Functions

Basics of Functions

Function Definition

```
ReturnType functionName (Arg1Type arg1, Arg2Type arg2, ...)
{
    Declarations and statements;
    return value or return
}
```

- if ReturnType is ommitted, by default compiler thinks int is the ReturnType
- if there is no return in side the function body, execution falls off the end, with no return value
- function can be used after definition in the same file;
- to call a function before its definition or from other files requires function declaration

Basics of Functions

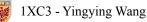
Function Definition

```
ReturnType functionName (Arg1Type arg1, Arg2Type arg2, ...)
{
    Declarations and statements;
    return value or return
}
```

- if **ReturnType** is ommitted, by default compiler thinks **int** is the **ReturnType**
- if there is no return in side the function body, execution falls off the end, with no return value
- function can be used after definition in the same file;
- to call a function before its definition or from other files requires function declaration
- Function Declaration

```
ReturnType functionName (Arg1Type arg1, Arg2Type arg2, ...);
```

- no function body/implementation, declaration ends with;
- should be consistent with the function definition
- argument names can be omitted: ReturnType functionName (Arg1Type, Arg2Type, ...);



Formal parameters

```
int fn(int arg1, float arg2)
   printf("inside int fn(int arg1=%d,
float arg2=%f)\n", arg1, arg2);
    return 2:
int main()
    fn(1, 99.5);//return value ignored
    return 0/;
    actual parameters
```

```
int fn(int, float);
int main()
    fn(1, 99.5);
    return 0;
int fn(int arg1, float arg2)
    printf("inside int fn(int arg1=%d,
float arg2=%f)\n", arg1, arg2);
    return 2:
```



functions defined in a different file

```
#include <stdio.h>
                                         #include <stdio.h>
                                         float callU()//definition
float callU();//declaration
                                             int extension = 3323;
int main()
                                             printf("in callU() extension =
                                         %d\n", extension);
    callU();//call
                                             return 880.112;
    return 0;
                                                         function.c
```

External Variables

- External Variables
 - defined outside of any functions
 - o vs internal (automatic) variables defined inside of a function
- External variables are globally accessible, referencing by names
- External variables provide an alternative to function arguments and return values
- External variables are more convenient if a large number of variables need to be shared among functions
- External variables are permanent, retain values from one function invocation to the next
 - o automatic variables come into existence when function is entered, and disappear when it is left

```
char grade = 'N'; //defined outside of functions, external / global
int fn(int, float);
int main()
    grade = 'A'; //main can access it
    fn(1, 99.5);
    grade = 'C';
    fn(2, 66.0);
    return 0;
int fn(int arg1, float arg2)//not passing through arguments and return values
    printf("inside int fn(int arg1=%d, float arg2=%f)\n", arg1, arg2);
    printf("inside fn accessing grade=%c\n", grade);//fn can access it
    return 2;
                                                                       1XC3 - Yingying Wang
```

external variables defined in a different file

```
#include <stdio.h>
                                         #include <stdio.h>
                                         int extension = 3323;//definition
float callU();
extern int extension;//declaration
                                         float callU()
int main()
                                             printf("in callU() extension =
                                         %d\n", extension);//access here
    extension = 4321;//access here
                                             return 880.112;
    callU();
    return 0;
                                                         function.c
```

Scope

- Automatic variables
 - the scope is the remaining of the function after the variable definition
 - variables of the same name in different functions are unrelated

- External variables or functions
 - o from the point it is declared to the end of the file
 - an extern declaration is mandatory to use an external variable
 - that is defined in a different file,
 - before its definition in the same file

Static

External Static Variables

limits the scope of the variable to the rest of the source file not conflict with the same names in other files of the same project

extern variables defined in a different file

```
#include <stdio.h>
                                         #include <stdio.h>
float callU();
                                         //only this file
extern int extension;//error
                                         static int extension = 3323;
int main()
                                         float callU()
//cannot access in a different file
                                             printf("in callU() extension =
    extension = 4321;
                                         %d\n", extension);
                                             return 880.112;
    callU();
    return 0;
                                                         function.c
```

Static

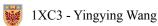
External Static Variables

limits the scope of the variable to the rest of the source file not conflict with the same names in other files of the same project

Internal Static Variables

variables local to a function that remain in existence private permanent storage within a function

```
int fn(int arg1, float arg2)
                                              int main()
    //static
                                                  grade = 'A';
    static int svalue = 0;
                                                  fn(1, 99.5); //svalue 0->1
    svalue ++;
                                                  grade = 'C';
    printf("static svalue=%d\n", svalue);
                                                  fn(2, 66.0); //svalue 1->2
    return 2;
                                                   return 0;
```



Static

External Static Variables

limits the scope of the variable to the rest of the source file not conflict with the same names in other files of the same project

Internal Static Variables

variables local to a function that remain in existence private permanent storage within a function

Static Functions

normally (without static) function names are global, visible to the entire project static function is invisible outside of the file which it is declared

functions defined in a different file

```
#include <stdio.h>
                                          #include <stdio.h>
float callU();//error
                                          //.....
int main()
                                          //only for this file
                                          static float callU()
    callU();
    return 0;
                                              return 880.112;
                                                          function.c
```



