```
template<class TClau, class TValor>
class Map
{
public:
    Map();
    Map(const Map<TClau, TValor>& m);
    ~Map() {}

    int longitud() const;
    bool esBuit() const;
    Map<TClau, TValor>& operator=(const Map< TClau, TValor>& m);

    TValor& operator[](const TClau& clau);
    const TValor& operator[](const TClau& clau) const;
    TClau& operator[](const int& posicio);
    const TClau& operator[](const int& posicio) const;

    void afegeix(const TClau& clau, const TValor& valor);
private:
    vector<pair<TClau, TValor>> m_vector;
    TClau defectoC;
    TValor defectoV;
};
```

```
template<class T>
inline T& SmartPointer<T>::operator*()
{
    cout << "op*" << endl;

    if (pointer == NULL)
        cout << "ERROR; APUNTADOR A NULL"<<endl;

    return *pointer;
}

template<class T>
inline T* SmartPointer<T>::operator->()
{
    cout << "op->" << endl;
    if (pointer == NULL)
        cout << "ERROR; APUNTADOR A NULL" << endl;

    return pointer;
}
```

```
template<class TClau, class TValor>
inline Map<TClau, TValor>& Map<TClau, TValor>::operator=(const Map<TClau, TValor>& m)
{
    if (this != &m)
    {
        int newsize = m.m_vector.size();
        m_vector.resize(newsize);

        for (int i = 0; i < newsize; i++)
            m_vector[i] = m.m_vector[i];
    }

    return *this;
}

template<class TClau, class TValor>
inline TValor & Map<TClau, TValor>::operator[](const TClau & clau)
{
    int t = m_vector.size() - 1;
    int b = 0;
    int m;

    bool trobat = false;

    while (b <= t && !trobat)
    {
        m = (b + t);
        m /= 2;
        if (clau == m_vector[m].first)
        {
            trobat = true;
            return m_vector[m].second;
        }
        else
        {
            if (clau < m_vector[m].first)
                t = m - 1;
            else
                b = m + 1;
        }
    }

    if (!trobat)
        return defectoV;
}
```

```
late<class TClau, class TValor>
ne const TClau & Map<TClau, TValor>::operator[](const int & posicio) const

int size = m_vector.size();

if ((posicio > 0) && (posicio < size))
    return m_vector[posicio].first;
else
    return defectoC;
```

```
template<class TClau, class TValor>
inline void Map<TClau, TValor>::afegeix(const TClau & clau, const TValor & valor)
{
    bool trobat = false;

    typename vector<pair<TClau, TValor>>::iterator actual = m_vector.begin();

    while (actual != m_vector.end() && !trobat)
    {
        if (clau < actual->first)
        {
            trobat = true;
            pair<TClau, TValor> nuevo;
            nuevo = make_pair(clau, valor);
            m_vector.insert(actual, nuevo);
        }
        else
        {
            if (clau == actual->first)
            {
                trobat = true;
                actual->second = valor;
            }
            else
                actual++;
        }
    }
}
```

Constructors:
1. Crida al constructor per defecte de la classe base
2. Crida al constructor específic de la classe derivada

Destructors:
1. Crida al destructor de la classe derivada
2. Crida al destructor de la classe base

```
void Llibre::mostra()
{
    Producte::mostra();
    cout << "Titol: " << m
    cout << "Autor: " << m
    cout << "N. pagines: " << m_nPagines << endl;
}
```

- Crida explícita a `mostra` de la classe base
- S'ha de posar si volem mostrar informació de la classe base

```
class Llibre: public P
{
public:
    ...
    void mostra();
    ...
```

- Mètode de la classe derivada
- Mostra informació específica de la classe derivada
- Substitueix la definició de `mostra` de la classe base

El procés d'herència es pot fer de dues maneres:

