COMsW 1003-1

Introduction to Computer Programming in C

Lecture 7

Spring2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html

Today

- Loops (from Lec6)
- Scope of variables
- Functions

Scope of Variables

- **Scope** is the portion of program in which a variable is valid
- Depends on where the variable is **declared**
- Variables can be
  - **Global** : valid everywhere
  - **Local** : valid in a specific portion of the program included in { }

Scope of Variables

- **Scope** is the portion of program in which a variable is valid
- Depends on where the variable is **declared**
- Variables can be
  - **Global** : valid everywhere
  - **Local** : valid in a specific portion of the program included in { }

```
#include <stdio.h>

double  x = 3;   /* global variable */

int main() {

   double y = 7.2;

   if( x > 2){

      double z = x / 2;

   }

   return(0);
}
```

Scope of x

Scope of y

Scope of z

Scope of variables

```c
#include <stdio.h>

double  z = 1;

int main() {
   printf("z1 = %lf\n", z);              // z1 = 1.0000000
   double z = 7;
   if( z > 2){
      double z =  0.5;
      printf("z2 = %lf\n", z);        // z2 = 0.5000000
   }
   printf("z3 = %lf\n", z);             // z3 = 7.0000000
   {
     double z = 11;
     printf("z4 = %lf\n",z);           // z4 = 11.0000000
   }
   printf("z5 = %lf\n",z);              // z5 = 7.0000000
   return(0);
}
```

Scope of variables

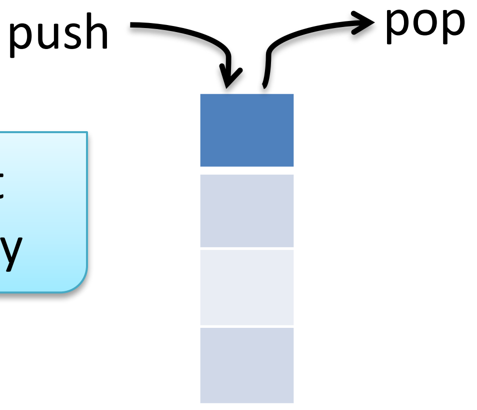
```
#include <stdio.h>

double  z = 1;

int main() {
   printf(”z1 = %lf\n”, z);          // z1 = 1.0000000
   double z = 7;
   if( z > 2){
      double z =  0.5;
      printf(”z2 = %lf\n”, z);      // z2 = 0.5000000
   }
   printf(”z3 = %lf\n”, z);         // z3 = 7.0000000
   {
     double z = 11;
     printf(”z4 = %lf\n”,z);        // z4 = 11.0000000
   }
   printf(”z5 = %lf\n”,z);          // z5 = 7.0000000
   return(0);
}
```

Class of Variables

- A variable can be either
  - **Temporary** : allocated in stack at beginning of block (if too many local variables allocated, stack overflow)
  - **Permanent** : allocated before the program starts

- **Global** variables are always **permanent**

- **Local** variables are **temporary** unless they are declared **static**



Stack: First In Last Out (FILO) type of memory

Variables – Scope and Class

Declared	Scope	Class	initialized
Outside all blocks	Global	Permanent	Once
Static outside all blocks	Global	Permanent	Once
Inside a block	Local	Temporary	Each time block is entered
Static inside a block	Local	Permanent	Once

From PCP Ch 9

```
#include <stdio.h>

int z = 0;
static int b;

int main() {

  int g =  0;

  while( z < 3){

      int y =  0;
      static int x = 0;

      y++;
      x++;
      z++;

      printf("x = %d, y = %d, z = %d\n", x, y, z);

  }
  return(0);
}
```

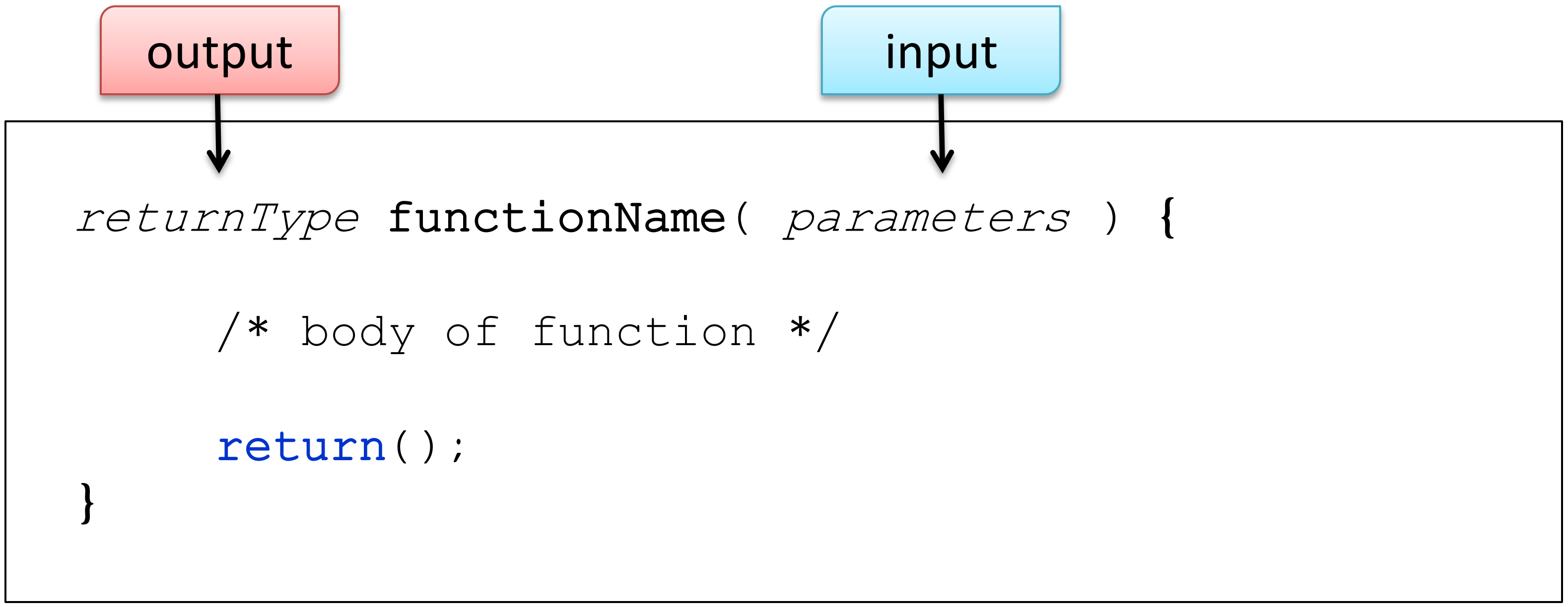
```
x = 1, y = 1, z = 1
x = 2, y = 1, z = 2
x = 3, y = 1, z = 3
```

