Embedded Systems Programming - PA8001

www2.hh.se/staff/vero/embeddedProgramming

Lecture 1

Verónica Gaspes
www2.hh.se/staff/vero

Center for Research on Embedded Systems
School of Information Science, Computer and Electrical Engineering

Programming Embedded Systems

Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Programming Embedded Systems

Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Programming Embedded Systems

Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Programming Embedded Systems

Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Programming Embedded Systems

Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Programming Embedded Systems

Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Programming Embedded Systems

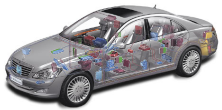
Course Goals

On completion of the course students will be able to

1. program embedded applications
2. understand and use a kernel to support concurrency, real-time and reactivity
3. design, structure and analyze programs for embedded systems
4. explain different mechanisms for communication and synchronization between processes
5. explain characteristics of real-time systems and constructions to deal with them in programs
6. compare, select and apply programming language constructs designed for concurrency and real-time

Yet another programming course?

The programming techniques we learn about in this course address the fact that embedded computer systems are interfaced to physical equipment that they monitor and control.

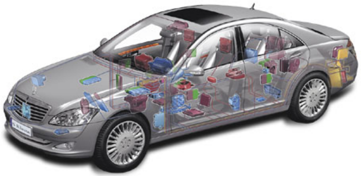


REAL-Time Trend Graphs
Show the effect of diet, exercise, medication and lifestyle on glucose levels
REAL-Time Readings
• Help patients take action sooner
• Up to 288 glucose readings per day_ every 5 minutes, 24 hours a day
REAL-Time Alarms
Protect patients by warning of low and high glucose levels
Wireless Transmitter
Small, discreet, and waterproof
Glucose Sensor
Up to 3-day continuous use
REAL-Time Trend Arrows
Point up or down to show the direction and rate of change in glucose levels
ACT
PARADIGM
10:52
108

Cars that run on code

IEEE Spectrum, Feb. 2009

. . . if you bought a premium-class automobile recently, it probably contains close to 100 million lines of software code, says Manfred Broy, a professor of informatics at Technical University, Munich, and a leading expert on software in cars. All that software executes on 70 to 100 microprocessor-based electronic control units (ECUs) networked throughout the body of your car.



Even low-end cars now have 30 to 50 ECUs embedded in the body, doors, dash, roof, trunk, seats and just about anywhere else the car's designers can think to put them.

Yet another programming course?

Concurrency

A major issue is that external real-world elements exist and evolve in parallel, so we will deal with how to express concurrency in our programs.

Time constrained reactions

Another issue is the need for timely reaction to the physical environment, we will thus deal with how to express time constraints and achieve reactivity in our programs.

Yet another programming course?

Concurrency

A major issue is that external real-world elements exist and evolve in parallel , so we will deal with how to express concurrency in our programs.

Time constrained reactions

Another issue is the need for timely reaction to the physical environment , we will thus deal with how to express time constraints and achieve reactivity in our programs.

Yet another programming course?

Concurrency

A major issue is that external real-world elements exist and evolve in parallel, so we will deal with how to express concurrency in our programs.

Time constrained reactions

Another issue is the need for timely reaction to the physical environment, we will thus deal with how to express time constraints and achieve reactivity in our programs.

Cars that run on code

IEEE Spectrum, Feb. 2009

Most of the time the air bag system is just monitoring the car's condition, but if the air bags are triggered by, say, a multiple vehicle collision, the software in the ECU controlling their deployment has 15 to 40 milliseconds to determine which air bags are activated and in which order, says Broy.

But also . . .

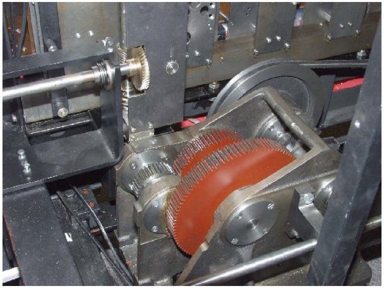
In embedded systems it is often the case that the programs we write have to directly access the hardware that is connected to the processor.

In order to be able to practice with embedded systems, we start the course from this end! The next two lectures are about using C and programming close to hardware!

But also . . .

In embedded systems it is often the case that the programs we write have to directly access the hardware that is connected to the processor.

In order to be able to practice with embedded systems, we start the course from this end! The next two lectures are about using C and programming close to hardware!



