Imperative programming

11th Lecture

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

1. Pointers and Arrays in C
   - Passing Arrays as Parameters
   - Higher-order Functions
   - Constants in C
2. Type Constructs
   - Enumeration Types
   - Compound Value Types

Differences between Pointers and Arrays

```
int v[] = {6, 3, 7, 2};
int *p = v;

v[ 1 ] = 5;
p[ 1 ] = 8;

int w[] = {1,2,3};
p = w;  /* ok */
v = w;  /* compilation error */

printf( "%d %d\n", sizeof( v ), sizeof( p ) );
```

See last example here:
`http://gsd.web.elte.hu/lectures/imper/imper-lecture-8/`.

Passing Arrays as Parameters: Generalization?

```
double distance( double a[3], double b[3] ){
   double sum = 0.0;
   int i;
   for( i=0; i<3; ++i ){      /* hard-coded value :-( */
      delta = a[i] - b[i];
      sum += delta*delta;
   }
   return sqrt( sum );
}

int main(){
   double p[3] = {36, 8, 3}, q[3] = {0, 0, 0};
   printf( "%f\n", distance(p,q) );
   return 0;
}
```

Passing Arrays as Parameters: Size Fixed at Compile Time

```
#define DIMENSION 3

double distance( double a[DIMENSION], double b[DIMENSION] ){
   double sum = 0.0;
   int i;
   for( i=0; i<DIMENSION; ++i ){
      delta = a[i] - b[i];
      sum += delta*delta;
   }
   return sqrt( sum );
}

int main(){
   double p[DIMENSION] = {36, 8, 3}, q[DIMENSION] = {0, 0, 0};
   printf( "%f\n", distance(p,q) );
   return 0;
}
```

Passing Arrays as Parameters: Size Fixed at Runtime?

```
double distance( double a[], double b[] ){
   double sum = 0.0;
   int i;
   for( i=0; i<???; ++i ){    /* this is not Python */
      delta = a[i] - b[i];
      sum += delta*delta;
   }
   return sqrt( sum );
}

int main(){
   double p[] = {3.0, 4.0}, q[] = {0.0, 0.0};
   printf( "%f\n", distance(p,q) );
   return 0;
}
```

Passing Arrays as Parameters: Erroneous Approach

```
double distance( double a[], double b[] ){
   double sum = 0.0;
   int i;
   for( i=0; i<sizeof(a)/sizeof(a[0]); ++i ){
      delta = a[i] - b[i];
      sum += delta*delta;
   }
   return sqrt( sum );
}

int main(){
   double p[] = {3.0, 4.0}, q[] = {0.0, 0.0};
   printf( "%f\n", distance(p,q) );
   return 0;
}
```

Passing Arrays as Parameters: Correct

```c
double distance( double a[], double b[], int dim ){
   double sum = 0.0;
   int i;
   for( i=0; i<dim; ++i ){
      delta = a[i] - b[i];
      sum += delta*delta;
   }
   return sqrt( sum );
}

int main(){
   double p[] = {3.0, 4.0}, q[] = {0.0, 0.0};
   printf( "%f\n", distance(p,q,sizeof(p)/sizeof(p[0])) );
   return 0;
}
```

Passing Complex Structure as Parameter

```
int main( int argc, char *argv[] ){ ... }
```

- argc: positive number
- argv[0]: name of program
- argv[i]: arguments in command line $(1 \leq i < \text{argc})$
  - array of characters, NUL ('\0') at the end
- argv[argc]: NULL

```
int main( void ){ ... }
int main( int argc, char *argv[], char *envp[] ){ ... }
int main(){ ... }
```

Multidimensional Arrays as Parameters

```
double m[4][4] = {{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}};

transpose(m);

{
   int i,j;
   for( i=0; i<4; ++i ){
      for( j=0; j<4; ++j ){
         printf("%3.0f", m[i][j]);
      }
      printf("\n");
   }
}
```

Solution is Too Rigid

```
void transpose( double matrix[4][4] ){  /* double matrix[][4] */
   int size = sizeof(matrix[0])/sizeof(matrix[0][0]);
   int i, j;
   for( i=1; i<size; ++i ){
      for( j=0; j<i; ++j ){
         double tmp = matrix[i][j];
         matrix[i][j] = matrix[j][i];
         matrix[j][i] = tmp;
      }
   }
}

double m[4][4] = {{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}};
transpose(m);
```

Continuous Representation: Continuous Memory Area

```
void transpose( double *matrix, int size ){  /* size*size double */
   int i, j;
   for( i=1; i<size; ++i ){
      for( j=0; j<i; ++j ){
         int idx1 = i*size+j,  /* instead of matrix[i][j] */
             idx2 = j*size+i;  /* instead of matrix[j][i] */
         double tmp = matrix[idx1];
         matrix[idx1] = matrix[idx2];
         matrix[idx2] = tmp;
      }
   }
}

double m[4][4] = {{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}};
transpose( &m[0][0], 4 );  /* transpose( (double*)m, 4 ) */
```

