OOP Notes

عند تعريف (struct) user defined datatype .. مش محتاج ف ال main اكتب struct .
تعتبر cin , cout .. objects و ليس functions .
الـ << و >> بنعملهم operator overloading .

الـ cin .. بتدعم الـ multiple reading و ميفيش format specifier .

الـ allocating and deallocating :
بنستخدم 2 keywords .. new & delete .. و الكلمتين دول بيرجعوا pointer .

```
Int *ptr;
Ptr=new int ;
```

نلاحظ انه بيحجز single cell بيشاور على عنوانها ف الميموري .

لكن ف الغالب بنعمل array allocating :

```
Int n;
Cin >> n
Ptr= new int [n]; …… allocating
Delete ( [ ] ptr ) ; …….. deallocating
```

الفرق بين call by value و call by reference :

By value : بياخد نسخه من البيانات اللى ف الـ main .. و يوديها لل function .. و بعد ال calling نتيجة الـ function بتختفى من الـ memory .
ولا يحدث تغيير فى الـ original data .

By reference: يجمع بين خصائص ال calling by value and address .
الـ function parameters تعتبر Alias name للـ original data .
التغيير الذى تحدثه الدالة .. يحدث على الـ original data .
لا يتم حجز اماكن جديدة فى الميموري للـ parameters لانه calling by reference .

Lec 2

تعتبر ال Class.. blueprint او user defined type .
تعتبر ايضاً template تحتوى على مجموعة attribute و (methods (actions .

الـ object الذى يتم انشاءه من الكلاس .. يحتوى على نفس ال methods , attr .

يتم استخدام مفهوم separation of concern بين تعريف الـ attribute و استخدامه
الـ functions لو متعلقة بالـ class attribute تبقى member function
لو مش متعلقة بال class fields يبقى نعرفها بره الكلاس على انها Stand alone fun .

خلى بالك الـ C++ not fully OOP و لكن يمكن ممارسة قواعد البرمجة الهيكلية .

Data hiding : encapsulation

رغم اننا عرفنا الـ fieldsعلى انها private .. الا انى فى get function
بعمل Return للـ attribute عادى باسمه طالما احنا جوا نفس الكلاس .

ليه بنستخدم data hiding ؟
عشان نعمل conditions على الـ input فى الـ set functions .
و نعمل full control of values .
نعمل complete separation بين .. تعريف ال field و استخدامه .

خلى بالك الـ struct بيكون محتواه public by default .
الكلاس محتوياتها private by default .

خلى بالك فى الـ functions :

الـ return keyword بتعمل anonymous object (temp)
مش متخزن فى حته لسه ف الميموري .. لكن موجود فى الـ runtime .. الى ان يتم استخدامه .

مينفعش ارجع reference لـ local variable .

: Function overloading

هو احد اوجه الـ polymorphism .
عبارة عن ان يحدث اختلاف فى (نوع – عدد – ترتيب) الـ parameters .
الـ Returnملوش علاقة بالموضوع .

مش بطئ عشان بيحصل نتيجة للـ static binding .
ممكن اعمل Return لـ object جواه كذا قيمة بالتالى ده يسمى composite return .
قيمة الـ membersالخاصة بالـ object عبارة عن .. .rubbish

الـ automatic initialization : constructor :

هو عبارة عن special function يتم استدعاءه مره واحده فى عمر الـ object .
ياخد نفس اسم الكلاس .
هناك constructor overloading .
بيرجع بـ void .. و ملوش Return type .

لو مش عامل ولا واحد .. يتم انشاء constructor by default .
انما لو تم عمل واحد على الاقل .. يتم الغاء الـ Default constructor اوتوماتيك .

Lec 3

الـ Destructor بيكون public.
ال constructor بيكون public او private .
هنتكلم على ال stack :
انا مش عاوز Static fixed allocation ..
اذن هنعمل ; [10] stk = new int .. ده معناه ان ال stkبيشاور على array موجود فى الـ Heap .. بيشاور على اول Address فى الـ heap .
مع النهاية تنتهى الـ Stack و لا تنتهى الـ heap .. لذلك تظهر فائدة ال destructor .

: Static member variable
عبارة عن shared variable بين كل الناس .. و يحمل القيمة النهائية
لان كل الـ objects بيشاروا عليه .. و اى تغيير يحصل فيه .. كلهم يحسوا بنفس التغيير .

بنادى عليه بره الكلاس من خلال الـ :: Scope operator .
مش بياخد this.