The lab environment

AVR-butterfly

A demonstration board including

- An 8-bit CPU with 6 Kbyte memory for code storage and 512 bytes for data storage
- 100-segment LCD (6 digits)
- mini joystick
- temperature and voltage sensors
- piezo speaker

No operating system! Free C-based programming environment for your PC!

Yet another lab environment

Smart phones with Android

After 4 weeks we will move to programming in Java for Android.

We will use Java/Android support for GUIs and the package for concurrency. We will try to use both camera and GPS.

Plan for the course

- Bare metal programming in C. Laborations 0 and 1.
- Concurrent threads and mutual exclusion. Implementing and using a little kernel. Laboration 2.
- Reactive objects. Programming using a little kernel that supports reactive objects. Laboration 3.
- Real-Time, scheduling and programming with time constraints. Laboration 3.
- Programming language support for embedded systems programming. Java, some paper reading and computer based exercises on android. Laboration 4.

Plan for the course

- Bare metal programming in C. Laborations 0 and 1.
- Concurrent threads and mutual exclusion. Implementing and using a little kernel. Laboration 2.
- Reactive objects. Programming using a little kernel that supports reactive objects. Laboration 3.
- Real-Time, scheduling and programming with time constraints. Laboration 3.
- Programming language support for embedded systems programming. Java, some paper reading and computer based exercises on android. Laboration 4.

Plan for the course

- Bare metal programming in C. Laborations 0 and 1.
- Concurrent threads and mutual exclusion. Implementing and using a little kernel. Laboration 2.
- Reactive objects. Programming using a little kernel that supports reactive objects. Laboration 3.
- Real-Time, scheduling and programming with time constraints. Laboration 3.
- Programming language support for embedded systems programming. Java, some paper reading and computer based exercises on android. Laboration 4.

Plan for the course

- Bare metal programming in C. Laborations 0 and 1.
- Concurrent threads and mutual exclusion. Implementing and using a little kernel. Laboration 2.
- Reactive objects. Programming using a little kernel that supports reactive objects. Laboration 3.
- Real-Time, scheduling and programming with time constraints. Laboration 3.
- Programming language support for embedded systems programming. Java, some paper reading and computer based exercises on android. Laboration 4.

Plan for the course

- Bare metal programming in C. Laborations 0 and 1.
- Concurrent threads and mutual exclusion. Implementing and using a little kernel. Laboration 2.
- Reactive objects. Programming using a little kernel that supports reactive objects. Laboration 3.
- Real-Time, scheduling and programming with time constraints. Laboration 3.
- Programming language support for embedded systems programming. Java, some paper reading and computer based exercises on android. Laboration 4.

Plan for the course

- Bare metal programming in C. Laborations 0 and 1.
- Concurrent threads and mutual exclusion. Implementing and using a little kernel. Laboration 2.
- Reactive objects. Programming using a little kernel that supports reactive objects. Laboration 3.
- Real-Time, scheduling and programming with time constraints. Laboration 3.
- Programming language support for embedded systems programming. Java, some paper reading and computer based exercises on android. Laboration 4.

Administrivia

- Web page under
  www2.hh.se/staff/vero/embeddedProgramming
- Teachers
  veronica.gaspes@hh.se
  essayas.gebrewahid@hh.se
- 2 lectures per week
- 4hs supervised lab time per week
- 4-5 relatively big labs – with deadlines, part of the examination. Work in groups of 2.
- 1 written exam

Count on 20 hours work per week plus preparation for the exam.

Literature

third edition
Alan Burns and Andy Wellings
Real-Time Systems & Programming Languages
Ada 95, Real-Time Java and Real-Time POSIX
includes real-time Java™

To some extent the book

Real-Time Systems and Programming Languages

by Allan Burns and Andy Wellings is a very good exposition and I will follow some arguments from the book. However, there are large parts of the book that we will not cover.

We will also use some documents that will be made available on-line.

Acknowledgment

Most of the software and the ideas in the course have been developed by Johan Nordlander at Luleå Technical University.

Course Evaluation Follow-up

Last year's students evaluation

1. Most of the students were very positive to the course!
2. Some students did not seem to know that laborations had a deadline and that labs have to be passed while the course takes place (submitting on deadline is of course a requirement to passing them).
3. 50% of the students were very satisfied with the seminars during the course. There were no seminars in the course!

We are interested in constructive comments about the course! You are very welcome to talk to me or email me with your opinions during the course.