- `public`

```
class Llibre: public Producte
```

- `private`

```
class Llibre: private Producte
```

Qualificació a la classe base	Accés des de classes no derivades	Herència **public**		Herència **private**	
		Accés des de classe derivada	Qualificació a la classe derivada	Accés des de classe derivada	Qualificació a la classe derivada
private	no accessible	no accessible	private	no accessible	private
protected	no accessible	accessible	protected	accessible	private
public	accessible	accessible	public	accessible	private

Un objecte de la classe derivada es pot convertir a un objecte de la classe base encara que es perdi informació

```
Objecte_classe_base = Objecte_classe_derivada
```

Un objecte de la classe base NO es pot convertir a un objecte de la classe derivada perquè no disposem de valor per tots els atributs específics

```
Objecte_classe_derivada = Objecte_classe_base
```

- Una classe derivada no té accés a les dades privades de la classe base
- Si ens interessa accedir a la part privada de la classe base:
  - Declarar els atributs/mètodes a la classe base com a **`protected`**:
    - Les classes derivades hi poden accedir com si fossin `public`
    - La resta de classes no derivades ho segueixen veient com a `private` i per tant, no hi poden accedir

```
class Producte
{
public:
    ...
    virtual void mostra();
    ...
};
```

Declaració del mètode com a virtual

```
class Llibre: public Producte
{
public:
    ...
    void mostra();
    ...
};
```

```
class Electrodomestic : public Producte
{
public:
...
  void mostra();
...
};
```

Sense utilitzar apuntadors

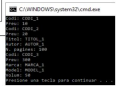
```
list<Producte> llistaProductes;
llistaProductes.push_back(p);
llistaProductes.push_back(l);
llistaProductes.push_back(e);
list<Producte>::iterator it;
for (it = llistaProductes.begin();
  it != llistaProductes.end(); it++)
  it->mostra();
```

Utilitzant apuntadors

```
list<Producte*> llistaProductes;
llistaProductes.push_back(&p);
llistaProductes.push_back(&l);
llistaProductes.push_back(&e);
list<Producte*>::iterator it;
for (it = llistaProductes.begin();
  it != llistaProductes.end(); it++)
  (*it)->mostra();
```

Crida sempre a mètode `mostra` de la classe `Producte`

Crida al mètode `mostra` de la clase utilitzada per inicialitzar l'apuntador



Destructors virtuals

```
bool LlistaProductes::eliminaProducte(co
{
  ...
  if ((*it)->getCodi() == codi)
      delete *it;
      m_productes.erase(it);
      return true;
  else
      it++;
}
```

```
class Producte
{
    ...
    virtual ~Producte();
}
```

```
Eliminant Llibre
Destructor Llibre
Destructor Producte
Eliminant Electrodomestic
Destructor Electrodomestic
Destructor Producte
```

```
cout << "Eliminant Llibre" << endl;
llista.eliminaProducte(l1.getCodi());
cout << "Eliminant Electrodomestic" << endl;
llista.eliminaProducte(e1.getCodi());
```

- Al declarar el destructor com a virtual dinàmicament es crida al destructor de la classe derivada de l'objecte corresponent (`Llibre` / `Electrodomestic`)
- Recordem que primer es crida al destructor de la classe derivada (`Llibre` / `Electrodomestic`) i després al de la classe base (`Producte`)
- Els constructors no es poden declarar com a virtuals (sempre es crida al constructor de l'objecte que estem creant)

```
class Producte
{
public:
  ...
  virtual Producte* clone() { return new Producte(*this); }
  ...
}
```

- A la classe base afegim un mètode virtual `clone()`
- Retorna un apuntador a un nou objecte dinàmic de la classe inicialitzat com una còpia de l'objecte actual

```
class Llibre: public Producte
{
public:
  ...
  Llibre* clone() { return new Llibre(*this); }
  ...
}
```

