Embedded Software

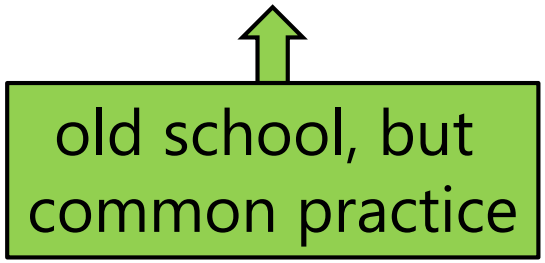
CSE2425

3. C tools

Koen Langendoen
Embedded and Networked Systems

C development cycle

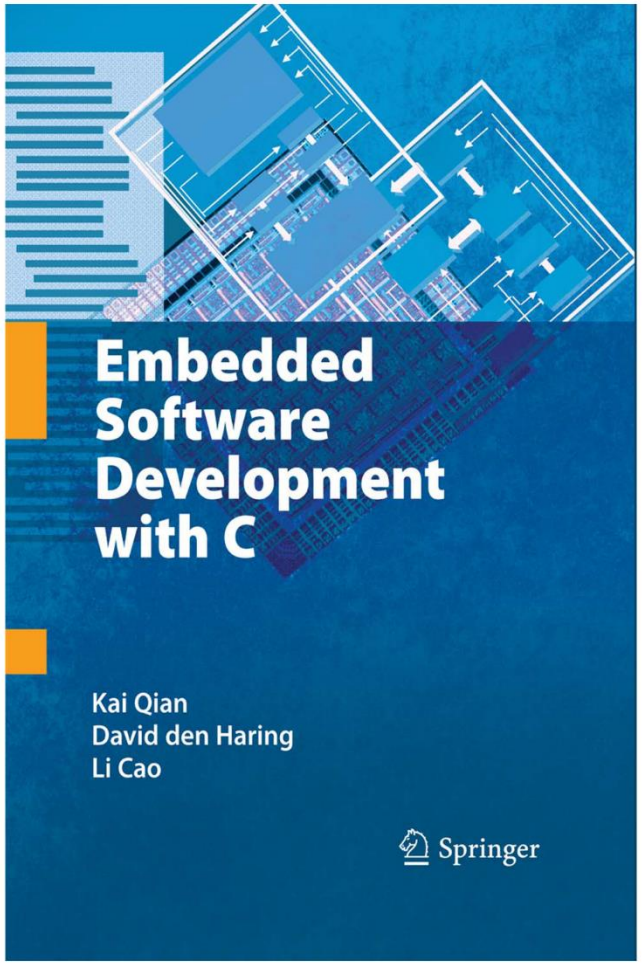
		UNIX toolbox	IDE	
1.	[Think]			
2.	Edit	2. vi, emacs, gedit	2.	eclipse
3.	Compile	3. gcc, make	3.	
4.	Test	4. a.out	4.	
5.	Debug	5. printf, gdb, valgrind	5.	
6.	Tune	6. gprof	6.	



