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[i] + 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

```
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.

- To declare a pointer variable: add a star to the type you want to point to. Example:

  `int *a;`

  declares a variable `a` of type `int *`, which can be used to hold the address of (or a "pointer to") an `int`.

- Two unary operators ("inverses"):

  – `&` operator - "address of" operator. Can be applied to any variable. Type of resulting expression has "one more star" than original expression.

  – `*` operator - "dereference" operator. Can be applied only to pointers. Accesses the object that the pointer points to. Type of resulting expression has "one less star" than original expression.

- Don't confuse the `*` operator with the `*` in the declaration of a variable (nor with multiplication).

Pointers: Example

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

ip = &x;    /* ip now points to x */
printf( "%d\n", *ip );     /* 1 */
printf( "%d\n", *ip + 2 ); /* 3 */
y = *ip;    /* y is now 1 */
*ip = 0;    /* x is now 0 */

printf( "%d\n", x );       /* 0 */

cp = &x;    /* doesn't work; types don't match */
*cp = 'z';  /* what happens? */
cp = &c;
*cp = 'z';
printf( "%c\n", c );       /* z */
```

printf vs. scanf

```
void main()
{
    int k;
    printf( "Enter an integer: " );
    scanf( "%d", &k );
    printf( "%d", k );
}
```

Also works:

```
void main()
{
    int k, *pk;
    pk = &k;
    printf( "Enter an integer: " );
    scanf( "%d", pk );
    printf( "%d", k );
}
```

Who wants what information?

- `printf( "%d", ... );` expects an `int`, since it needs to know *what* to print out

- `scanf( "%d", ... );` expects the *address* of an `int`, since it needs to know *where* to place the int typed in

  – `scanf` doesn't care about the actual value of the `int` that it should write to

More practice

```
void main()
{
  int a = 3, b = 3;
  int *pa, *pb;

  pa = &a;
  pb = &b;

  if( pa == pb )
    printf( "pa and pb are equal.\n" );
  if( *pa == *pb )
    printf( "*pa and *pb are equal.\n" );

  (*pa)++; /* careful: different from *pa++ */
  *pb += *pa;
  printf( "a: %d, b: %d\n", a, b );

  pb = pa;
  *pa += *pb;
  printf( "a: %d, b: %d\n", a, b );
  if( pa == pb )
    printf( "pa and pb are equal.\n" );
  if( *pa == *pb )
    printf( "*pa and *pb are equal.\n" );
}
```

How to swap two values?

What's wrong with this?

```
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 */
```

- Do not try to dereference non-pointer variables.

```
int k;
printf( "%d", *k ); /* 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*.

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

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

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