# Laboratorio 5 - Tablas de Hash

Estructuras de Datos Avanzadas
Universidad Nacional de San Agustín
Nombre: Alexander Rusvell Apaza Torres

1. Código Fuente:
   a) Clase hashNode: Esta clase será el nodo de la tabla Hash, tendrá la clave y una lista en la que se almacenarán varias palabras si hay colisión.

```
class hashNode
{
    private:
        int key;
        list<string> values;
    public:
        hashNode(int key, string value){
            this->key = key;
            this->values.push_back(value);
        }
        bool operator == ( const hashNode& h2){
            return this->key == h2.key;
        }
        bool operator == ( const int& h2){
            return this->key == h2;
        }
        template <int N> friend class Hash;
};
```

   b) Clase Hash: Esta clase tendrá un vector de punteros a hashNodes, y se le pasará por templates el tamaño de este vector.

```
template< int N>
class Hash
{
    private:
        vector < hashNode* > table;
    public:
        Hash(){
            this->table.assign(N,0);
        }

        bool get(string palabra){
            boost::algorithm::to_lower(palabra);
            int llave = dispFunction(palabra,N);
            if (table[llave]){
                auto it = find(table[llave]->values.begin(),
                               table[llave]->values.end(),
                               palabra);
                return it != table[llave]->values.end();
            }
            return false;
        }
        void fill (string archivo){
            string frase;
            fstream file;
            file.open(archivo);
            string line;
            while (!file.eof()){
                getline(file,line);
                this->put(line);
            }
        }
        void put(string key){
            boost::algorithm::to_lower(key);
            int llave = dispFunction(key,N);
            if (table[llave])
                if (!this->get(key))
                    table[llave]->values.push_back(key);
            else
                table[llave]=new hashNode(llave,key);
        }
};
```

c) Función de disipación: La siguiente función toma un string como parametro y lo transforma en un número. Esto lo hace sacando cada uno de los carácteres, transformandolos en su forma ASCII y multiplicandolos por un peso obtenido según la posición del carácter. También es posible pasar un parámetro extra por si tenemos un tamaño definido en nuestro vector de hashNodes.

```
int dispFunction(string value, int modulo = 0){
    int res = 0;
    int i=0;
    unsigned char a;
    while(value[i]!='\0'){
        a = value[i];
        res = res + int(a) * i+1;
        i++;
    }
    if (modulo) return res%modulo;
    else return res;
}
```

2. Resultado: El programa usará el método **fill** de la tabla hash pasandole un archivo de texto para que este lo lea y llene el vector de hashNodes. Luego nos pedirá ingresar una oración y la separará según los espacios y preguntará si cada palabra ya existe en nuestra tabla Hash usando el método **get** y nos lo mostrará en pantalla.

```
int main(int argc, char const *argv[])
{
    Hash<1500> nHash;
    nHash.fill("palabras/listado-general.txt");

    string query;
    cout<<"Ingrese una oración"<<endl;
    getline(cin,query);

    vector<string> palabras;

    string palabra;
    string lim = " ";

    size_t pos = 0;
    while ((pos = query.find(lim)) != std::string::npos) {
        palabra = query.substr(0, pos);
        palabras.push_back(palabra);
        query.erase(0, pos + lim.length());
    }
    palabras.push_back(query);
    cout<<endl;

    for(auto i:palabras){
        if (nHash.get(i)) cout<< i <<" esta bien escrita"<<endl;
        else cout<<i<<" no esta bien escrita"<<endl;
    }

    return 0;
}
```

Resultado en consola:

```
Ingrese una oración
yo soy la contraparte

yo esta bien escrita
soy esta bien escrita
la esta bien escrita
contraparte esta bien escrita
```