1. Thinking

2. Editing

Thou shalt indent (to the book)




```
main()
{
while(1)
myBit = !myBit;
}
```

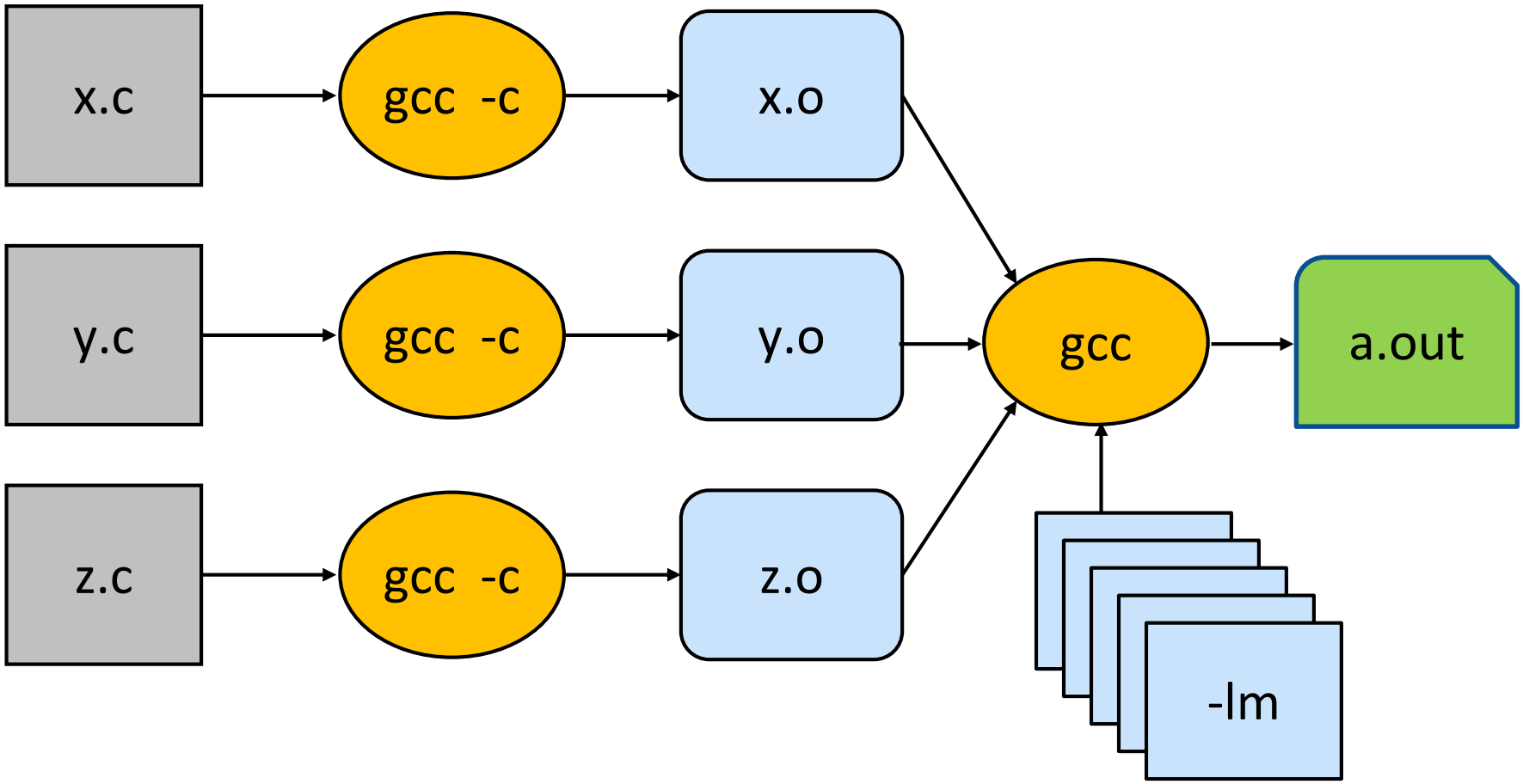
devices.

When you read data from a port or a bit of the port, you need to set in reading mode by writing 1's to all bits of the port or that bit before reading. The actual input data come from some input device connected to the port and the bit of the port. The input device can be a keypad, a switch, or a button.

Here is an example which reads data from one port and writes data to another port.

```
void main(void)
 {
   while (1)   //Foregoing Loop
   {
    P1 = 0XFF; // set P1 reading mode
P1 = 0X7F; //set P1 to initial value 0x7F (127 decimal)

    while(P1 > 0) //Repeat the loop until P1 = 0
    {
    DELAY_Wait(30); //delay for about 30 ms

    P2 = P1;    //read the Port 1, write it to Port 2

    temp = P2;  //copy the value into a variable

temp >>= 1; //cut temp to half by bitwise shift right

     P1 = temp; //assign the new value to Port 1

    }   //end while (P1>0)
    }
  }
```

3. Compiling and linking



Automating compilation

- Dependencies
  - if one file changes many others may need to be recompiled
  - .c depends on .h -> recompile
  - executable depends on .o -> re-link
  - transitive closure

- Make
  - program to generate/check/activate compilation process
  - driven by Makefile capturing dependencies and actions

Makefile

- Rule looks like

```
target: dependencies
        action
```

<tab>!!

```
CC      = gcc
CFLAGS  += -Wall -g -std=c99

SOURCES := $(wildcard *.c)
OBJECTS := $(SOURCES:.c=.o)
TARGET  := prog

$(TARGET): $(OBJECTS)
        $(CC) -o $@ $(OBJECTS)

clean:
        rm -f $(TARGET) $(OBJECTS)
```

- Example Makefile to compile series of C files into one binary
  - built-in rules
- $make

```
gcc -Wall -g -std=c99   -c -o a.o a.c
gcc -Wall -g -std=c99   -c -o b.o b.c
gcc -o prog a.o b.o
```

