C Data Types and Functions

Alon Efrat
Computer Science Department
University of Arizona

Outline
- Data Types
- Function
- Global Variable
- Pointer
- Argument Passing in Function Calls

Basic Concepts
- Variable
- Constants
- Type
- Declaration
- Operator
- Expression

Data Types and Sizes
- Basic data type
  - char
  - int
  - float
  - double
- Qualifiers
  - short
  - long
  - unsigned
- Size is machine dependent
- sizeof()

Constants
- int: 1234
- long: 123456789L
- unsigned long: 123456789UL
- double: 123.4, 1e-2
- float: 123.4F, 1e-2F
- long double: 123456789.123L
- Octal: 0234
- Hexadecimal: 0x2B3
- char: ‘g’, ‘\007’, ‘\x7’
- String: “I am a string”, “hello” “world”
- Difference between ‘x’ and ‘X’

Enumeration Constant
- Enum boolean { NO, YES };
- Enum boolean { BELL=\a, TAB=\t};
- Type of “int”
- Names must be distinct
- An alternative of “#define” with type checking and auto-generated values
/* A demonstration of casting effect */
#include <stdio.h>
main()
{
    printf("%d \n", 2.0 );
    /*Printed value = 1073741824 */
    printf("%d \n", 3 * (2/3) );
    /*Printed value = 0 */
    printf("%d \n", 3 * (2/3.0) );
    /*Printed value = 2 */
    printf("%d \n", 3 * (2/(float)3) );
    /*Printed value = 2 */
}

**Outline**
- Data Types
- Function
- Global Variable
- Pointer
- Argument Passing in Function Calls

**Type Conversion**
- Implicit type conversion
  - Promote the “lower” type to “higher” type before the operation. The result is of “higher” type.
  - OK: Narrower \rightarrow wider
- Warning: conversion causes loss of information
- Error: Meaningless expressions
- Explicit type conversion
  - Unary operator: cast, i.e. (type_name) expr
  - Makes the program more readable

**Function Declaration**
- `<static> <type_name>(parameter list);`
- Provides type checking information to the compiler
  - type: return type
  - name: function name
  - parameter list: specify the parameters passed to the function
  - static: the function is local in the file
  - extern: definition is provided in another file
- Function declarations are often in header files used by many source files
- Undeclared function are assumed to return int and won’t have type checking

```c
<static> type name(parameter list) {
    <static> type name-list;
    <static> type name-list;
    ...
    Statement;
    Statement;
    ...
}
```

**Local Variables**
- Invisible outside of the function
- `<static>`: cause the variable to be stored in permanent storage.
- Can be initialized when declared
- Initialization
  - automatic: every time the function is called. Default initial value is undefined.
  - static: only the first time the function is called. Default initial value is 0.
Parameters

- Specify a list of parameters (in order) passed to the function when calling it
- Specify the type of each parameter
- Specify the variable representing the parameter within the function; they are regarded as local variables and are invisible outside the function. (will talk about scope in detail later)

Return Value

- When to return
  - Come to the "return" statement
  - Fall off the end of the function
- The use of the return value
  - The result of the computation
  - Status of the computation: usually, "0" - OK, "positive" - user error code, "negative" - system error
- Return type
  - int: need to "return var;" (var is integer compatible)
  - char: need to "return var;" (var is char compatible)
  - void: either "return;" or fall off the end
  - etc.

Outline

- Data Types
- Function
- Global Variable
- Pointer
- Argument Passing in Function Calls

Global Variables

- Usually defined at the beginning of the file
  - \texttt{\textless static | extern\textgreater \ type name_list;}
- \texttt{static}: local in this file
- \texttt{extern}: storage allocated in another file
- Can be initialized when declared
- Initialization
  - Conceptually before the program starts execution
  - Default initial value is 0
- Common case
  - Declared in one file and allocate storage
  - Declared in other files as "extern" to access it

Outline

- Data Types
- Function
- Global Variable
- Pointer
- Argument Passing in Function Calls

Pointer

- A variable containing the address of another variable.
- How to identify the memory storage of a variable
  - Starting address
  - How many bytes occupied
- Example:
  ```
  int a;
  int *p; /* Read: \texttt{p} is a pointer to a 'cell' containing an int */
  p = &a; /* Set \texttt{p} to point to a */
  ```
Pointers operators

int * p1 - define a pointer (p1) that points to an int,
*p1 - refers to the value stored in the cell pointed by p1,
&a - returns the address of a;

- int a; int *p1, *p2;
- a = 17;
- p1 = &a;
- p2 = p1;
- printf("%d,%d,%d", a, *p1, *p2);
- /* prints 17, 17, 17 */

Pointer Operations - cont

- Generic pointer: void *
- Type case: (int *) p
- *: can’t be used on generic pointer
- Examples:
  - int a=17;
  - int *p1;
  - void *p2;
  - p1 = p2 = &a;
  - printf("%d", *p1); /* OK */
  - printf("%d", *p2); /* Bad idea */

Outline

- Data Types
- Function
- Global Variable
- Pointer
- Argument Passing in Function Calls

What’s wrong with swap()?

void swap(int x, int y);
void main(void)
{
  int a, b;
  a = 10;
  b = 17;
  swap(a, b);
}
void swap(int x, int y)
{
  int temp;
  temp = x;
  x = y;
  y = temp;
}

Correct swap()

void swap(int *x, int *y);
void main(void)
{
  int a, b;
  a = 10;
  b = 17;
  swap(&a, &b);
}
void swap(int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}

Acknowledgement

- Stanley Yao