

CSE I

Nicholas Wickström

IDE, Högskolan i Halmstad

2011

Outline

- 1 Introduction
 - Programming of embedded systems
- 2 Programming in C
 - C basics

Outline

- 1 Introduction
 - Programming of embedded systems
- 2 Programming in C
 - C basics

Programming of embedded systems

- Characteristics of an embedded systems - (Simple computer systems model; processor m. peripherals, I/O, memory)
- Programming embedded systems req detailed knowledge about the hardware (specification, data sheets, libraries,)
- Computer languages for embedded systems programming - Assembler, C, C++??, java??
- Development tools (IDE, debugger, memory, lexical checkers)

Programming of embedded systems

- Characteristics of an embedded systems - (Simple computer systems model; processor m. peripherals, I/O, memory)
- Programming embedded systems req detailed knowledge about the hardware (specification, data sheets, libraries,)
- Computer languages for embedded systems programming - Assembler, C, C++??, java??
- Development tools (IDE, debugger, memory, lexical checkers)

Programming of embedded systems

- Characteristics of an embedded systems - (Simple computer systems model; processor m. peripherals, I/O, memory)
- Programming embedded systems req detailed knowledge about the hardware (specification, data sheets, libraries,)
- Computer languages for embedded systems programming - Assembler, C, C++??, java??
- Development tools (IDE, debugger, memory, lexical checkers)

Programming of embedded systems

- Characteristics of an embedded systems - (Simple computer systems model; processor m. peripherals, I/O, memory)
- Programming embedded systems req detailed knowledge about the hardware (specification, data sheets, libraries,)
- Computer languages for embedded systems programming - Assembler, C, C++??, java??
- Development tools (IDE, debugger, memory, lexical checkers)

Programming of embedded systems

- No supporting operating system
- Optimize system for computational performance and memory
- Programming close to the hardware add another dimension, added difficulty

Programming of embedded systems

- No supporting operating system
- Optimize system for computational performance and memory
- Programming close to the hardware add another dimension, added difficulty

Programming of embedded systems

- No supporting operating system
- Optimize system for computational performance and memory
- Programming close to the hardware add another dimension, added difficulty

Outline

- 1 Introduction
 - Programming of embedded systems
- 2 Programming in C
 - C basics

C basics

- Basic structure of a C program, reserved words
- Variables, types
- Storage classes *auto*, *register*, *volatile*, *static*, *extern*

C basics

- Basic structure of a C program, reserved words
- Variables, types
- Storage classes *auto*, *register*, *volatile*, *static*, *extern*

C basics

- Basic structure of a C program, reserved words
- Variables, types
- Storage classes *auto*, *register*, *volatile*, *static*, *extern*

enkel.c

```
/* enkel.c */
#include <stdlib.h> /* Common used functions in here */
#include "simplsum.h" /* CalculateSum defined here */
#define OK 1

int main(void)
{
    int nCounter, n;
    float fSum, f;
    ...
    fSum = CalculateSum(nArg1, fArg2);
    ...
    return OK;
}
```

simplesum.h - simplesum.c

```
/* simplesum.h */  
extern float CalculateSum(int nArg1, float fArg2);  
  
/* simplesum.c */  
float CalculateSum(int nArg1, float fArg2)  
{  
    return( (float)nArg1+fArg2 )  
}
```


Reserved words

**auto, break, case, char, const, continue, default, do, double,
else, enum, extern, float, for, goto, if, int, long, register,
return, short, signed, sizeof, static, struct, switch, typedef,
union, unsigned, void, volatile, while**

Variables

- Scope - Local, Global
- Life-time
- Memory usage - dependant on developing environment and hardware (*limits.h*, *float.h*)
- Data type
- Declaration/Definition/Initialization

Data types

- Basic types - **integers, floating-point numbers**
- Types based on the basic types - **array, struct, union, enumerating**
- Pointers to other types or functions
- Special type - **void**
- Integer types- *signed/unsigned* (**int, short, char, long**)
- Floating-point types - **float, double, long double**
- Constants **const**

Data types (cont.)

```
int nVect[100]; /* Vektor of integers */
struct car_spec {
    int nCyls;
    float fHorsePower;
};

struct car_spec rSaab, rVolvo;
/* rSaab and rVolvo variables of the type: struct car_spec */
rSaab.nCyls = 4; /* structure member operator */

enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
             AUG, SEP, OCT, NOW, DEC};
/* enumerating type */
```

Type casting

```
int    nApples;
float  fParts;
float  fResult;

fResult = nApples/fParts; /* Implicit type cast */
/* Result is float */

/* Note! 5/2 => 2 while 5.0/2.0 => 2.5 */

fResult = (float) nApples; /* Explicit type cast */
```

Storage classes

```
int      nApples;
/* auto int nApples,
   memory reserved automatically */

register int nCnt;
/* Eg. a loop counter stored in a register */

volatile int nInterrupt;
/* not in register or removed in optimization */

static int nLocal; /* Internal global variable */
extern int nGlobal; /* One definition,
several declarations */
```