C Data Types and Functions

Stanley Yao
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()

```
#include <stdio.h>

#define PRINT_SIZE(type) \
 printf("%d\t:" #type "\n", \
 sizeof(type))

int main(void)
{
    PRINT_SIZE(char);
    PRINT_SIZE(unsigned char);
    PRINT_SIZE(int);
    PRINT_SIZE(short int);
    PRINT_SIZE(long int);
    PRINT_SIZE(long long int);
    PRINT_SIZE(long);
    PRINT_SIZE(float);
    PRINT_SIZE(double);
    PRINT_SIZE(long double);
    return 0;
}
```


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
- Values need not be distinct
- An alternative of “#define” with type checking and auto-generated values

Type Convertion

- Implicit type conversion
  - Promote the “lower” type to “higher” type before the operation. The result is of “higher” type.
  - OK: Narrower → 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

Outline

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

Function Declaration

- **<static|extern> 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

Function Definition

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


Local Variables

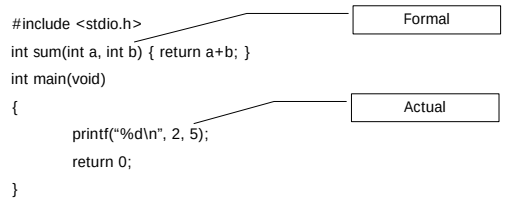
- Invisible outside of the function
- static: cause the variable to be stored in permanent storage
- Can be initialized when declared
- Initialization
  - automatic: every time the function is called. Default initial value is undefined.
  - static: only the first time the function is called. Default 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)

Parameters & Arguments

- Parameter (formal argument)
- Argument (actual argument)



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.

Example Program

```
static void counter(char *);

static void counter(char *msg)
{
   static int cnt = 0;
   int len;
   len = strlen(msg);
   cnt ++;
   printf("cnt=%d, len=%d, %s\n",
     cnt, len, msg);
}
```


Outline

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

Global Variables

- **<static|extern> type name_list;**
- static: local in this file
- 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

Variables Summary

Var Type	Scope	Storage	Init	Default Init Value
automatic	Function	Auto	Every Time	undefined
static automatic	Function	Permanent	First Time	0
external	All files	Permanent	Before Execution	0
static external	This file	Permanent	Before Execution	0

"static" Overload Summary

Item Modified	Meaning
Global variable	Local in the file
Local variable	Permanent storage
Function	Local in the file

Outline

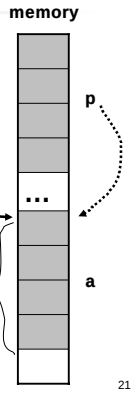
- Data Types
- Function
- Global Variable
- Pointer
- Argument Passing in Function Calls

Pointer

- A variable containing the address of an other variable.

- How to identify the memory storage of a variable
  - Starting address
  - How many bytes occupied
- Example:

```
int a;
int *p;
p = &a;
```

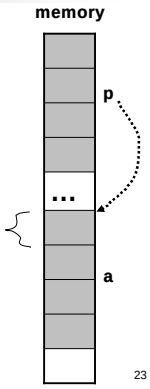


Pointer Operations

- Pointer type declaration: `int *`
- Generic pointer: `void *`
- Type case: `(int *) p`
- &
- *: can't be used on generic pointer
- Examples:
  - int a, b;
  - int *p;
  - p = &a;
  - *p = 10;
  - b = *p + 1;

Pointer Base Type

```
int a;
char *p;
p = &a;
*p = *p + 1;
```

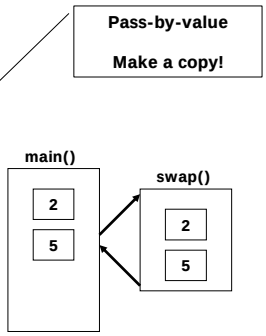


Outline

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

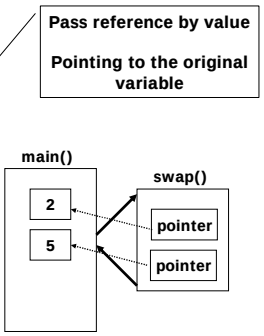
What's wrong with swap()?

```
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 tmp;
   tmp = x;
   x = y;
   y = tmp;
}
```



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 tmp;
   tmp = *x;
   *x = *y;
   *y = tmp;
}
```



Acknowledgement

- John H. Hartman, *Classnotes for Csc352-Spring03*, CS Dept., University of Arizona, 2003
- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language (2nd Ed.),* Prentice Hall, 1988