const int c;
};
const int A::c=10; [header]
```

Constant method and mutable

> Constant method is a method of the class which does not modify the state of the class.
> Mutable allows the variable to be changed by a const function.

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

class Point{
    double mutable x;
    double y;
    public:
    Point(double x=0, double y=0):x(x),y(y){}
    double radius() const{
        x=20;
        y=20;
        return x + y;
    }
};

int main(){
    Point point(10,10);
    cout<<point.radius();

    return 0;
}
```

L5

Static class members

> shared instance
> we reference them with :: (scope operator)

```
class Point{
        double x;
    double y;
        public:
        static string name;
        Point(double x=0, double y=0):x(x),y(y){}
};
string Point::name = "point";

int main(){
        Point point(10,10);
        cout<<Point::name<<endl; //point
        return 0;
}
```

Friend Function and Classes

> if you want a function to be able to access the private attributes or methods of your class you friend that function or class.

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

class Point{
    double x;
    double y;
    public:
    friend double radius(Point& point);
    friend class Point_V;
    Point(double x=0, double y=0):x(x),y(y){}
};

class Point_V{
    double v;
    public:
    void setv(Point& point){
        v = point.y;
    }
};

double radius(Point& point){
    return point.x + point.y;
}

int main(){
    Point point(10,10);
    cout<<radius(point);
    return 0;
}
```

Namespaces

> A way to group the names of variables and function.
> Preprocessor handles namespaces.

```
#include <iostream>

namespace est_geo{
    const double PI = 3.14;
    double circum(double r){
        return 2*PI*r;
    }
    double area(double r){
        return PI*r*r;
    }
}

namespace pe_geo{
    const double PI = 3.14159265;
    double circum(double r){
        return 2*PI*r;
    }
    double area(double r){
            return PI*r*r;
        }
}


int main(){
    double R;
    std::cin>>R;

    std::cout<<est_geo::area(R)<<std::endl;
    std::cout<<pe_geo::area(R)<<std::endl;

    using namespace est_geo;

    std::cout<<circum(R)<<std::endl;
    std::cout<<pe_geo::circum(R)<<std::endl;

    return 0;
}
```

Input/output

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

int main(){
        double x;
        cin>>x;
        x++;
        cout<<x;
    clog<<x;
    cerr<<x;

    return 0;
}
```

L6

Operator Overloading

Difference between the function and operator:

1. Different way to pass parameter (left, right)
2. You can only have max 2 parameters

Operator can be overloaded:

- name and parameter list
- at least one parameter must not be a build in type
- new operator can not be created
- order and precedence remain the same
- some operators can not be overloaded:
  . (dot)
  : (scope)

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

class Complex{
    double a,b;
    public:
    Complex(double a=0, double b=0):a(a),b(b){}
    //we want to add to complex numbers with the + operator
    Complex operator+(const Complex& c) const{
        return Complex(a+c.a,b+c.b);
    }
    friend ostream& operator<<(ostream& os, const Complex& x);
};

ostream& operator<<(ostream& os, const Complex& c){
    os<<c.a<<"+"<<c.b<<"i"<<endl;
    return os;
}

int main(){
    Complex c1(8,6);
    Complex c2(2,4);

    Complex c3 = c1.operator+(c2);
    Complex c4 = c1 + c2;

    cout<<c3<<endl;
    cout<<c4<<endl;
    return 0;
}
```

Don't forget the 2 constants inside the class:

1. To make the right parameter safe (const Complex& c)
2. To make the left parameter safe (function const{})