CS113: Lecture 7

Topics:

- typedef, struct
- Introduction to Pointers
Create your own types: typedef

#define N 3

typedef double scalar; /* note defs outside fns */
typedef scalar vector[N];

typedef scalar matrix[N][N];
    /* alternatively:
       typedef vector MATRIX[N]; */

/* add(x,y,z) adds the vectors y and z, placing the result in x */
void add( vector x, vector y, vector z )
{
    int i;
    for( i = 0; i < N; i++ )
    {
        x[i] = y[x] + z[i];
    }
}
Structures

- The structure mechanism allows us to aggregate variables of different types

- struct definitions generally go outside of the functions, as in the following example

```c
struct card_struct
{
    int pips;
    char suit;
};

typedef struct card_struct card;

void some_function()
{
    struct card_struct a;
    card b; /* a, b have the same type */

    b.pips = 3;
    b.suit = 'D';
    a = b;
}
```
struct example: points in the plane

#include <math.h>

struct point_struct
{
    double x;
    double y;
};

typedef struct point_struct point;

double distance( point p1, point p2 )
{
    double dx, dy, dist;
    dx = p1.x - p2.x;
    dy = p1.y - p2.y;
    dist = sqrt((dx * dx) + (dy * dy));
    return( dist );
}

void main()
{
    point a = { 3.5, 4.5 };
    point b = { 6.5, 0.5 };
    printf( "distance: %f\n", distance( a, b ) );
}
Introduction to Pointers

- A variable in a program is stored in a certain number of bytes at a particular memory location, or address, in the machine.

- Pointers allow us to manipulate these addresses explicitly.

- Two unary operators ("inverses"):
  - & operator - "address of" operator. Can be applied to any variable. "Adds a star" to the type.
  - * operator - "dereference" operator. Can be applied only to pointers. Accesses the object that the pointer points to. "Removes a star" from type.

- Example.

```c
int x = 1, y = 2;
int *ip;
char *cp;

ip = &x;    /* ip now points to x */
y = *ip;    /* y is now 1 */
*ip = 0;    /* x is now 0 */
```
An example

void main()
{
    int x = 5, y = 6;
    int *ip;

    ip = &x;
    printf( "printf 1: %d\n", x );
    printf( "printf 2: %d\n", *ip );

    *ip = 10;
    printf( "printf 3: %d\n", x );
    printf( "printf 4: %d\n", *ip );

    (*ip)++;
    ip = &y;
    printf( "printf 5: %d\n", x );
    printf( "printf 6: %d\n", *ip );
}

Another example

```c
void f( int a, int *ip )
{
    *ip = a;
}

void main()
{
    int x = 5, y = 6;
    int *z;

    f( 10, &y );
    printf( "x: %d y: %d\n", x, y );

    f( x, &y );
    printf( "x: %d y: %d\n", x, y );

    z = &x;
    f( 8, z );
    printf( "x: %d y: %d\n", x, y );
}
```
How to swap two values?

What's wrong with this?

```c
void swap( int x, int y )
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}

void main()
{
    int a = 3, b = 5;
    swap( a, b );
    printf( "a is %d, b is %d\n", a, b );
}
```
A correct swap

void swap( int *px, int *py )
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}

void main()
{
    int a = 3, b = 5;
    swap( &a, &b );
    printf( "a is %d, b is %d\n", a, b );
}
Be careful with your new toys.

• Do not point at constants.

    int *ptr;
    ptr = &3; /* illegal */

• Do not point at expressions that are not variables.

    int k = 1, *ptr;
    ptr = &(k + 99); /* illegal */

• What's wrong with this?

    int *function_3()
    {
        int b;
        b = 3;
        return &b;
    }

    void main()
    {
        int *a;
        a = function_3();
        printf( "a is equal to %d\n", *a );
    }
An example

Good to know the right-hand rule.

```c
void main()
{
    int a, b;
    int *pc, *pd;
    int **ppe, **pf;

    a = 3;
    b = 5;
    pc = &a;
    pd = &b;
    (*pd)++;
    printf( "a: %d  b: %d\n", a, b );
    *pc += *pd;
    printf( "a: %d  b: %d\n", a, b );

    ppe = &pc;
    pf = &pd;
    *pf = pc;
    *pd = 12;
    printf( "a: %d  b: %d\n", a, b );

    **ppe = 50;
    **pf = 15;
    printf( "a: %d  b: %d\n", a, b );
}
```