

ELEC3730 Embedded Systems Lecture 4: Further C Essentials

- Functions
- Dynamic Memory Allocation
- The Standard Library



Functions - why and how ?

- If a program is too long
- Modularization – easier to
 - code
 - debug
- Code reuse
- Passing arguments to functions
 - *By value*
 - *By reference*
- Returning values from functions
 - *By value*
 - *By reference*

Functions – basic example

```
#include <stdio.h>
int sum(int a, int b); /* Function prototype /

void main(void)
{
    int total = sum(4,5); /* call to the function *
    printf("The sum of 4 and 5 is %d", total);
}

/* Function definition - arguments passed by value */
int sum(int a, int b)
{
    return (a+b); /* return by value */
}
```

Function Example - Arguments by reference

```
#include <stdio.h>
int sum(int *pa, int *pb); /* Function Prototype */

void main(void)
{
    int a=4, b=5;
    int *ptr = &b;
    int total = sum(&a,ptr); /* call to the function */

    printf("The sum of 4 and 5 is %d", total);
}

/* Function Definition. Arguments passed by reference */
int sum(int *pa, int *pb)
{
    return (*pa+*pb); /* return by value */
}
```

Why pointer arguments?!

```
include <stdio.h>

void swap(int, int);

main()
{
    int num1 = 5, num2 = 10;
    swap(num1, num2);
    printf("num1 = %d and num2 = %d\n", num1, num2);
}

/* Function Definition. Args passed by value */
void swap(int n1, int n2)
{
    int temp;

    temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Why pointer arguments? This is why

```
include <stdio.h>

void swap(int *, int *);

main()
{
    int num1 = 5, num2 = 10;
    swap(&num1, &num2);
    printf("num1 = %d and num2 = %d\n", num1, num2);
}

/* Function Definition. Args passed by reference */
void swap(int *n1, int *n2)
{
    int temp;

    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Pointer to function

```
int func();      /*function returning integer*/
int *func();    /*function returning pointer to integer*/
int (*func)();  /*pointer to function returning integer*/
int **func();   /*pointer to func returning ptr to int*/
```

- Advantage ? more flexibility

Dynamic Memory Allocation

- Pointer errors can be nasty

```
void main ()
{
    int *p; /* p=address. It is not initialised here */
    *p = 5; /* We have no idea where this data is stored*/
}
```
- Reserve space for the "pointed to" object:

```
#include <stdlib.h>
void main ()
{
    int *p;
    p = (int *)malloc (sizeof (int));
    *p = 5;
    printf ("value at p = %d\n", *p);
    free (p); /* must do this yourself! */
}
```

Finding Bugs

- Simple: Insert "print" statements:

```
printf ("At point A, %s = %d\n", "x", x);
```
- Sophisticated: Use a debugger to help you catch bugs in the act:
 - run step-by-step
 - insert breakpoints
 - display variables

Programming Errors I

```
#include <stdio.h>

void main ()
{
    int i;

    scanf ("%d", &i);
    if (i = 0)
        puts ("false");
    else
        puts ("true");
}
```

- What's wrong?

Programming Errors I

```
#include <stdio.h>
void main ()
{
    int i;

    scanf ("%d", &i);
    if (i = 0)
        puts ("false");
    else
        puts ("true");
}
```

- The assignment, `i = 0`, will return a value of 0, so the program will always output "true"
- Use:
`if(i == 0)`

Programming Errors II

```
#include <stdio.h>

void main ()
{
    int i;

    /* echo one number */
    scanf ("%d", &i);
    printf ("input = %d\n", i);
}
```

- What's wrong?
- Hint: Segmentation fault / Bus error

Programming Errors II

```
#include <stdio.h>

void main ()
{
    int i;

    /* echo one number */
    scanf ("%d", i);
    printf ("input = %d\n", i);
}
```

- Must provide the address of an integer if the function is going to assign a value to it.
- Use:
`scanf ("%d", &i);`

Programming Errors III

```
#include <stdio.h>

void main ()
{
    int *pc;

    scanf ("%d", pc);
    printf ("%d\n", pc);
}
```

- What's wrong?

Programming Errors III

```
#include <stdio.h>
#include <stdlib.h>

void main ()
{
    int *pc;

    scanf ("%d", pc);
    printf ("%d\n", pc);
}
```

- *pc is an uninitialized pointer so results are unpredictable
- Use

```
pc = malloc(sizeof(int));
if (pc != NULL)
{
    ...
}
free(pc);
```

Programming Errors IV

```
#include <stdlib.h>

void main ()
{
    double *p;
    p = malloc (sizeof (double *));
}
```

- What's wrong?

Programming Errors IV

```
#include <stdlib.h>

void main ()
{
    double *p;
    p = malloc (sizeof (double *));
}
```

- Insufficient memory allocated for a double (typically, 8 bytes)
- Use

```
malloc (sizeof (double));
```

Programming Errors V

```
#include <stdio.h>

void main ()
{
    char s[] = "hi";

    if (s == "hi")
        puts ("Strings are equal");
    else
        puts ("Strings are not equal");
}
```

- What happens?

