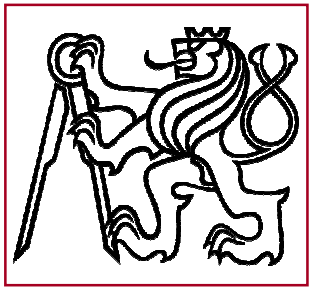
Y35PES

Programming for Embedded Systems

Ondřej Špinka, Pavel Němeček
spinkao@fel.cvut.cz nemecp1@fel.cvut.cz
http://dce.felk.cvut.cz/pes



A Simple C Program Example (1)

The Fibonacci Series

$x_0 = 0 \quad x_1 = 1 \quad x_n = x_{n-1} + x_{n-2}$

```
#include <stdio.h>

#define INDEX 10

int  fibofnc ( int n ) {
    if ( n <= 0 ) return 0;
    if ( n == 1 ) return 1;
    return fibofnc ( n – 1 ) + fibofnc ( n – 2 );
}

int main ( ) {
    int res = fibofnc ( INDEX );
    printf  ( "fibofnc ( %d ) = %d\n", INDEX, res );
    return 0;
}
```

Leonardo Fibonacci (1170-1250)

A Simple C Program Example (2)

```
int  fiboseries ( int **series, int n ) {
    int i;

    if ( n < 0 ) return 0;

    *series = malloc ( sizeof ( int ) * ( n + 1 ) );
    if ( !series ) return 0;

    *series[0] = 0;
     if ( n > 0 ) *series[1] = 1;

    if ( n > 1 )
        for ( i = 2; i <= n; i++ ) *series[i] = *series[i - 1] + *series[i - 2];

    return 1;
}
```

A Simple C Program Example (3)

```
#include <stdio.h>

void  fiboprint ( int *series, int n ) {
    while ( n-- >= 0 )
        printf ( "%d%c", *(series++), n >= 0 ? ' ' : '\n' );
}
```

C Program Decomposition (1)

fibolib.c

```
#include "fibolib.h"
#include <malloc.h>
#include <stdio.h>

int  fiboseries ( int **series, int n ) {
    ...
}

void  fiboprint ( int *series, int n ) {
    ...
}
```

fibolib.h

```
#ifndef FIBOLIB_H
#define FIBOLIB_H

int  fiboseries ( int **series, int n );
void  fiboprint ( int *series, int n );

#endif
```

C Program Decomposition (2)

fibonacci.c

```
#include "fibolib.h"
#define FIBO_NUM 10

void  main (  ) {
     int  *fibo;

    if ( fiboseries ( &fibo, FIBO_NUM ) < 0 ) {
         printf ( "fibo error!\n" );
         while ( 1 ); // for embedded MCU
    }

    fiboprint ( *fibo, FIBO_NUM );
    while ( 1 ); // for embedded MCU
}
```

Makefile (1)

A **makefile** is a batch file containing the compilation and linking directives for a **project**. It is especially useful when compiling big projects consisting of a lot of source files. There are four types of information in a makefile:

- Description of dependencies
- Compilation commands
- Variable settings
- Comments

Basically, a makefile contains the procedural information about how to compile and link a project. **Directions** and **prerequisites** are given for each automatically generated file (**target**), allowing the program **make** to create it.

Makefile (2)

```
all: hello

hello: hello.c
	arm-elf-gcc -Wall hello.c –o hello

clean:
	rm –f hello
```

