```
int manhattan_linear(char* state)
 {
	//compute manhattan distance
	int total = 0;
	total = manhattan(state);

	int row;
	vector<int> goal_v;

	//for each row determine the linear conflicts
	for (row = 0; row <X; row++)
	{
		goal_v.clear();

		//determine elements in correct row
		for (int i=0;i<X;i++)
		{
			for (int j=0;j<X;j++)
			{
				if (state[i+row*3] == goal.s[j+row*3])
				{	if(int(state[i+row*3]) !=0 )
						goal_v.push_back(goal.s[j+row*3]);

				}
			}
		}

		if (goal_v.size()>1)
		{
			vector<int> goal_pos;

			//evaluate goal positions for each element in correct row
			for (int i=0; i<int(goal_v.size()); i++)
			{
				for (int z= row*3; z<(row+1)*3; z++)
				{
					if (goal_v[i] == goal.s[z])
						{
							goal_pos.push_back(z);
						}
				}
			}

		//determine if elements need to pass each other to reach goal, if yes add 2
			for(int i=0;i<int(goal_pos.size());i++)
			{
				for(int j=i+1;j<int(goal_pos.size());j++)
				{
					if (goal_pos[i]>goal_pos[j])
					{

						total += 2;
						break;
					}
				}
			}

		}
```

```
	}

	int col;

	//for each column determine the linear conflicts
	for (col = 0; col <Y; col++)
	{
		goal_v.clear();

		//determine elements in correct column
		for (int i=0;i<X;i++)
		{
			for (int j=0;j<X;j++)
			{
				if (state[3*i+col] == goal.s[3*j+col])
				{	if(int(state[3*i+col]) != 0)
					goal_v.push_back(goal.s[3*j+col]);
				}
			}
		}

		if (goal_v.size()>1)
		{
			vector<int> goal_pos2;

			//evaluate goal positions for each element in correct column
			for (int i=0; i<int(goal_v.size()); i++)
			{
				for (int z=col; z<=(X-1)*Y+col; z+=3)
				{
					if (goal_v[i] == goal.s[z])
						{
							goal_pos2.push_back(z);
						}
				}
			}

		//determine if elements need to pass each other to reach goal, if yes add 2
			for(int i=0;i<int(goal_pos2.size());i++)
			{
				for(int j=i+1;j<int(goal_pos2.size());j++)
				{
					if (goal_pos2[i]>goal_pos2[j])
					{
						total += 2;
						break;
					}
				}
			}

		}

	}


	return total;
}
```