Beyond make

- Writing Makefiles is cumbersome and error prone
- tools to auto generate Makefiles
  - makedepend
  - automake
  - cmake
- IDE
  - Eclipse + CDT plug-in

4. Testing

- Every module and functionality needs to have an (automated) test
- Regression testing: change -> test old functionality
- Easy for simple functions (unit testing), but what about
  - Input (keyboard) & output (screen)?
  - cross compilation?
  - multi threading?
- Complicated “test harness”

5. Debugging

Typical C-bugs

- memory management
  - buffer overflows / out of bounds
  - dangling pointers
  - memory leaks
- pointers
  - casting
  - arithmetic
- exception handling
  - none!

segmentation fault

Prevention is better than cure

Defensive coding

- use **asserts** – a lot!
- use **const** qualifier

- check return values of all system/library calls

```
char *s = malloc(123);
assert(s != NULL);
```

- check sanity of parameters inside functions

```
void copy(int A[], const int B[], int N) {
    assert( N > 0);
    for (int i = 0; i < N; i++) {
        ...
```

Printf()

Valgrind

- Programming tool for
  - memory debugging
  - memory leak detection
  - profiling

Named after the main entrance to Valhalla in Norse mythology

- Executes your program in a safe environment
  - valgrind [valgrind-options] program [program-options]
  - checks every memory reference
  - keeps track of who allocated/freed what memory when

Example: concat()

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

char *concat(char *s, char *t) {
  return strcpy(s+strlen(s), t);
}

int main() {
  char *a = "hello ";
  char *b = "world";

  printf( "%s + %s = %s\n", a, b, concat(a, b));

  return 0;
}
```

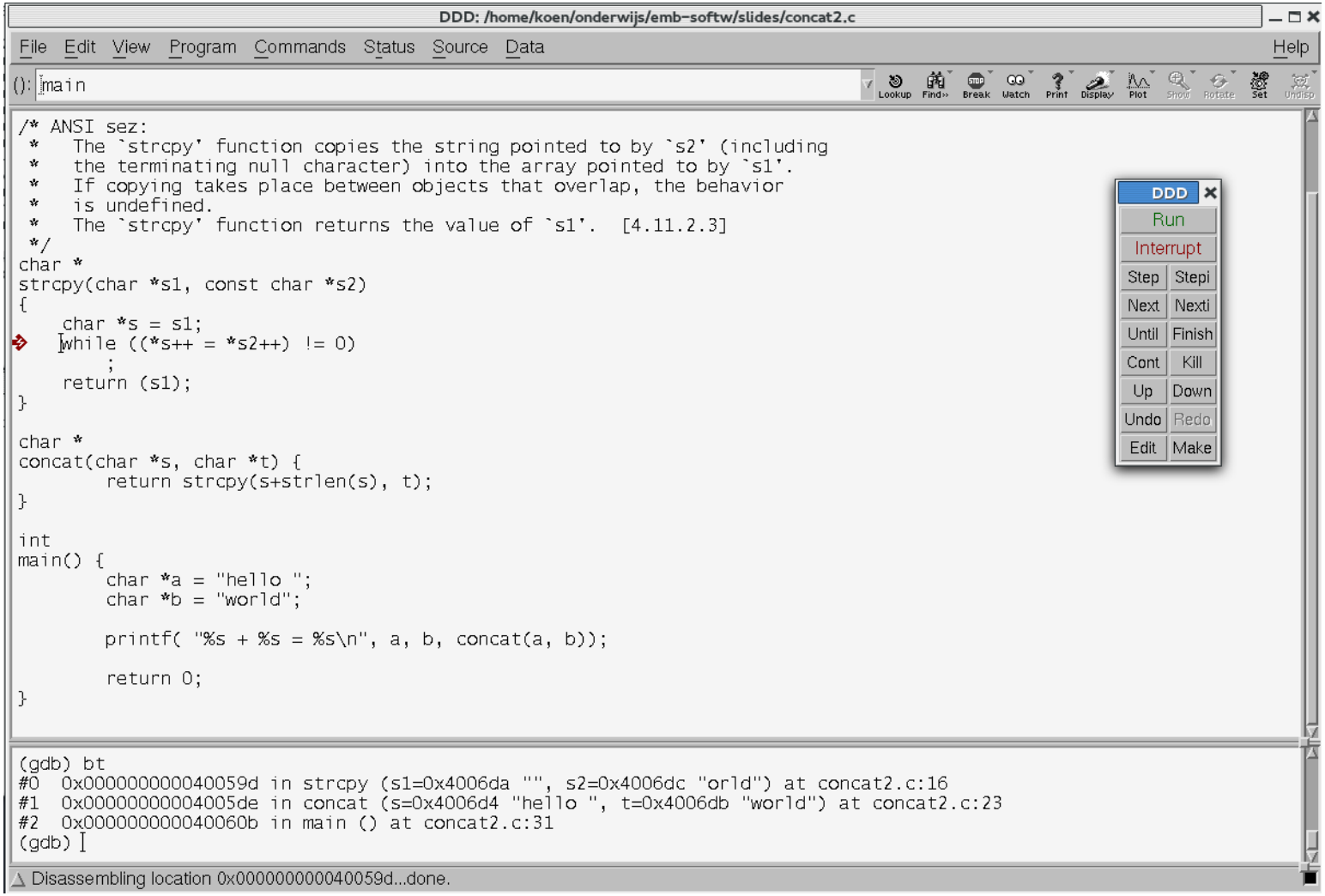
GDB

- Core files are your best friends
  - state of the program at the time of crash
  - $ limit coredumpsize unlimited
  - $ gdb a.out core
- Post mortem inspection
  - call stack
  - global data
  - local variables
- Controlled execution
  - break points
  - step by step

GDB Quick reference

Command	Action
gdb program [core]	debug program [using coredump core]
bt	backtrace: display program stack
up [n]	select frame n frames up
down [n]	select frame n frames down
l	list source code of current frame
p expr	display the value of an expression
b [file:]function	set breakpoint at function [in file]
run [arglist]	start your program [with arglist]
c	continue running your program
n	next line, stepping over function calls
s	next line, stepping into function calls

DDD = GDB with graphical UI



Take 2: concat()