y is initialized every time

Functions

- Functions allow to write and reuse pieces of code that accomplish a task

- Help keeping large codes ordered



```
returnType functionName( parameters ) {

    /* body of function */

    return();
}
```

Functions - Example

The function *sumTwoNumbers* takes two numbers as input and returns their sum.

```
double sumTwoNumbers( double n1, double n2 ) {

      double s;

      s = n1 + n2;

      return(s);
}
```

Functions - Example

The function *sumTwoNumbers* takes two numbers as input and returns their sum.

```
double sumTwoNumbers( double n1, double n2 ) {

    double s;

    s = n1 + n2;

    return(s);          // return s;
}
```

Returned type must be consistent!

These two notations are equivalent

sumNumbers.c

Functions – Example

```
#include <stdio.h>
double sumTwoNumbers( double n1, double n2 ){
      double s;
      n1++;
      s = n1 + n2;
      return(s);
}
int main() {
      double x, y, z;
       x = 2;
       y = 2;
       z = sumTwoNumbers(x, y);
       printf("%f + %f = %f\n", x, y, z);
       return(0);
}
```

Function Declaration must happen BEFORE its use in the main() function

Scope of n1 and n2

is scope of function!

2 + 2 = 5 !

Functions - void

printArrow.c

- If a function does not take any input
- If a function does not return any value

```
/* function to print an arrow to command line */
void printArrow(void){

       /* function body */

       return;
}

/* function to print multiple arrows to command line */
void printMultipleArrows(int nTimes){

       int i;

       for(i = 0; i < nTimes; i++){

           printArrow();
        }

       return;
}


int main() {

     int x = 3;

     printMultipleArrows(x);

     return(0);

}
```

Functions - void

printArrow.c

- If a function does not take any input
- If a function does not return any value

```
/* function to print an arrow to command line */
void printArrow(void){

        /* function body */

        return;
}

/* function to print multiple arrows to command line */
void printMultipleArrows(int nTimes){

        int i;

        for(i = 0; i < nTimes; i++){

            printArrow();
         }

        return;
}


int main() {

     int x = 3;

     printMultipleArrows(x);

     return(0);
}
```

Function does not return anything

Function invoked without passing any parameter ()

Functions - void

printArrow.c

- If a function does not take any input
- If a function does not return any value

```
/* function to print an arrow to comman
void printArrow(void){

        /* function body */

        return;
}

/* function to print multiple arrows to command line */
void printMultipleArrows(int nTimes){

        int i;

        for(i = 0; i < nTimes; i++){

            printArrow();
         }

        return;
}


int main() {

      int x = 3;

      printMultipleArrows(x);

      return(0);

}
```

Return can be viewed as equivalent of break for functions

Function is declared before being used

Functions – Passing Arrays

length.c

```
/* function to compute the length of a string*/
 int length( char s[] ){

      int size = 0;

      while(s[size] != '\0'){
          size++;
      }

      return size;
 }

 /* function to copy a string*/
 char[] copyString( char s[] ){

      char s2[100];

      strcpy(s2, s);

      return s2;
 }
```

Functions – Passing Arrays

length.c

```
/* function to compute the length of a string*/
int length( char s[] ){

     int size = 0;

     while(s[size] != '\0'){
          size++;
     }

     return size;
}

/* function to copy a string*/
~~char[]~~ copyString( char s[] ){

     char s2[100];

     strcpy(s2, s);

     return s2;
}
```

Functions – exit()

`exit()` is used to exit (=terminate) the program

Different from return, which simply exits the function

Exit() is defined inside the library stdlib.h

```
#include <stdlib.h>

 int length( char s[] ){

        int size = 0;

        while(s[size] != '\0'){

             if(s[size] == 'm')
                  exit(-1);

             size++;
        }

        return size;
 }
```