Course Evaluation Follow-up

Last year's students evaluation

1. Most of the students were very positive to the course!
2. Some students did not seem to know that laborations had a deadline and that labs have to be passed while the course takes place (submitting on deadline is of course a requirement to passing them).
3. 50% of the students were very satisfied with the seminars during the course. There were no seminars in the course!

We are interested in constructive comments about the course! You are very welcome to talk to me or email me with your opinions during the course.

Course Evaluation Follow-up

Last year's students evaluation

1. Most of the students were very positive to the course!
2. Some students did not seem to know that laborations had a deadline and that labs have to be passed while the course takes place (submitting on deadline is of course a requirement to passing them).
3. 50% of the students were very satisfied with the seminars during the course. There were no seminars in the course!

We are interested in constructive comments about the course! You are very welcome to talk to me or email me with your opinions during the course.

Course Evaluation Follow-up

Last year's students evaluation

1. Most of the students were very positive to the course!
2. Some students did not seem to know that laborations had a deadline and that labs have to be passed while the course takes place (submitting on deadline is of course a requirement to passing them).
3. 50% of the students were very satisfied with the seminars during the course. There were no seminars in the course!

We are interested in constructive comments about the course! You are very welcome to talk to me or email me with your opinions during the course.

Course Evaluation Follow-up

Last year's students evaluation

1. Most of the students were very positive to the course!
2. Some students did not seem to know that laborations had a deadline and that labs have to be passed while the course takes place (submitting on deadline is of course a requirement to passing them).
3. 50% of the students were very satisfied with the seminars during the course. There were no seminars in the course!

We are interested in constructive comments about the course! You are very welcome to talk to me or email me with your opinions during the course.

Programming Embedded Systems

Cross Compiling

When preparing software for an embedded system, we use an ordinary computer. We edit and compile on any computer, but the generated code has to run in the embedded processor. The compilers we use are called cross compilers.

In the program we will not use libraries that use the devices of our computer (like keyboard, screen, hard disk). Instead we have to use libraries for the devices in the embedded system, typically via named registers.

Makefiles

A lot of information on how to use the cross compiler, on what sources, what libraries to link, etc is provided in makefiles. We will use them but not discuss them in detail.

Programming Embedded Systems

Cross Compiling

When preparing software for an embedded system, we use an ordinary computer. We edit and compile on any computer, but the generated code has to run in the embedded processor. The compilers we use are called cross compilers.

In the program we will not use libraries that use the devices of our computer (like keyboard, screen, hard disk). Instead we have to use libraries for the devices in the embedded system, typically via named registers.

Makefiles

A lot of information on how to use the cross compiler, on what sources, what libraries to link, etc is provided in makefiles. We will use them but not discuss them in detail.

Programming Embedded Systems

Cross Compiling

When preparing software for an embedded system, we use an ordinary computer. We edit and compile on any computer, but the generated code has to run in the embedded processor. The compilers we use are called cross compilers.

In the program we will not use libraries that use the devices of our computer (like keyboard, screen, hard disk). Instead we have to use libraries for the devices in the embedded system, typically via named registers.

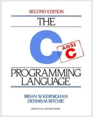
Makefiles

A lot of information on how to use the cross compiler, on what sources, what libraries to link, etc is provided in makefiles. We will use them but not discuss them in detail.

Programming in C

What?

C was designed to be machine independent (has to be compiled!) but supporting low level capabilities (low level access to memory, minimal run-time support).



Why?

We will use it in this course because

- There are C compilers for most micro controllers.
- Exposing the run time support needed for reactive objects helps us understand concurrency, object orientation and real time.

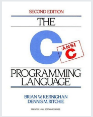
Today

We will look at some C constructs, bit-level ops and some similarities/differences with Java.

Programming in C

What?

C was designed to be machine independent (has to be compiled!) but supporting low level capabilities (low level access to memory, minimal run-time support).



Why?

We will use it in this course because

- There are C compilers for most micro controllers.
- Exposing the run time support needed for reactive objects helps us understand concurrency, object orientation and real time.

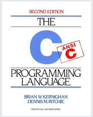
Today

We will look at some C constructs, bit-level ops and some similarities/differences with Java.

Programming in C

What?

C was designed to be machine independent (has to be compiled!) but supporting low level capabilities (low level access to memory, minimal run-time support).



Why?

We will use it in this course because

- There are C compilers for most micro controllers.
- Exposing the run time support needed for reactive objects helps us understand concurrency, object orientation and real time.

Today

We will look at some C constructs, bit-level ops and some similarities/differences with Java.