- A la classe derivada redefinim aquest mètode canviant el tipus de retorn de l'apuntador i canviant el tipus de l'objecte dinàmic que es crea

```
class Producte
{
public:
  ...
  virtual Producte* clone() = 0;
  virtual float calculaPreu(int nUnitats) = 0;
  virtual float calculaDespesesEnviament() = 0;
  virtual void mostra() = 0;
private:
  ...
};
```

```
void Producte::mostra()
{
  cout << "Codi: " << m_codi << endl;
  cout << "Preu: " << m_preu << endl;
}
```

```
void Llibre::mostra()
{
  Producte::mostra();
  ...
}
```

```
virtual funcio() = 0;
```

Funcions virtuals pures

- Mètodes que no s'utilitzen a la classe base
- Mètodes que volem que la classe derivada redefineixi obligatòriament
- Podem definir codi pel mètode de la classe base (p.ex. `calculaDespesesEnviament`, `mostra`), però no cal si no és necessari (p.ex. `clone`, `calculaPreu`)
- Les classes derivades han de redefinir el codi del mètode. Si no ho fan el mètode també passa a ser una funció virtual pura a la classe derivada

Classes Abstractes

Qualsevol classe que tingui alguna funció virtual pura passa a ser una **classe abstracta**

No se poden declarar objectes de les classes abstractes

```
Producte p;
```
⟶ ERROR

Sí que es poden declarar apuntadors a objectes de les classes abstractes (per poder manipular objectes de les classes derivades

```
Producte *p = new Llibre;
...
p->mostra();
```

Les classes abstractes permeten definir una interfície pública que tots les classes derivades han de tenir i redefinir de forma específica

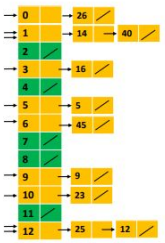
Col·lisions solució1: Encadenament separat o Hash Obert:

- La taula hash es converteix en un vector de llistes
- A cada posició del vector guardem, a una llista, tots els elements que s'han indexat a la mateixa posició.
- **Factor de càrrega λ = n / m**. És el nombre mig de claus per llista. S'ha d'intentar que sigui **menor a 1, 0.75** estaria be. (**n** és el nombre d'elements desats a la taula i **m** el nombre de posicions.). Llistes d'1 element.
- **Problema:** si les llistes són molt llargues les cerques no són eficients.
- **Solució:** Incrementar mida de la taula i fer **rehashing**

Rehashing: Si fem la taula més gran M varia i també varia h(x). Per tant hem de recalcular les posicions de totes les dades desades. Aquest procés s'anomena rehashing i és costós.

Exemple: 23,45,16,26,40,14,5,12,9,25.

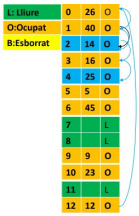
Amb h(x) = x%13



Exemple:
Amb h(x) = x%13 i M=13

Amb sondeig lineal: $h_i(x)=(h(x)+i)\%M$

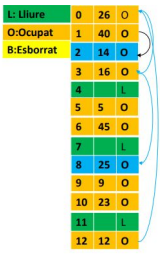
Inserim:
23%13=10
45%13=6
16%13=3
26%13=0
40%13=1
14%13=1 !! COL·LISIÓ → Reassignació: (1+1)%13=2
5%13=5
12%13=12
9%13=9
25%13=12 !! COL·LISIÓ
→ Reassignació: (12+1)%13=0; (12+2)%13=1; (12+3)%13=2; (12+4)%13=3;(12+5)%13=4;



Exemple:
Amb h(x) = x%13 i M=13

Amb sondeig quadràtic: $h_i(x)=(h(x)+ i^2)\%M$

Inserim:
23%13=10
45%13=6
16%13=3
26%13=0
40%13=1
14%13=1 !! COL·LISIÓ → Reassignació: (1+ 1²)%13=2
5%13=5
12%13=12
9%13=9
25%13=12 !! COL·LISIÓ
→ Reassignació: (12+1)%13=0; (12+4)%13=3; (12+9)%13=8



