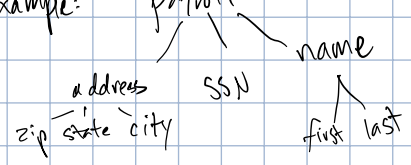
terminal vars
code output

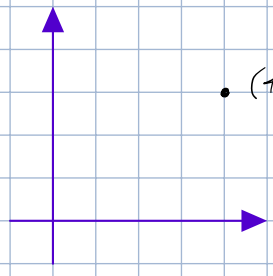
Structure — collection of 1 or more variables grouped together under single name for convenient handling
possibly of different types

help organize complicated data

for example: payroll



Basics of Structures



Let's make a point

```
struct point {
    int x;
    int y;
};
```

same member names can be used for different structs

```
struct point pt
struct point maxpt = {320, 200};
```

struct-name.member

Structures and Functions

legal operations on a structure: copying it or assigning to it as a unit, taking its address w/ & & accessing its members

structures can't be compared

3 ways to use structures w/ fn's: pass components seperately, pass entire structure, pass pointer to it

```
struct point makepoint(int x, int y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

can be used to initialize structures dynamically

```
struct point addpoint(struct point p1, struct point p2)
    p1.x += p2.x
    p1.y += p2.y
    return p1
```

→ works b/c fn is call by value not reference

```
struct point *pp
```

pp is pointer to structure of type struct point

(*pp).x & (*pp).y → its members

if p is a pointer to a structure

p → member-of-structure

pp → x

say r is a rectangle structure. all 4 are equivalent & *rp = &r

```
r.pt1.x
rp -> pt1.x
(r.pt1).x
(rp -> pt1).x
```

Arrays of Structures

```
struct key {
    char *word;
    int count;
} keytab[nkeys];
```

declare a structure type key

define an array keytab of structures of this type

sets storage

same

```
struct key {
    char *word;
    int count;
};
struct key keytab[nkeys];
```

can initialize array too

```
struct key {
    char *word;
    int count;
} keytab[ ] = {
    {"word1", 0},
```

{"word2", 0},
{"word3", 0},
};

size of struct key * # of entries = size of keytab

in C, can find integer equal to size of object/type in bytes
sizeof object — var, array, structure
sizeof (type name) — basic type, pointer, structure

Pointers to Structures

struct key *binsearch

don't generate an illegal pointer, make sure to not go beyond limits [-1] [max]

pointer arithmetic deals w/ lots of sizeof() problems

Typedef

can create new data types

typedef int Length — now Length is a synonym for int

can be used in same way
Length len, maxlen;
Length *lengths[];

typedef char *String

does not create a new type, adds new name for existing types

why?
- ☺ Parameterize program against portability problems
- ☺ Better documentation

Unions

union — variable that may hold objects of different types & sizes
compiler keeps track of size & alignment requirements

way to manipulate different kinds of data in single area of storage w/o machine dependent info.

example: table manager
value of a particular constant must be stored in a var, table helps if value occupies same amt. of storage & stored in same place

union u_tag {
int ival;
float fval;

```
    char *sval
} u ;
```
→ variable u will be large enough to hold largest of 3 type

any type may be assigned to u & used, so long as usage is consistent → type received must be type most recently stored

have to keep track which type is currently stored in a union

access a member of a union

union-name . member

union-pointer -> member

unions can occur within structs & arrays . vice versa

example:

```
struct {
    char  *name
    int   flags
    int   utype
    union {
        int   ival
        float fval
        char  *sval
    } u ;
} symtab[NSYM] ;
```

member ival
→ symtab[i].u.ival

first char of sval
→ symtab[i].u.sval[0]
→ *symtab[i].u.sval

union is a struct in which all members offset zero from base

struct is big enough to hold largest member

Self-referential Structure

binary tree!

node has a value, left child, & right child . can have 1 or 2 childs

```
struct tnode {
    char  *word ;
    int   count ;
    struct tnode *left ;
    struct tnode *right ;
} ;
```

it is illegal for a structure to contain instance of itself
→ here, we're declaring a pointer to new declaration

leaves have a null value

use malloc?