Program anatomy

A C program consists of

- function declarations,
- a special function `main` that is executed when the program is run,
- global variable declarations,
- type declarations.

There are no classes in C! Larger programs are organized in files, we'll come back to this later today.

Program anatomy

A C program consists of

- function declarations,
- a special function `main` that is executed when the program is run,
- global variable declarations,
- type declarations.

There are no classes in C! Larger programs are organized in files, we'll come back to this later today.

Program anatomy

A C program consists of

- function declarations,
- a special function `main` that is executed when the program is run,
- global variable declarations,
- type declarations.

There are no classes in C! Larger programs are organized in files, we'll come back to this later today.

Program anatomy

A C program consists of

- function declarations,
- a special function `main` that is executed when the program is run,
- global variable declarations,
- type declarations.

There are no classes in C! Larger programs are organized in files, we'll come back to this later today.

Program anatomy

A C program consists of

- function declarations,
- a special function `main` that is executed when the program is run,
- global variable declarations,
- type declarations.

There are no classes in C! Larger programs are organized in files, we'll come back to this later today.

Program anatomy

A C program consists of

- function declarations,
- a special function `main` that is executed when the program is run,
- global variable declarations,
- type declarations.

There are no classes in C! Larger programs are organized in files, we'll come back to this later today.

A first example

```
#include <stdio.h>
int value;
void inc(){
  value++;
}
main(){
  int x;
  value = 0;
  x = value;
  inc();
  printf("%d%s%d",
           value,"  ",x);
  printf("\n");
}
```

preprocessor instruction so that we can use functions defined elsewhere (in `stdio.h`)

A global variable declaration

A function declaration

The function where everything starts! It includes a local variable declaration.

Syntax for statements and declarations, very much like Java!

A first example

```
#include <stdio.h>
int value;
void inc(){
  value++;
}
main(){
  int x;
  value = 0;
  x = value;
  inc();
  printf("%d%s%d",
            value,"  ",x);
  printf("\n");
}
```

preprocessor instruction so that we can use functions defined elsewhere (in `stdio.h`)

A global variable declaration

A function declaration

The function where everything starts! It includes a local variable declaration.

Syntax for statements and declarations, very much like Java!

A first example

```
#include <stdio.h>
int value;
void inc(){
  value++;
}
main(){
  int x;
  value = 0;
  x = value;
  inc();
  printf("%d%s%d",
         value,"  ",x);
  printf("\n");
}
```

preprocessor instruction so that we can use functions defined elsewhere (in `stdio.h`)

A global variable declaration

A function declaration

The function where everything starts! It includes a local variable declaration.

Syntax for statements and declarations, very much like Java!

A first example

```
#include <stdio.h>
int value;
void inc(){
  value++;
}
main(){
  int x;
  value = 0;
  x = value;
  inc();
  printf("%d%s%d",
            value,"  ",x);
  printf("\n");
}
```

preprocessor instruction so that we can use functions defined elsewhere (in `stdio.h`)

A global variable declaration

A function declaration

The function where everything starts! It includes a local variable declaration.

Syntax for statements and declarations, very much like Java!

A first example

```
#include <stdio.h>
int value;
void inc(){
  value++;
}
main(){
  int x;
  value = 0;
  x = value;
  inc();
  printf("%d%s%d",
         value,"  ",x);
  printf("\n");
}
```

preprocessor instruction so that we can use functions defined elsewhere (in `stdio.h`)

A global variable declaration

A function declaration

The function where everything starts! It includes a local variable declaration.

Syntax for statements and declarations, very much like Java!

A first example

```
#include <stdio.h>
int value;
void inc(){
  value++;
}
main(){
  int x;
  value = 0;
  x = value;
  inc();
  printf("%d%s%d",
         value,"  ",x);
  printf("\n");
}
```

preprocessor instruction so that we can use functions defined elsewhere (in `stdio.h`)

A global variable declaration

A function declaration

The function where everything starts! It includes a local variable declaration.

Syntax for statements and declarations, very much like Java!

Standard IO – some details

This is very different from Java!

Formatted output

The function `printf` takes a variable number of arguments:

- Just one, it has to be a string! or
- A first formatting string followed by the values that have to be formatted

Examples

```
printf("Hello World!");
```

Just a string

```
printf("%d%s",value,"\n");
```

Format an integer followed by a string

```
printf("%s%#X",":  ",i);
```

Format a string followed by an integer using hexa-digits

Check the documentation for the library for more details.