solució: Inserció Esborrat Hashing tancat.Estructura

```
class Hash
{
  public:
    Hash(int numIni=10, const string& descripcioDefecte="Definicio no existent") ;
    ~Hash();
    string& operator[](const string& clau);
    void insert(const string& clau, const string& descripció);
    bool find(const string& clau, string& definicio) const;
    bool esborra(const string& clau);
    friend ostream& operator<<(ostream &out, const Hash& h);
  private:
    int codi(string s) const;
    void insertIntern(const string& clau, const string& descripcio);
    std::vector<std::pair<string, string>> m_diccionari;
    std::vector<int> m_estat;
    int m_numOcupats;
    int m_maxElements;
    string m_descripcioDefecte;
    static const int LLIURE = 0;
    static const int OCUPAT = 1;
    static const int ESBORRAT = 2;
};
```

Desem a cada posició el seu estat: Lliure, Ocupat o esborrat

Definim els diferents estats: Lliure, Ocupat o esborrat

```
#include <iostream>
#include <vector>
#include <utility>
#include <string>
using namespace std;
```

solució: Implementar Inserció Hashing tancat. Constructor

```
Hash::Hash(int numIni, const string& descripcioDefecte)
{
    m_maxElements = numIni;
    m_descripcioDefecte=descripcioDefecte;

    m_diccionari.resize(m_maxElements,pair<string,string>("",""));
    m_numOcupats = 0;
}
```

```
Hash::Hash(int numIni, const string& descripcioDefecte)
{
    m_maxElements = numIni;
    m_descripcioDefecte=descripcioDefecte;
    m_diccionari.resize(m_maxElements, pair<string, string>("", ""));

    m_estat.resize(m_maxElements, LLIURE);

    m_numOcupats = 0;
}
```

solució: Implementar Inserció Hashing tancat. InsertIntern

```
void Hash::insertIntern(const string& clau, const string& descripcio)
{
    int index = codi(clau);
    int vegades = 1;
    while ((m_estat[index] == OCUPAT) && (vegades <m_maxElements))
    {
        index = codi2(index, vegades);
        vegades++;
    }
    if (vegades >= m_maxElements)throw "ERROR AL TROBAR POSICIO INSERCIO A HASH";

    m_diccionari[index].first = clau;
    m_diccionari[index].second = descripcio;
    m_estat[index] = OCUPAT;
    m_numOcupats++;
}
```

Com encara no tenim redimensió de la taula posem aquest control d'excepcions

```
int Hash::codi2(int index, int vegades) const
{
        return (int)(index + pow(vegades, 2)) % m_maxElements;
}
```

solució: Implementar Inserció Hashing tancat. Insert

```
void Hash::insert(const string& clau, const string& descripcio)
{//Cerquem posicio amb collisions
  int index = codi(clau);
  int vegades = 1;
  bool trobat = false;
  while ((m_estat[index] != LLIURE) && (!trobat))
  {
      if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
          trobat = true;
      else
      {
          index = codi2(index,vegades);
          vegades++;
      }
  }
  if ((m_estat[index] == OCUPAT) && (m_diccionari[index].first == clau))
  {//Element trobat es una modificacio de la seva descripcio
      m_diccionari[index].second = descripcio;
  }
  else{   //Inserim un nou element
      insertIntern(clau, descripcio);
  }
}
```

Cerca element

Les posicions poden ser OCUPAT o ESBORRAT

Valida si trobat i modifica

Insereix nou element

solució: Implementar Inserció Hashing tancat. Cerca

```
nt Hash::cerca(const string& clau) const
//Cerquem posicio amb collisions
{
  int index = codi(clau);
  int vegades = 1;
  while ((m_estat[index] != LLIURE) && (!trobat))
  {
      if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
          trobat = true;
      else
      {
          index = codi2(index,vegades);
          vegades++;
      }
  }
  return index;
```