Register Variables

Register Variables

advises the compiler that the variable will be heavily used, to be placed in machine registers

can only be applied to automatic variables or the formal parameters of a function

excessive register declarations may get ignored

not possible to take the address of a register variable

vary from machine to machine

```
register float x;
register int i;
```

Initialization

External and static variables

explicit initializer must be a constant expression in the absence of explicit initialization, initialized to zero initialization done once, before the program begins

```
int eint = 3;
//error below: must be const
float efloat = 1 + eint;
//valid below
#define C NUM 3
float efloat2 = 1 + C NUM;
//zeros by default
int eint2;
float efloat3;
```

Initialization

External and static variables

explicit initializer must be a constant expression in the absence of explicit initialization, initialized to zero initialization done once, before the program begins

Automatic and register variables

initializer does not need to be a constant in the absence of explicit initialization, values undefined

```
int aint = 3;
//the following is allowed
float afloat = 1 + aint:
//no explicit initializer
int aint2;//undefined value
float afloat3://undefined value
printf("aint2 = %d, afloat3 =
%f\n", aint2, afloat3);
```

Initialization

External and static variables

explicit initializer must be a constant expression in the absence of explicit initialization, initialized to zero initialization done once, before the program begins

Automatic and register variables

initializer does not need to be a constant in the absence of explicit initialization, values undefined

Array

may be initialized by a list of initializers enclosed in braces { }, separated by commas when the size of the array is **omitted**, the compiler will compute the size by counting the initializer when the size of the array is **specified**, but there are fewer initializers, the others will be zero when the size of the array is **specified**, but there are more initializers, it is an error char array (string) can be initialized using both braces { } and double quotes ""

```
//data array initialization
int data1 [9] = \{1,2,3,4,5,6,7,8,9\};
int data2 [9] = \{1,2,3,4,5\};
//int data3 [8] = \{1,2,3,4,5,6,7,8,9\}; //invalid
int data4 [9];
int data5 [] = \{1,2,3,4,5,6,7,8,9\};
int data6 [] = {1,2,3,4,5}; //different from data2
//string char array initialization
char cdata1[] = \{'q', 'o', 'o', 'd'\}; //\{'q', 'o', 'o', 'd', '\0'\};
char cdata2[] = "good";//ends with '\0'
char cdata3[4] = "good";//problematic
char cdata4[5] = "good";
//char cdata5[3] = "good";
printf("cdata1 = %s, cdata2 = %s, cdata3 = %s, cdata4 = %s\n",
    cdata1, cdata2, cdata3, cdata4);
```

Variable Definition vs. Declaration

Definition vs Declaration

```
declaration announces a variable and its type
```

definition allocates storage, and also serve as declaration for the rest of the file

For external variables

there must be only one definition among all files

other files may contain extern declarations to access it

initialization goes only with the definition

array size is optional in an extern declaration

```
extern int eArray[5];
//or
```

extern int eArray[];

Block

Code Block

```
int x;
int y;

int func(double x)
{
    double y;//this is not a function, just a variable named y
    {
        char x;
    }
}
```

variables defined inside the code block hide any identically named variables in the outer blocks remain in existence until the right brace

Recursion

A function call itself either directly or indirectly, e.g. fibonacci sequence F(n) = F(n-1) + F(n-2)F(0) = 0 and F(1) = 1

```
void countToN recursive(int n)//count from 0 to n void countToN iterative(int n)
    if(n<0)
        return;//terminating condition
    //first count 0 to n-1
    countToN recursive(n-1);//call itself
    //count n
   printf("count %d\n", n);
```

for(int i = 0; $i \le n$; ++ i) printf("count %d\n", i);



int dLen = strlen(strNum), dNum=0, i=0;

Header Files

Definitions & Declarations shared among files

- use #include to insert the content of a (header) file into the current file
- search system header files enclosed in <> under the system directory
- search files enclosed in " " under the user program/current directory, or use -I to specify
- changes in one header impact all files that include it, triggering recompilation of those files

```
//system headers, search under system directory
#include <stdio.h>
#include <stdint.h>
#include <limits.h>
#include <float.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "a2.h"//user defined header file
```

The C Preprocessor

File Inclusion

```
#include <stdio.h>
#include "a2.h"
Macro Substitution
```

Conditional Inclusion

1XC3 - Yingying Wang

The C Preprocessor #define MAX LINE NUM 100 File Inclusion #define FOREVER for(; ;) //macro with arguments: different from functions #define MAX(A, B) ((A) > (B) ? (A) : (B)) **Macro Substitution**

#define SQ(x) x*x Conditional Inclusion

//undef previously defined macro #undef MAX //#takes input string #define PRINT V(v) printf(#v " = %d\n", v)

//concatenation, white spaces surrounding ## are removed

#define CONCAT(first, second) first ## second

The C Preprocessor

File Inclusion

#endif

Macro Substitution

avoid including definition multiple times

Conditional Inclusion

#ifndef A2 H

#define A2 H

//declarations & definitions in a2.h

//declarations & definitions in a2.h

#endif

#define HDR "linux.h" #elif OS == WINDOWS #define HDR "windows.h" #define HDR "mac.h"

//check operating system MACRO

//include header accordingly

#define OS WINDOWS

#if OS == UBUNTU

#elif OS == MAC

#include HDR

#endif

#if !defined(A2 H)

#define A2 H

1XC3 - Yingying Wang

Thank you!