Standard IO – some details

This is very different from Java!

Formatted output

The function `printf` takes a variable number of arguments:

- Just one, it has to be a string! or
- A first formatting string followed by the values that have to be formatted

Examples

```
printf("Hello World!");
```

Just a string

```
printf("%d%s",value,"\n");
```

Format an integer followed by a string

```
printf("%s%#X",":   ",i);
```

Format a string followed by an integer using hexa-digits

Check the documentation for the library for more details.

Standard IO – some details

This is very different from Java!

Formatted output

The function `printf` takes a variable number of arguments:

- Just one, it has to be a string! or
- A first formatting string followed by the values that have to be formatted

Examples

```
printf("Hello World!");
```

Just a string

```
printf("%d%s",value,"\n");
```

Format an integer followed by a string

```
printf("%s%#X",":  ",i);
```

Format a string followed by an integer using hexa-digits

Check the documentation for the library for more details.

Standard IO – some details

This is very different from Java!

Formatted output

The function `printf` takes a variable number of arguments:

- Just one, it has to be a string! or
- A first formatting string followed by the values that have to be formatted

Examples

```
printf("Hello World!");
```

Just a string

```
printf("%d%s",value,"\n");
```

Format an integer followed by a string

```
printf("%s%#X",":  ",i);
```

Format a string followed by an integer using hexa-digits

Check the documentation for the library for more details.

Standard IO – other functions

Standard streams

```
#include <stdio.h>
main(){
    char x;
    char buf[10];
   printf("waiting ... \n");
    x = getchar();
    gets(buf);
   printf("got it! \n");
    putchar(x);
   printf("\n");
   printf(buf);
   printf("\n");
}
```

Files

```
#include <stdio.h>
main(){
    FILE *f;
   char x;

    f = fopen("vero","r");
    x = getc(f);
    fclose(f);

    f = fopen("vero","w");
    fprintf(f,"%c",x);
    fclose(f);
}
```

Standard IO – other functions

Standard streams

```c
#include <stdio.h>
main(){
    char x;
    char buf[10];
  printf("waiting ... \n");
    x = getchar();
    gets(buf);
  printf("got it! \n");
    putchar(x);
  printf("\n");
  printf(buf);
  printf("\n");
}
```

Files

```c
#include <stdio.h>
main(){
    FILE *f;
  char x;

    f = fopen("vero","r");
    x = getc(f);
    fclose(f);
    f = fopen("vero","w");
    fprintf(f,"%c",x);
    fclose(f);
}
```

Switching

For discrete types it is possible to choose different actions depending on the value of an expression of that type

```
#include <stdio.h>
main(){
  char x;
  printf("waiting ... \n");
  x = getchar();
  switch(x) {
    case ’a’: printf("This is the first letter \n");
    case ’b’: printf("This is the second letter \n");
    default : printf("This is some other letter \n");
   }
}
```

Arrays

```
#include<stdio.h>
main(){
  int a[10];
  int i;
  for(i = 0;i<10;i++){
    a[i]=i*i;
  }
  for(i = 9; i>=0; i--){
    printf("%d%s%d%s",
           i,"  ",a[i],"\n");
  }
}
```

Different from Java:

`int [] a = new int[10];`

for-control variables have to be declared as variables. In Java they can be declared locally in the loop control

Arrays

```
#include<stdio.h>
main(){
    int a[10];
    int i;
    for(i = 0;i<10;i++){
      a[i]=i*i;
   }
   for(i = 9; i>=0; i--){
      printf("%d%s%d%s",
               i,"  ",a[i],"\n");
   }
}
```

Different from Java:

`int [] a = new int[10];`

`for`-control variables have to be declared as variables. In Java they can be declared locally in the loop control

Arrays

```
#include<stdio.h>
main(){
  int a[10];
  int i;
  for(i = 0;i<10;i++){
    a[i]=i*i;
  }
  for(i = 9; i>=0; i--){
    printf("%d%s%d%s",
          i,"  ",a[i],"\n");
  }
}
```

Different from Java:

`int [] a = new int[10];`

for-control variables have to be declared as variables. In Java they can be declared locally in the loop control

Structures

In C there are no classes! However there is one way of putting together what would correspond to the fields in a class.

```
struct point{
  int x;
  int y;
};

double distance0 ( struct point  p){
  return sqrt( p.x *p.x + p.y*p.y);
}

main(){
  struct point  p = {3,4} ;
  printf("point %d %d \n",p.x,p.y);
  printf("distance %f \n",distance0(p));
}
```

