Imperative programming

9th Lecture

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Alias in C and Python

Alias: multiple names refer to the same storage space

C: Pointers

```
int xs[] = {1,2,3};
int *ys = xs;
xs[2] = 4;
printf("%d\n", ys[2]);
```

Accessed Type of Pointers

```
int *p = (int *)malloc(sizeof(int));
if( NULL != p )
{
    *p = 123;
    *p = 12.3;  /* automatic type conversion */
    printf("%d\n", *p);
    free(p);
}
```

Accessed Type of Pointers: Type Enforcement

```
float *q = (float *)malloc(sizeof(float));
if( NULL != q )
{
    int *p = (int *)q;
    *q = 12.3;
    printf("%d\n",*p);
    free(q);
}
```

Accessing Dynamic Storage

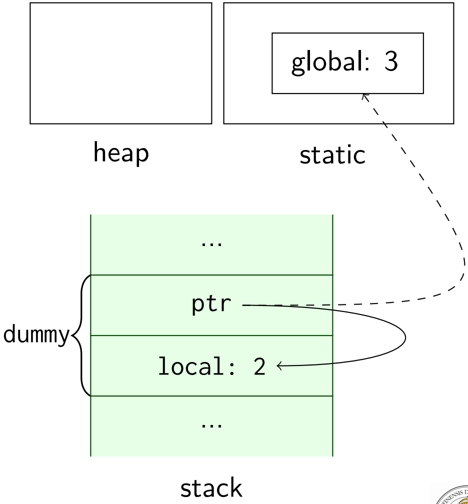
C

- Explicit (pointer)
- Static type-checking
- Strongly typed
- Deallocation

Pointer on non-dynamic variable

```
int global = 1;

void dummy(void)
{
    int local = 2;
    int *ptr;
    ptr = &global; *ptr = 2;
    ptr = &local;  *ptr = 4;
}
```



Invalid Pointer

Bogus

```c
int *make_ptr()
{

    int n = 42;
    return &n;
}
```

Correct

```c
int *make_ptr()
{
    int *ptr = (int*)malloc(sizeof(int));
    *ptr = 42;
    return ptr;
}
```

```c
        printf("%d\n", *make_ptr());
```

Outline

Function declarations and definitions

```
int f( int n );
int g( int n ){ return n+1; }

int h();
int i(void);

int j(void){ return h(1); }

int h( int p, int q ){ return p+q; }

extern int k(int,int);

int printf( const char *format, ... );
```

Techniques of Parameter Passing

- **pass-by-value**, call-by-value
- call-by-value-result – Ada
- call-by-result – Ada
- call-by-reference – Pascal, C++
- **call-by-sharing**
- call-by-need – Haskell
- call-by-name – Scala
- substitute-as-text – C-macro

Pass-by-value Parameter Passing

```
int gcd( int a, int b )
{
    int c;
    while( b != 0 ){
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
int main()
{
    int n = 1984, m = 356;
    int r = gcd(n,m);
    printf("%d %d %d\n",n,m,r);
}
```

Input Semantics

```
void swap( int a, int b )
{
    int c = a;
    a = b;
    b = c;
}

int main()
{
    int n = 1984, m = 356;
    swap(n,m);
    printf("%d %d\n",n,m);
}
```

Passing Pointer as Pass-by-value

```c
void swap( int *a, int *b )
{
    int c = *a;
    *a = *b;
    *b = c;
}
```

```c
int main()
{
    int *n, *m;
    n = (int*) malloc(sizeof(int));
    m = (int*) malloc(sizeof(int));
    if( n != NULL && m != NULL )
    {
        *n = 1984;
        *m = 356;
        swap(n,m);
        printf("%d %d\n",*n,*m);
        free(n); free(m);
        return 0;
    } else return 1;
}
```

Emulating Pass-by-reference

```c
void swap( int *a, int *b )
{
    int c = *a;
    *a = *b;
    *b = c;
}
```

```c
int main()
{
    int n = 1984, m = 356;
    swap(&n,&m);
    printf("%d %d\n",n,m);
}
```

Pass-by-reference – Pascal

```
program swapping;

   procedure swap( var a, b: integer );  (* var: Pass-by-reference *)
   var
      c: integer;
   begin
      c := a; a := b; b := c
   end;

   var n, m: integer;

begin
   n := 1984; m := 356;
   swap(n,m);
   writeln(n,' ',m)    (* 356 1984 *)
end.
```

Pass-by-reference – C++

```
#include <cstdio>

void swap( int &a, int &b )   /* &: Pass-by-reference */
{
    int c = a;
    a = b;
    b = c;
}

int main()
{
    int n = 1984, m = 356;
    swap(n,m);
    printf("%d %d\n",n,m);
}
```

Call-by-value-result – Ada

```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Swapping is

   procedure Swap( A, B: in out Integer ) is
      C: Integer := A;
   begin
      A := B; B := C;
   end Swap;

   N: Integer := 1984;
   M: Integer := 356;

begin
   Swap(N,M);
   Put(N); Put(M);  -- 356 1984
end Swapping;
```

Call-by-sharing Parameter Passing

```
void swap( int t[] )
{
    int c = t[0];
    t[0] = t[1];
    t[1] = c;
}
int main()
{
    int arr[] = {1,2};
    swap(arr);
    printf("%d %d\n",arr[0],arr[1]);
}
```

This is not call-by-reference

```
void twoone( int t[] )
{
    int arr[] = {2,1};
    t = arr;
}

int main()
{
    int arr[] = {1,2};
    twoone(arr);
    printf("%d %d\n",arr[0],arr[1]);
}
```

Returning automatic variable?

C: erroneous

```
int *twoone()
{
    int arr[] = {2,1};
    return arr;
}
```

On Demand Parameter Passing

```
f True  a _ = a
f False _ b = b + b

main = print result
  where result = f False (fact 20) (fact 10)

        fact 0 = 1
        fact n = n * fact (n-1)
```

Substitute-as-text

```
#define DOUBLE(n) 2*n
#define MAX(a,b) a>b?a:b

int main()
{
    printf("%d %d\n", MAX(10,100), DOUBLE(10));
    {
       int n = 5;
       printf("%d\n", DOUBLE(n+1));
       printf("%d\n", MAX(5,++n));
    }
}
```

Substitute-as-text – tricky

```
#define DOUBLE(n) 2*n
#define MAX(a,b) a>b?a:b

int main()
{
    printf("%d %d\n", MAX(10,100), DOUBLE(10));
    {
       int n = 5;
       printf("%d\n", DOUBLE(n+1)); /* printf("%d\n", 2*n+1); */
       printf("%d\n", MAX(5,++n));
    }
}
```

Substitute-as-text – parenthesizing

```
#define DOUBLE(n) (2*(n))
#define MAX(a,b) ((a)>(b)?(a):(b))

int main()
{
    printf("%d %d\n", MAX(10,100), DOUBLE(10));
    {
       int n = 5;
       printf("%d\n", DOUBLE(n+1)); /* (2*((n)+1)) */
       printf("%d\n", MAX(5,++n));
    }
}
```

Substitute-as-text – still dangerous

```c
#define DOUBLE(n) (2*(n))
#define MAX(a,b) ((a)>(b)?(a):(b))

int main()
{
    printf("%d %d\n", MAX(10,100), DOUBLE(10));
    {
       int n = 5;
       printf("%d\n", DOUBLE(n+1)); /* (2*((n)+1)) */
       printf("%d\n", MAX(5,++n));  /* ((5)>(++n)?(5):(++n)) */
    }
}
```