```d
struct vec2{
        int x;
        int y;
}
struct vec3{
        int x;
        int y;
        int z;
}

//------------

//mixin monkeydata!(monkeytypes: vec3,vec2)

struct vec2pointy_{
        import voidarrays;
        mypointer!4 x;
        mypointer!4 y;
        void opUnaray(string op:"++"){
                ++x;
                ++y;
        }
        void opUnarry(string op:"--"){
                --x;
                --y;
        }
        void opBinary(string op:"+")(int a){
                x+a;
                y+a;
        }
        void set(T)(ref T setter){
                valid!T;
                x.set!(T.x)(setter.x);//assuming T is vec2, will need to mixin in a meta
mixin here
                y.set!(T.y)(setter.y);
        }
        T get(T)(){
                valid!T;
                T foo=void;
                foo.x=x.get!T.x;
                foo.y=x.get!T.y;
                return foo;
        }
        void valid(T)(){
                static assert(T.x.sizeof <= 4);
                static assert(T.y.sizeof <= 4);
        }
}

struct vec3pointy_{
        import voidarrays;
        mypointer!4 x;
        mypointer!4 y;
        mypointer!4 z;
        void opUnaray(string op:"++"){
                ++x;
                ++y;
                ++z;
        }
        void opUnarry(string op:"--"){
                --x;
                --y;
                --z;
        }
        void opBinary(string op:"+")(int a){
                x+a;
                y+a;
                z+a;
        }
        void set(T)(T setter){
```

```
                valid!T;
                x.set!(T.x)(setter.x);
                y.set!(T.y)(setter.y);
                z.set!(T.z)(setter.z);
        }
        T get(T)(){
                valid!T;
                return T(x.get!(T.x),y.get!(T.y),z.get!(T.z));
        }
        void valid(T)(){
                static assert(T.x.sizeof <= 4);
                static assert(T.y.sizeof <= 4);
                static assert(T.z.sizeof <= 4);
        }
}

struct vec2pointy{
        import std.traits;
        import monkeytyping;
        enum ispointy=true;
        alias tolitteral=tovec2;
        alias mylitteral=vec2;
        vec2pointy_ grey;

        //mixin op_nonsense!1;

        auto ref opBinary(string op,T)(auto ref T a) if (hasMember(T,"ispointy"){
                return operation!op(tolitteral,a.tolitteral);
        }
        auto ref opBinary(string op)(auto ref T a) if (!hasMember(T,"ispointy"){
                return operation!op(tolitteral, a);
        }
        ref typeof(this) opAssign(T a) if (hasMember(T,"ispointy"){
                this = a.tolitteral;
        }
        ref typeof(this) opAssign(T a) if (!hasMember(T,"ispointy"){
                static if(issubtype!(mylitteral,T){
                        static foreach(foo;definitions!T){
                                mixin(foo.name,"= a.",foo.name,";");
                }} else { static assert(issubtype!(T,mylitteral),"These types airnt compa
dible")
                        static foreach(foo;definitions!mylitteral){
                                mixin(foo.name,"= a.",foo.name,";");
        }}}
        ref typeof(this) opOpAssign(auto ref T a){
                mixin("this = this",op,"a;");
                return this;
        }
        ref typeof(this) opUnarry(string op:"++"){
                ++grey;}
        ref typeof(this) opUnarry(string op:"--"){
                --grey;}

        void movepointer(int x){
                grey + x;}

        vec2pointy tovec2pointy{// my subtype system returns duplicates so I'm rolling wi
th it
                return vec2pointy(grey.x,grey.y);}
        vec2 tovec2{
                return grey.get!vec2;}

        this(ref vec2 construct){
                grey.x=&construct.x;
                grey.y=&construct,y;
        }
        this(vec2.x* x_,vec2.y* y_){
                grey.x=x_;
                grey.y=y_;
        }
```

```
        void setpointers(T)(T litteral){
                import monkeytyping;
                static assert(issubtype!(vec2,T));
                static foreach(def, definitions!T){
                        //grey.x=&litteral.x;
                        //grey.y=&litteral.y;
        }}
}

struct vec3pointy{
        vec3pointy_ grey;

        //mixin op_nonsense!0;

        vec2pointy tovec2pointy(){
                return vec2pointy(grey.x,grey.y);}
        vec2 tovec2(){
                return grey.get!vec2;}
        vec3pointy tovec3pointy(){
                return vec3pointy(grey.x,grey.y,grey.z);}
        vec3 tovec3(){
                return grey.get!vec3;}

        this(vec3 construct){
                grey.x=&construct.x;
                grey.y=&construct,y;
                grey.z=&construct.z;
        }
        this(vec3.x* x_,vec3.y* y_,vec3.z z_){
                grey.x=x_;
                grey.y=y_;
                grey.z=z_;
        }
        void setpointers(){}
}

struct vec2soa_(size_t n){
        import voidarray;
        voidarray!(4,n) x;
        voidarray!(4,n) y;
        vec2pointy opIndex(size_t i){
                return vec2pointy(x[i],y[i]);
        }
        size_t opDollar(){ return n-1;}
}
struct vec3soa_(size_t n){
        import voidarray;
        voidarray!(4,n) x;
        voidarray!(4,n) y;
        voidarray!(4,n) z;
        vec2pointy opIndex(size_t i){
                return vec2pointy(x[i],y[i],z[i]);
        }
        size_t opDollar(){ return n-1;}
}
struct vec2soaslice{
        /*immutable?*/ size_t start__;
        vec2pointy start;
        /*immutable?*/ size_t end__;
        vec2pointy end;
        vec2pointy front(){return start;}
        void popFront(){start++;}
        bool empty(){return start > end;}
}
struct vec3soaslice(){}

struct vec2aosoaslice(bool expanding,size_t soa=512){
        vec2aosoa!soa* parent;
        size_t start;
        static if(expanding){
```

```
                size_t end(){return(*parent).count;}}
        else{
                size_t end;}
        import lazynullable;
        nullable!vec2soaslice lasthead;
        size_t segment(){
                size_t natspilt= (start/soa+1)*soa-1;
                return min(natspilt,end);
        }
        vec2soaslice front(){
                if(lasthead.isnull){
                        auto seg=segment;
                        lasthead=vec2soaslice(start,(*parent)[start],seg,(*parent)[seg]);
                }
                return lasthead;
        }
        void popFront(){
                start=lasthead.end__;
                start++;
                lasthead.isnull=true;
        }
        bool empty(){return start>end;}
}

struct vec2aosoa(size_t soa=512){
        vec2soa_!soa[] chunks;
        size_t count;
        dollar opDollar(){return dollar;}
        vec2pointy opIndex(size_t i){
                assert(i<count,"accessing random data is frowned on, use [0..i]
                                if you intended to create i`th T, or [$..i] if you intend
ed
                                to make i elements");
                return vec2pointy((chunks[i/soa])[i%soa]);
        }
        vec2pointy opIndex(dollar i){[count-1];}
        vec2aosoaslice!(true,soa) opSlice(){return [0..$];}
        vec2aosoaslice!(false,soa) opSlice(size_t i,size_t j){

        }
        vec2aosoaslice!(true,soa) opSlice(size_t i,dollar j){

        }
        vec2aosoaslice!(false,soa) opSlice(dollar i,size_t i){

        }
        void expand(size_t i){

        };
        alias [] this;
}
```