Structures

In C there are no classes! However there is one way of putting together what would correspond to the fields in a class.

```
struct point{
  int x;
  int y;
};
```

```
double distance0 ( struct point  p){
  return sqrt( p.x *p.x + p.y*p.y);
}
```

```
main(){
  struct point  p = {3,4} ;
  printf("point %d %d \n",p.x,p.y);
  printf("distance %f \n",distance0(p));
}
```

Structures

In C there are no classes! However there is one way of putting together what would correspond to the fields in a class.

```
struct point{
  int x;
  int y;
};
```

```
double distance0 ( struct point  p){
  return sqrt( p.x *p.x + p.y*p.y);
}
```

```
main(){
  struct point  p = {3,4} ;
  printf("point %d %d \n",p.x,p.y);
  printf("distance %f \n",distance0(p));
}
```

Structures

In C there are no classes! However there is one way of putting together what would correspond to the fields in a class.

```
struct point{
  int x;
  int y;
};
```

```
double distance0 ( struct point  p){
  return sqrt( p.x *p.x + p.y*p.y);
}
```

```
main(){
  struct point  p = {3,4} ;
  printf("point %d %d \n",p.x,p.y);
  printf("distance %f \n",distance0(p));
}
```

New Types

In order to avoid repeated use of `struct point` as a type, it is allowed to define new types:

```
struct point{
  int x;
  int y;
};

typedef struct point Pt;

double distance0 ( Pt p ){
  return sqrt(p.x*p.x + p.y*p.y);
}
```

New Types

In order to avoid repeated use of `struct point` as a type, it is allowed to define new types:

```
struct point{
  int x;
  int y;
};
```

```
typedef struct point Pt;
```

```
double distance0 ( Pt p ){
  return sqrt(p.x*p.x + p.y*p.y);
}
```

New Types

In order to avoid repeated use of `struct point` as a type, it is allowed to define new types:

```
struct point{
  int x;
  int y;
};
```

```
typedef struct point Pt;
```

```
double distance0 (Pt p){
  return sqrt(p.x*p.x + p.y*p.y);
}
```

New Types

In order to avoid repeated use of `struct point` as a type, it is allowed to define new types:

```
struct point{
  int x;
  int y;
};

typedef struct point Pt;

double distance0 ( Pt p ){
  return sqrt(p.x*p.x + p.y*p.y);
}
```

Pointers

In Java

a declaration like

```
Point p;
```

associates `p` with an address. In order to create a point we need to use the constructor via `new`:

```
   new Point(3,4)
```

This returns the address of a place in memory assigned to this particular point, so it makes sense to do

```
  p = new Point(3,4);
```

In C

We need to use pointers

```
Pt *p;
p = (Pt *)malloc(sizeof(Pt));
p->x = 3; // or (*p).x = 3
p->y = 4;
```

`malloc` is a call to the OS (or platform specific library) requesting a chunk of memory.

Pointers provide direct access to memory addresses!

Pointers

In Java

a declaration like

```
Point p;
```

associates `p` with an address. In order to create a point we need to use the constructor via `new`:

```
   new Point(3,4)
```

This returns the address of a place in memory assigned to this particular point, so it makes sense to do

```
  p = new Point(3,4);
```

In C

We need to use pointers

```
Pt *p;
p = (Pt *)malloc(sizeof(Pt));
p->x = 3; // or (*p).x = 3
p->y = 4;
```

`malloc` is a call to the OS (or platform specific library) requesting a chunk of memory.

Pointers provide direct access to memory addresses!

Pointers

In Java

a declaration like

```
Point p;
```

associates p with an address. In order to create a point we need to use the constructor via `new`:

```
    new Point(3,4)
```

This returns the address of a place in memory assigned to this particular point, so it makes sense to do

```
  p = new Point(3,4);
```

In C

We need to use pointers

```
Pt *p;
p = (Pt *)malloc(sizeof(Pt));
p->x = 3; // or (*p).x = 3
p->y = 4;
```

`malloc` is a call to the OS (or platform specific library) requesting a chunk of memory.

Pointers provide direct access to memory addresses!

Pointers

In Java

a declaration like

```
Point p;
```

associates `p` with an address. In order to create a point we need to use the constructor via `new`:

```
   new Point(3,4)
```

This returns the address of a place in memory assigned to this particular point, so it makes sense to do

```
  p = new Point(3,4);
```

In C

We need to use pointers

```
Pt *p;
p = (Pt *)malloc(sizeof(Pt));
p->x = 3; // or (*p).x = 3
p->y = 4;
```

`malloc` is a call to the OS (or platform specific library) requesting a chunk of memory.

