```
class Name_Value_Pair {
public:
    char* Name;
    int Value;
};

class Table {
public:
    //overloaded constructors
    Table(int);    //use the int for the number of symbols
    Table();    //use this constructor for an Op Table
    int Size();
    bool Is_In_Table(char*);
    int Get_Value(char*);
    void Put_In_Table(char*, int);
    int Update_Values(int);
    void Put_Table(ofstream&);
    void Put_Literals(ofstream&, int);
private:
    Name_Value_Pair* nvp;
    int table_size;
    int number_of_items;
};

//Public Operations

Table::Table(int number_of_pairs)
{
    nvp = new Name_Value_Pair[number_of_pairs];
    table_size = number_of_pairs;
    number_of_items = 0;
}

Table::Table()
{
    table_size = 22;
    number_of_items = 22;
    nvp = new Name_Value_Pair[22];

    nvp[0].Name = "LD";
    nvp[0].Value = 0;

    nvp[1].Name = "LDI";
    nvp[1].Value = 1;

    nvp[2].Name = "ST";
    nvp[2].Value = 2;

    nvp[3].Name = "ADD";
    nvp[3].Value =  3;

    nvp[4].Name = "SUB";
    nvp[4].Value =  4;

    nvp[5].Name = "MUL";
    nvp[5].Value =  5;

    nvp[6].Name = "DIV";
    nvp[6].Value =  6;

    nvp[7].Name = "OR";
    nvp[7].Value =  7;

    nvp[8].Name = "AND";
    nvp[8].Value =  8;

    nvp[9].Name = "SHL";
    nvp[9].Value =  9;

    nvp[10].Name = "SHR";
    nvp[10].Value =  10;

    nvp[11].Name = "IO";
    nvp[11].Value =  11;

    nvp[12].Name = "BR";
    nvp[12].Value =  12;

    nvp[13].Name = "BRZ";
    nvp[13].Value =  13;

    nvp[14].Name = "BRN";
    nvp[14].Value =  14;

    nvp[15].Name = "BRS";
    nvp[15].Value =  15;

    nvp[16].Name = "ORI";
    nvp[16].Value =  16;

    nvp[17].Name = "END";
    nvp[17].Value =  17;

    nvp[18].Name = "EQU";
    nvp[18].Value =  18;

    nvp[19].Name = "NMD";
    nvp[19].Value =  19;

    nvp[20].Name = "CCD";
    nvp[20].Value =  20;

    nvp[21].Name = "RES";
    nvp[21].Value = 21;
}

int Table::Size()
{
    return number_of_items;
}

bool Table::Is_In_Table(char* name)
{
    int n = 0;
    while (n < number_of_items)
    {
    if (strcmp(name, nvp[n].Name) == 0)
    {
        return true;
    }
    n++;
    }
    return false;
}

int Table::Get_Value(char* name)
{
    int n = 0;
    while (n < number_of_items)
    {
    if (strcmp(name, nvp[n].Name) == 0)
    {
        return nvp[n].Value;
    }
    n++;
    }
}

void Table::Put_In_Table(char* name, int value)
{
    nvp[number_of_items].Name = new char[strlen(name)];
    strcpy(nvp[number_of_items].Name, name);
    nvp[number_of_items].Value = value;
    number_of_items++;
}

int Table::Update_Values(int new_value)
{
    int n = 0;
    while (n < number_of_items)
    {
    nvp[n].Value += new_value;
    new_value++;
    n++;
    }
    return new_value;
}

void Table::Put_Table(ofstream& outs)
{
    int n = 0;
    while (n < number_of_items)
    {
    outs <<"\nName: " << nvp[n].Name;
    outs <<"\nValue:" << hex <<  nvp[n].Value << '\n';
    n++;
    }
}

void Table::Put_Literals(ofstream& out, int loc)
{
    int n = 0;
    while (n < number_of_items)
    {
    out << 'T';
    out.width(2);
    out.fill('0');
    out << hex << loc;
    out.width(5);
    out.fill('0');
    char* token;
    token = strtok(nvp[n].Name, "=");
    out << hex << atoi(token);
    out << '\n';
    n++;
    loc++;
    }
}
```