Programming Errors V

```
#include <stdio.h>
main ()
{
  char s[] = "hi";
  if (s == "hi")
    puts ("Strings are equal");
  else
    puts ("Strings are not equal");
}
```

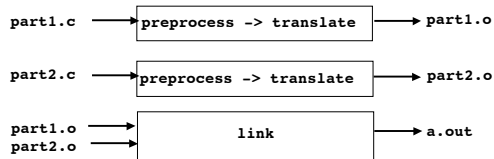
- Use `strcmp()` to compare strings, otherwise you are comparing addresses.

- Use:

```
if strcmp(s,"hi")
  puts("Strings are equal");
else
  puts("Strings are not equal");
```

Separate Compilation

```
% gcc -c part1.c
% gcc -c part2.c
% gcc part1.o part2.o
% a.out
```



Program with multiple files

```
#include <stdio.h>
#include "mypgm.h"

void main(void)
{
  myproc();
}
```

hw.c

```
#include <stdio.h>
#include "mypgm.h"

void myproc(void)
{
  mydata=2;
  . . . /* some code */
}
```

mypgm.c

```
void myproc(void);
int mydata;
```

mypgm.h

- Library headers
 - Standard
 - User-defined

C Preprocessor Directives

- Operations before compilation phase:
 - include "header" files that contain pre-supplied functions:

```
#include <string.h>
```

- this line will be substituted by the contents of the file
- define symbolic constants:

```
#define MY_LARGE_NUMBER 12345  
#define MY_STRING "abc"  
#define PI 3.14
```

Header Files

```
/* File: myheader.h  
 * Definitions for program build.c  
 */  
  
/* Booleans */  
typedef int boolean_t;  
#define TRUE 1  
#define FALSE 0  
  
/* node type */  
typedef struct node_t  
{  
    ...  
}  
  
/* Function prototypes */  
node_t *NewNode(char *label, int cost);  
void KillNode (node_t *node);
```

The C standard library

C itself is a simple language

Main functionality achieved by "library functions"

Functions divided into classes – specs in header files

- Stdlib.h: Utility functions
- Stdio.h: Input and Output
- Math.h: Mathematical functions
- String.h: String functions
- Time.h: Time and data functions

Stdlib.h

- Constants
 - `EXIT_FAILURE` : failure status of `exit()`
 - `EXIT_SUCCESS` : success status of `exit()`
 - `RAND_MAX` : maximum value returned by `rand()`
 - `NULL` : null pointer constant (zero)
- Functions
 - Return absolute values
 - `int abs(int n); long labs(long n);`
 - Convert character to number it represents
 - `double atof(const char* s); int atoi(const char* s);`
 - Dynamic Memory Allocation
 - `void* malloc(size_t size); void free(void* p);`
 - Exit program
 - `void exit(int status);`
 - Generate a random number
 - `int rand(void); void srand(unsigned int seed);`

Stdio.h

- Constants
 - `FILE` : type of object holding information necessary to control a stream
 - `EOF` : value used to indicate end-of-stream or to report an error
- Functions
 - Open and close a stream to a file
 - `FILE* fopen(const char* filename, const char* mode);`
 - `int fclose(FILE* stream);`
 - Delete/Rename a file
 - `int remove(const char* filename);`
 - `int rename(const char* oldname, const char* newname);`
 - Print to and read from a file
 - `int fprintf(FILE* stream, const char* format, ...);`
 - `int fscanf(FILE* stream, const char* format, ...);`
 - Print to and read from console stream
 - `int printf(const char* format, ...);`
 - `int scanf(const char* format, ...);`
 - Get a character from std input, put a character to the std output
 - `int getchar(void);`
 - `int putchar(int c);`

Math.h

- Functions
 - Exponential of x
 - `double exp(double x);`
 - Natural logarithm
 - `double log(double x);`
 - Base 10 logarithm
 - `double log10(double x);`
 - Base-10 logarithm of x
 - `double pow(double x, double y);`
 - Square root of x
 - `double sqrt(double x);`
 - Smallest integer not less than x
 - `double ceil(double x);`
 - Largest integer not less than x
 - `double floor(double x);`
 - Absolute Value of x
 - `double fabs(double x);`
 - Standard Trigonometric Functions
 - `double sin(double x); double cos(double x);`
 - `double tan(double x);`
 - Standard Inverse Trigonometric Functions
 - `double asin(double x); double acos(double x);`
 - `double atan(double x);`

String.h

- Functions
 - Concatenate ct and s and return s
`char* strcat(char* s, const char* ct);`
 - Compare cs with ct
`int strcmp(const char* cs, const char* ct);`
 - Return length of cs
`size_t strlen(const char* cs);`
 - Copies n bytes from ct to s and return s
`void* memcpy(void* s, const void* ct, size_t n);`
 - Copies n bytes from ct to s and return cs
`void* memmove(void* s, const void* ct, size_t n);`
 - Compares n bytes pointed to be cs and ct and return 0 if all equal
`int memcmp(const void* cs, const void* ct, size_t n);`
 - Replaces each of the first n characters of s by c and return s
`void* memset(void* s, int c, size_t n);`