Pointers provide direct access to memory addresses!

Brief on pointers

In Java, memory is reclaimed automatically by the garbage collector. In C, it has to be done by the programmer using another system call:

```
    free(p);
```

In Java all objects are used via addresses. Even when calling functions. In C the programmer is in charge:

```
double distance0 ( Pt *p ){
  return sqrt(p->x*p->x + p->y*p->y);
}
Pt q = {3,4};
printf("distance %f \n",distance0( &q ));
```

Brief on pointers

In Java, memory is reclaimed automatically by the garbage collector. In C, it has to be done by the programmer using another system call:

```
      free(p);
```

In Java all objects are used via addresses. Even when calling functions. In C the programmer is in charge:

```
double distance0 ( Pt *p ){
  return sqrt(p->x*p->x + p->y*p->y);
}
Pt q = {3,4};
printf("distance %f \n",distance0( &q ));
```

Arrays and Pointers

In C, array identifiers are pointers! And pointer arithmetic is available:

```
#include<stdio.h>
main(){
   int a[10];  int *b = a;
  int i;
  for(i=0;i<10;i++){
    a[i]=i*i;
  }
  printf("%d\n", a[0] );
  printf("%d\n", *b );
  printf("%d\n", a[3] );
  printf("%d\n", *(b+3) );
}
```

IO hardware

Access to devices is via a set of registers, both to control the device operation and for data transfer. There are 2 general classes of architecture.

Memory mapped

Some addresses are reserved for device registers! Typically they have names provided in some platform specific header file.

Separate bus

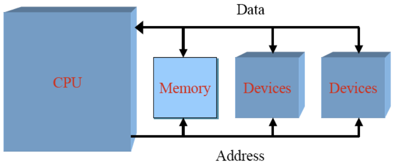
Different assembler instructions for memory access and for device registers.

IO hardware

Access to devices is via a set of registers, both to control the device operation and for data transfer. There are 2 general classes of architecture.

Memory mapped

Some addresses are reserved for device registers! Typically they have names provided in some platform specific header file.



Separate bus

Different assembler instructions for memory access and for device registers.

IO hardware

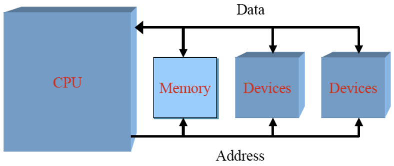
Access to devices is via a set of registers, both to control the device operation and for data transfer. There are 2 general classes of architecture.

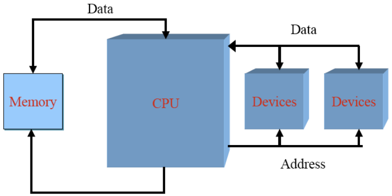
Memory mapped

Some addresses are reserved for device registers! Typically they have names provided in some platform specific header file.

Separate bus

Different assembler instructions for memory access and for device registers.





Bits and Bytes

The contents of device registers are specified bit by bit: each bit has a specific meaning.

Nibbles

A sequence of 4 bits. Enough to express numbers from 0 to 15

0000	0
0001	1
...	...
1111	15

We use hexa-digits for these numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f and we think of their bit-patterns.

Bytes

A sequence of 8 bits. Enough to express numbers from 0 to 255.

00000000	0
00000001	1
...	...
1111111	255

We use 2 hexa-digits , one for each nibble. For example, 0x11 is 00010001 (17 in using decimal digits)

Bits and Bytes

The contents of device registers are specified bit by bit: each bit has a specific meaning.

Nibbles

A sequence of 4 bits. Enough to express numbers from 0 to 15

0000	0
0001	1
...	...
1111	15

Bytes

A sequence of 8 bits. Enough to express numbers from 0 to 255.

00000000	0
00000001	1
...	...
11111111	255

We use hexa-digits for these numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f and we think of their bit-patterns.

We use 2 hexa-digits , one for each nibble. For example, 0x11 is 00010001 (17 in using decimal digits)

Bits and Bytes

The contents of device registers are specified bit by bit: each bit has a specific meaning.

Nibbles

A sequence of 4 bits. Enough to express numbers from 0 to 15

0000	0
0001	1
...	...
1111	15

We use hexa-digits for these numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f and we think of their bit-patterns.