Cerca element

Les posicions poden ser OCUPAT o ESBORRAT

solució: Implementar Inserció Hashing tancat.Estructura

```
class Hash
{public:
    Hash(int numIni=10, const string& descripcioDefecte="Definicio no existent") ;
    ~Hash();
    string& operator[](const string& clau);
    void insert(const string& clau, const string& descripció);
    bool find(const string& clau, string& definicio) const;
    bool esborra(const string& clau);
    friend ostream& operator<<(ostream &out, const Hash& h);
  private:
    int codi(const string& clau) const;
    int codi2(int index, int vegades) const;
    void insertIntern(const string& clau, const string& descripcio);
    int cerca(const string& clau) const;
    std::vector<std::pair<string, string>> m_diccionari;
    std::vector<int> m_estat;
    int m_numOcupats;
    int m_maxElements;
    string m_descripcioDefecte;
    static const int LLIURE = 0;
    static const int OCUPAT = 1;
    static const int ESBORRAT = 2;
```

Afegim cerca

```
#include <iostream>
#include <vector>
#include <utility>
#include <string>
using namespace std;
```

solució: Implementar Inserció Hashing tancat. Insert

```
void Hash::insert(const string& clau, const string& descripcio)
{//Cerquem posicio amb collisions
  int index = cerca(clau);

  if ((m_estat[index] == OCUPAT) && (m_diccionari[index].first == clau))
  {//Element trobat es una modificacio de la seva descripcio
      m_diccionari[index].second = descripcio;
  }
  else
  {   //Inserim un nou element
      insertIntern(clau, descripcio);
  }
}
```

Cerca element

Valida si trobat i modifica

Insereix nou element

solució: Implementar Inserció Hashing tancat. Find

```
bool Hash::find(const string& clau, string& definicio) const
{
  int index = cerca(clau);

  bool trobat = false;
  if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
  {
      definicio = m_diccionari[index].second;
      trobat = true;
  }
  else{      definicio = "";}
  return trobat;
}
```

Cerca element

Valida si trobat i retorna

No trobat

solució: Implementar Inserció Hashing tancat. Cerca

```
string& Hash::operator[](const string& clau)
{ int index = cerca(clau);

  if ((m_estat[index] == OCUPAT) && (m_diccionari[index].first == clau))
  {//Element trobat
    return m_diccionari[index].second;
  }
  else
  {//Element no trobat, l'inserim
    insertIntern(clau, m_descripcioDefecte);
    return m_descripcioDefecte;
  }
```

Cerca element

Valida si trobat i retorna

No trobat i l'inserim

solució: Implementar esborra Hashing tancat.

```
bool Hash::esborra(const string& clau)
{
  int index = cerca(clau);

  bool trobat = false;
  if ((m_estat[index] == OCUPAT) &&(m_diccionari[index].first == clau))
  {//Trobat l'esborrem
      //m_diccionari[index].second="";
      //m_diccionari[index].first= "";
      m_estat[index] = ESBORRAT;
      m_numOcupats--;
      trobat = true;
  }

  return trobat;
}
```

Cerca element

Valida si trobat i esborra

Retorna true

Si no el troba retorna false

Mètodes de la classe `forward_list`

- **`insert_after`:** afegeix un element a la posició següent a la que indica un *iterador*. Retorna un *iterador* a l'element afegit.
- **`erase_after`:** elimina l'element següent a la posició que indica un iterador. Retorna un *iterador* a l'element següent a l'element eliminat.
- **`push_front`**: afegeix un element al principi de la llista.
- **`pop_front`**: elimina el primer element de la llista.
- **`empty`:** retorna si la llista està buida.
- **`begin`**: retorna un *iterador* al primer element de la llista.
- **`before_begin`**: retorna un *iterador* a la posició anterior al primer element.
- **`end`:** retorna un *iterador* a la posició següent al final de la llista.