Alternative Representation: Array of Pointers

```
void transpose( double *matrix[], int size ){
   int i, j;
   for( i=1; i<size; ++i ){
      for( j=0; j<i; ++j ){
         double tmp = matrix[i][j];
         matrix[i][j] = matrix[j][i];
         matrix[j][i] = tmp;
      }
   }
}

double m[4][4] = {{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}};
double *helper[4]; for( i=0; i<4; ++i ) helper[i] = m[i];
transpose(helper,4);
```

Higher-order Functions

C: using Function Pointers

```
/* pointer to a function without parameter and with int result */
int (*fp)(void);

/* a function expecting int->int function and int, with int result */
int twice( int (*f)(int), int n );
```

C: Function Pointers

```
int twice( int (*f)(int), int n )
{
    n = (*f)(n);
    n = f(n);
    return n;
}

int inc( int n ){ return n+1; }


printf(  "%d\n",  twice( &inc, 5 )  );
```

C: Function Pointers - Some Remarks

```
int inc( int n ){ return n+1; }

int (*f)(int) = &inc;
f = inc;
f(3) + (*f)(3);

int (*g)() = inc;
g(3,4);  g();
```

Constants in C

Constants

```c
const int i = 3;
int const j = 3;

const int t[] = {1,2,3};


const int *p = &i;

int v = 3;
int * const q = &v;
```

Erroneous Usage

```c
i = 4;
j = 4;

t[2] = 4;
t = {1,2,4};

*p = 4;


q = (int *)malloc(sizeof(int));
```

Not fully safe

```
const int i = 3;
int * const q = &i;     /* warning only */
int * p = &i;           /* warning only */

*p = *q = 4;            /* i changes */
```

polymorph solution on const is not possible

```
char *strchr( const char *str, int c ){
    while( *str != 0 && *str != c ) ++str;
    return str;
}

char *p = strchr("Hello",'e'),      q[] = "Hello";
*p = 'o';    /* error! */           char *r = strchr(q,'e');
                                    *r = 'o';   /* ok */
```

Problems on Lifetime

`http://gsd.web.elte.hu/lectures/imper/imper-lecture-5/`

(at the end)

Outline

1. Pointers and Arrays in C
    - Passing Arrays as Parameters
    - Higher-order Functions
    - Constants in C

2. Type Constructs
    - Enumeration Types
    - Compound Value Types

Type Constructs

- Enumeration Types
- Pointer Types
- Compound Types

Enumeration Types

Haskell

```
data Color = White | Green | Yellow | Red | Black
```

in C: is just a whole number

```
enum color { WHITE, GREEN, YELLOW, RED, BLACK };

const char* property( enum color code ){
    switch( code ){
        case WHITE:  return "clean";
        case GREEN:  return "jealous";
        case YELLOW: return "envy";
        case RED:    return "angry";
        case BLACK:  return "sad";
        default:     return "?";
    }
}
```

Enumeration Type in C

```
enum color { WHITE = 1, GREEN, YELLOW, RED = 6, BLACK };

typedef enum color Color;

const char* property( Color code ){ ... }

int main( int argc, char *argv[] )
{
    for( --argc; argc>0; --argc )
    {
       printf("%s\n", property( atoi(argv[argc]) ));
    }
    return 0;
}
```

Compound Value Types

- Sequence
- Set
- Map
- Cartesian Product
- Union
- Class

Sequence Types

- C Arrays
- Python Lists and Tuples
- Haskell Lists

Sequence: compound type of elements of the same type

Cartesian Product Types

Type constructed of elements of (potentially) different types.

- tuple
- record
- struct

C struct

```
struct month { char *name, int days };   /* type creation */

struct month jan = {"January", 31};      /* variable creation */

/* three-way comparison */
int compare_days_of_month( struct month left, struct month right )
{
    return left.days - right.days;
}
```

C struct

```
struct month { char *name; int days; };
struct month jan = {"January", 31};

struct date { int year; struct month *month; char day; };
struct person { char *name; struct date birthdate; };

typedef struct person Person;

int main(){
   Person pete = {"Pete", {1970,&jan,28}};
   printf("%d\n", pete.birthdate.month->days);
   return 0;
}
```

Parameter Passing

```
void one_day_forward( struct date *d ){
    if( d->day < d->month->days ) ++(d->day);
    else { ... }
}

struct date next_day( struct date d ){
    one_day_forward(&d);
    return d;
}

int main(){
   struct date new_year = {2019, &jan, 1};
   struct date sober;
   sober = next_day(new_year);
   return ( sober.day != 2 );
}
```