Bytes

A sequence of 8 bits. Enough to express numbers from 0 to 255.

00000000	0
00000001	1
...	...
11111111	255

We use 2 hexa-digits , one for each nibble. For example, 0x11 is 00010001 (17 in using decimal digits)

Bits and Bytes

The contents of device registers are specified bit by bit: each bit has a specific meaning.

Nibbles

A sequence of 4 bits. Enough to express numbers from 0 to 15

0000	0
0001	1
...	...
1111	15

Bytes

A sequence of 8 bits. Enough to express numbers from 0 to 255.

00000000	0
00000001	1
...	...
11111111	255

We use hexa-digits for these numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f and we think of their bit-patterns.

We use 2 hexa-digits , one for each nibble. For example, 0x11 is 00010001 (17 in using decimal digits)

Bits and Bytes

The contents of device registers are specified bit by bit: each bit has a specific meaning.

Nibbles

A sequence of 4 bits. Enough to express numbers from 0 to 15

0000	0
0001	1
...	...
1111	15

Bytes

A sequence of 8 bits. Enough to express numbers from 0 to 255.

00000000	0
00000001	1
...	...
11111111	255

We use hexa-digits for these numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f and we think of their bit-patterns.

We use 2 hexa-digits , one for each nibble. For example, 0x11 is 00010001 (17 in using decimal digits)

Bit level operations

You will need to test the value of a certain bit and you will need to change specific bits (while assigning a complete value). For this you will need bit-wise operations on integer (char, short, int, long) values.

`AND`	`a & b`
`OR`	`a | b`
`XOR`	`a ^ b`
`NOT`	`~a`
`ShiftL`	`a << b`
`ShiftR`	`a >> b`

Example

$$123 \& 234 = 106$$

123 = 0x7b	0	1	1	1	1	0	1	1
234 = 0xea	1	1	1	0	1	0	1	0
123&234 = 0x6a	0	1	1	0	1	0	1	0

Bit level operations

You will need to test the value of a certain bit and you will need to change specific bits (while assigning a complete value). For this you will need bit-wise operations on integer (char, short, int, long) values.

`AND`	`a & b`
`OR`	`a | b`
`XOR`	`a ^ b`
`NOT`	`~a`
`ShiftL`	`a << b`
`ShiftR`	`a >> b`

Example

$$123 \,\&\, 234 = 106$$

$123 = 0x7b$	0	1	1	1	1	0	1	1
$234 = 0xea$	1	1	1	0	1	0	1	0
$123\&234 = 0x6a$	0	1	1	0	1	0	1	0

Bit level operations

You will need to test the value of a certain bit and you will need to change specific bits (while assigning a complete value). For this you will need bit-wise operations on integer (char, short, int, long) values.

AND	`a & b`
OR	`a | b`
XOR	`a ^ b`
NOT	`~a`
ShiftL	`a << b`
ShiftR	`a >> b`

Example

$$123 \ \& \ 234 = 106$$

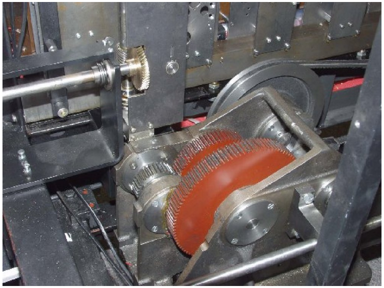
123 = 0x7b	0	1	1	1	1	0	1	1
234 = 0xea	1	1	1	0	1	0	1	0
123&234 = 0x6a	0	1	1	0	1	0	1	0

Laboration 0

Purpose

Become familiar with the lab environment and programming using bit patterns on bare metal.

The lab will be available most of the day, but we offer supervision in three passes a week.

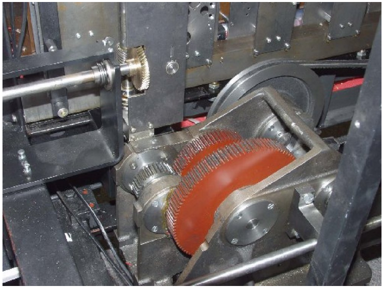


Laboration 0

Purpose

Become familiar with the lab environment and programming using bit patterns on bare metal.

The lab will be available most of the day, but we offer supervision in three passes a week.



Laboration 0

Purpose

Become familiar with the lab environment and programming using bit patterns on bare metal.

The lab will be available most of the day, but we offer supervision in three passes a week.