Classe `list` (llista doblement encadenada)

```
#include <list>

std::list<tipus> llista;
```

Mètodes:

- **`insert`:** afegeix un element a la posició **anterior** a la que indica un *iterador*. Retorna un *iterador* a l'element afegit.
- **`erase`:** elimina l'element de la posició que indica un iterador. Retorna un *iterador* a l'element següent a l'element eliminat.
- **`push_front/push_back`**: afegeix un element al principi/final de la llista.
- **`pop_front/pop_back`**: elimina el primer/últim element de la llista.
- **`empty`:** retorna si la llista està buida.
- **`begin/rbegin`**: retorna un *iterador* al primer/últim element de la llista.
- **`end/rend`:** retorna un *iterador* a la posició següent al final de la llista (end) o a la posició anterior al principi de la llista (rend).

```
// Inclusió fitxers llibreria
#include <fstream>
using namespace std;

ofstream fitxer;
// Variable per guardar el nom del fitxer
string nomFitxer = "nomDelFitxer.txt"
fitxer.open (nomFitxer);

if (fitxer.is_open())
{
    int numero;
    while (<hi_ha_valors_per_escriure>)
    {
        <obtenir_numero>

        fitxer << numero;
    }

    fitxer.close();
}
```

```
// Inclusió fitxers llibreria
#include <fstream>
using namespace std;

ifstream fitxer;
// Variable per guardar el nom del fitxer
string nomFitxer = "nomDelFitxer.txt"
fitxer.open (nomFitxer);

if (fitxer.is_open())
{
    int numero;
    fitxer >> numero;

    while (!fitxer.eof())
    {
        <tractar_numero>
        fitxer >> numero;
    }
    fitxer.close();
}
```

Mètodes de la classe stack:

- `void push (const <tipus_pila>& valor)`
  - Afegeix un element a dalt de tot de la pila
- `void pop()`
  - Elimina l'element de dalt de tot de la pila
  - La pila no pot estar buida
- `<tipus_pila>& top()`
  - Retorna l'element de dalt de tot de la pila
  - La pila no pot estar buida
- `bool empty()`
  - Retorna cert si la pila es buida, fals si no ho és

Operacions habituals de les piles:

- **`push`**: afegeix un element a dalt de tot de la pila.
- **`pop:`** elimina l'element de dalt de tot de la pila.
- **`top:`** retorna l'element de dalt de tot de la pila.
- **`esBuida:`** retorna cert si la pila es buida, fals si no ho és.

Mètodes de la classe queue:

- `void push (const <tipus_cua>& valor)`
  - Afegeix un element al final de tot de la cua com a últim element
- `void pop()`
  - Elimina el primer element de la cua
  - Suposa que la cua no està buida
- `<tipus_cua>& front()`
  - Retorna el primer element de la cua (sense eliminar-lo)
  - Suposa que la cua no està buida
- `<tipus_cua>& back()`
  - Retorna l'últim element de la cua (sense eliminar-lo)
  - Suposa que la cua no està buida
- `bool empty()`
  - Retorna cert si la cua es buida

```
std::unordered_map<string, string>::const_iterator itParaula=diccionari.find("caleidoscopi");
if (itParaula == diccionari.end())
        cout << "Definicio de caleidoscopi no hi es" <<endl;
Else    cout << "Definicio de caleidoscopi es: " <<itParaula->second <<endl;
```

ordered_map

```
map<std::string, std::string> diccionari =
una salutacio." },{"Adeu","Es un comiat."},
n periode de 24 hores."},
n periode de entre 28 i 31 dies."},
n periode de 12 mesos."}
};
string v[5] = { "Any","Hola","Mes","Dia","Adeu" };

cout << "=====CERQUES AL DICCIONARI=====" << endl;
for (int i=0;i<5;i++)
{  cout << "Definicio de " << v[i] << " es: "<< diccionari[v[i]] << endl;}

cout << "=====CONTINGUT DEL DICCIONARI=====" << endl;
for (auto& x : diccionari)
{  cout << x.first << ": " << x.second << std::endl;}
cout << "Definicio de caleidoscopi es: " << diccionari["caleidoscopi"] <<endl;

cout << "=====CONTINGUT DEL DICCIONARI=====" << endl;
for (auto& x : diccionari)
{ cout << x.first << ": " << x.second << std::endl;}
return 0;
}
```

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