ازاى نكتب الـ functionبطريقة Generic .. تتماشى مع اى data type ؟

اذن يتم استخدام template function .. فنجد ان ال compiler بيـ generate نسخ حسب ال data type اللى بيحددها المستخدم فى الـ main .

مثلا : template < class T >.. و هكذا .
طالما بقينا بره الكلاس و بعدها لازم نكتب template < class T > تانى .
فى الـ main مينفعش اعمل اكتر من data type.. هى بتبقى واحده per program .

Lec 4

Default input parameter :
عبارة عن دمج بين الـ Default constructor و ال parameterized .
بقدر ادى initial value لو ال user معرف object من غير values .

هام : مينفعش ادى parameter قيمة و الباقى لا !!!
طالما اديت قيمة لواحد لازم ادى اللى (بعده) .

مثال : لو عندى
Long multi (int a = 1 , int b = 2 , int c = 3 , int d =4)
و عملت ; (cout << multi (10 , 20
يبقى هيدي 10 للـ a .. و هيدي 20 لل b .
و c = 3 , d = 4 .

لو عملت Long multi (int a , int b = 2 , int c, int d) .. ده غلط
الصح اننا نعمل : Long multi (int a, int b, int c = 3 , int d =4)

Friend function :
Friend functions are functions defined outside a class (ie, not member functions), but which the class declares to be *friends* so that they can use the class's private members. This is commonly used in operator overloading.

انك تباصى اوبجكت من الـ main الى function.. هو بيعمل Shallow copy
مش بينسخ الداتا .. هو فقط بيعمل pointer بيشاور على نفس مكان الداتا .

و لما بنعمل Delete بيخلي المكان unallocated .

رحلة المشاكل .. : dynamic area problem
ان كله بيشاور على نفس الـ array data .. و بيحصل pass by value .. و بيبقى عندى 2 pointer بيشاوروا على نفس الـ data !! و التغيير فى اى واحد فيهم هيعمل تغيير ف الداتا الاصلية ..

الحل اننا نبعت الداتا by reference ..بحيث يبقى عندى object واحد و فى الدالة هنستخدم Alias name فقط .. يبقى مفيش objectجديد هيتعمل ولا Destructor .

لكن لسه بيحصل تعديل ف الداتا الاصلية !! لما بنلعب ف ال function .

يبقى نبعت الداتا من خلال constant reference :
يبقى عندنا object بعمل منه Reading only و مينفعش اعدل فى original data .
كأن ال fields > protected .

طب بردو محتاجين حل افضل ؟
نعمل نسخة حقيقية من الـ object نوديها كـ parameter.. و لما ادمر النسخة الحقيقية .. الداتا الاصلية تفضل زى ما هى ..
الحل نعمل Deep copy .. نعمل allocate و ننقل الداتا .. ده من خلال :

Copy constructor

هام : علامة = .. بتعمل member wise copy .
الـ local variable بيرجع دايما by value .
الـ temp object بيتعمل بال copy constructor .

كلمة this .. عبارة عن pointer للـ caller object .

Lec 5

لما بنعمل person p1 = p2;
يتم استدعاء copy constructor .. و ده عبارة عن member wise copy .

طب لتوفير الاوبجكتس .. ابعت بـ reference و const و هكذا .

: Operator overloading
لو عندنا c3 = c1+c2 ; .. يبقى c1عبارة عن invokerيستخدم this .
الـ c2بيروح للـ parameter بتاع ال Function بتاعت الـ operator overloading .

- A variable is declared which is *initialized from another object*, eg,

```
Person q("Mickey"); // constructor is used to build q.
Person r(p);        // copy constructor is used to build r.
Person p = q;       // copy constructor is used to initialize
 in declaration.
```

```
p = q;              // Assignment operator, no constructor or
copy constructor.
```

- A value parameter is initialized from its corresponding argument.

```
f(p);               // copy constructor initializes formal
value parameter.
```

- An object is returned by a function.

طب لو عندنا c3 = 7+c1; .. يبقى دى محتاجة stand alone function .. طب ليه ؟
عشان مش عارف انهى الـ invoker .. مفيش ع الشمال حاجة !!

طب لو عندى c3 = c1+=c2;
خلي بالك .. c1 بيتعملها Return < *this .. لانه pointer للـ caller .

---

http://www.fredosaurus.com/notes-cpp/oop-condestructors/copyconstructors.html