```
char *concat(const char *s, const char *t) {
  char *result = (char *) malloc(strlen(s) + strlen(t));
  strcpy(result, s);
  strcpy(result+strlen(s), t);
  return result;
}

int main() {
  char *a = "hello ";
  char *b = "world";

  printf( "%s + %s = %s\n", a, b, concat(a, b));
  return 0;
}
```

Take 3: concat()

```
char *concat(const char *s, const char *t) {
  char *result = (char *) malloc(strlen(s) + strlen(t) + 1);
  strcpy(result, s);
  strcpy(result+strlen(s), t);
  return result;
}

int main() {
  char *a = "hello ";
  char *b = "world";

  printf( "%s + %s = %s\n", a, b, concat(a, b));
  return 0;
}
```

Take 4: concat()

```
char *concat(const char *s, const char *t) {
  char *result = (char *) malloc(strlen(s) + strlen(t) + 1);
  strcpy(result, s);
  strcpy(result+strlen(s), t);
  return result;
}

int main() {
  char *a = "hello ";
  char *b = "world";
  char *c = concat(a,b);

  printf( "%s + %s = %s\n", a, b, c);
  free( c);
  return 0;
}
```

Take that: concat()

```
char *concat(const char *s, const char *t) {
  size_t ns = strlen(s) + 1;
  size_t nt = strlen(t) + 1;
  char *result = (char *) malloc( ns+nt-1);

  strncpy(result, s, ns);
  strncpy(result+ns-1, t, nt);

  return result;
}

int main() {
  char *a = "hello ";
  char *b = "world";
  char *c = concat(a,b);
```

safe guard against buffer overflow (attacks)

6. Performance tuning

What resources are consumed?

- Code

```
$ size concat7
text        data    bss     dec         hex         filename
1770        592     8       2370        942         concat7
```

- Memory
  - top
  - valgrind/massif – heap profile over time
- CPU
  - gprof – breakdown per function

Before you go….

Hints for successful programming (in C)

- “Haste makes waste”
  - copy & paste considered harmful
  - **no** special casing – follow the rule, not the exception
- “Seeing is believing”
  - test everything (unit testing) and always (regression testing)
  - [C] assert(‘the obvious’)
  - [C] check return values of (system) calls
- “The right tool for the job is worth its weight in gold”
  - [C] valgrind
- “Two heads are better than one”
  - the visitor effect
  - sharing ideas is great, but sharing code is plagiarism

code refactoring