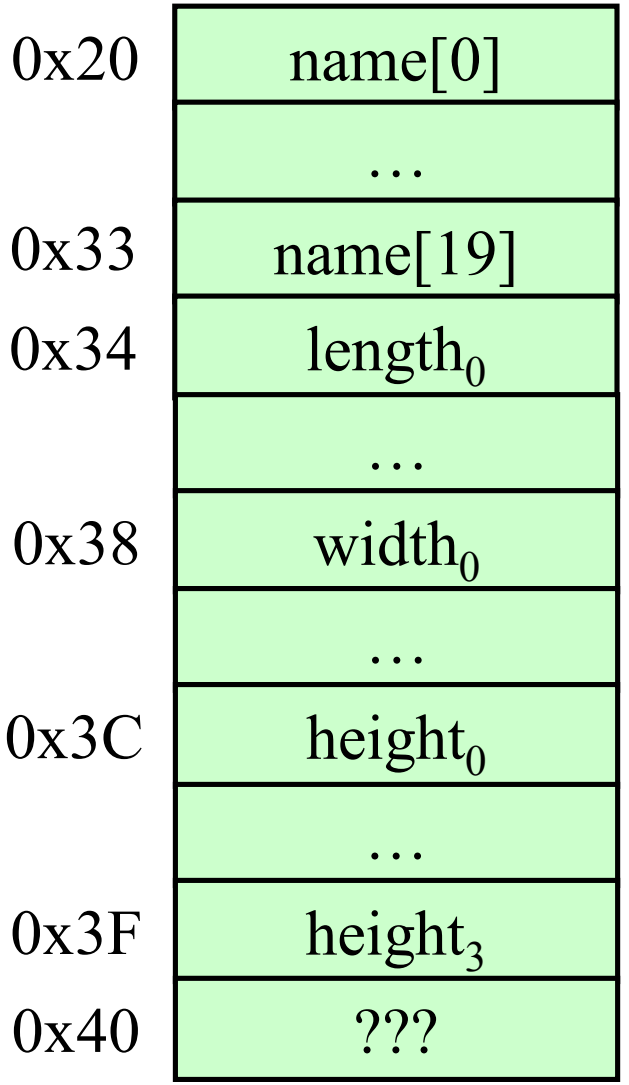
Makefile (3)

```
CC = arm-elf-gcc
CFLAGS = -Wall

all: fibonacci

%.o: %.c
	$(CC) $(CFLAGS) –c $<

fibonacci: fibonacci.o fibolib.o
	$(CC) $(CFLAGS) $^ - o $@

clean:
	rm –f *.o fibonacci
```

Structures and Unions in C (1)

A **structure** data type is used to declare an encapsulated, non-homogenous data type. Respective data items are aligned subsequently in the memory, possibly with gaps between them to allow correct data alignment.

```
struct my_structure {
        char name [20];
        int length;
        int width;
        int height;
}

struct my_structure a, b[5], *c;
```



Definition of a Custom Data Type

It is possible to define a custom data type in the C language. The keyword **typedef** is used for that purpose.

```
typedef struct {
        char name [20];
        int length;
        int width;
        int height;
} my_structure_type;

my_structure_type a, b[5], *c;
```

Structures and Unions in C (2)

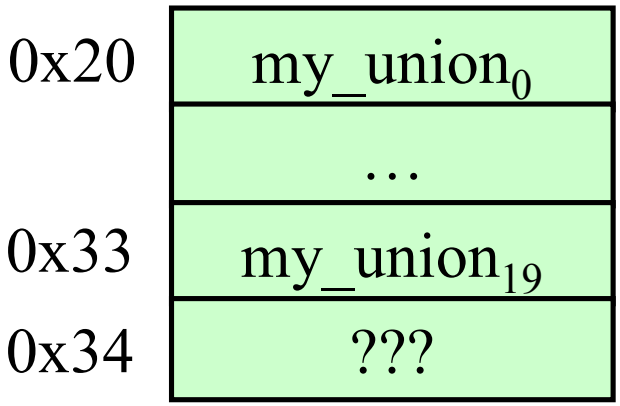
A structure often contains a pointer(s) to itself, to allow dynamic creation of **linked lists**.

```
typedef struct {
        char name [20];
        int length;
        int width;
        int height;
        my_structure_type *previous, *next;
} my_structure_type;

my_structure_type str;
…
str.next = (my_structure_type *) malloc ( sizeof ( my_structure_type ) );
```

Structures and Unions in C (3)

A **union** data type is analogous to structure, but memory is allocated only according the length of the longest data item. The data items in a union cannot be used all simultaneously, but only once in a time.

```
union my_union {
        char name [20];
        int length;
        int width;
        int height;
}

union my_union a, b[5], *c;
```



Enumerations

A **enum** data type is a user-defined index data type. User-defined items are indexed by unsigned integers [0, 1, …]

```
enum colors {red, green, blue};

enum colors color = red;

if ( color == red ) {
        …
}
```

Inline Code

The **inline functions** are special functions, which are not compiled as subroutines, called by branch-and-link instruction. Instead, an inline function body is directly copied by the preprocessor to the places where the function is being called. This contributes to a faster program execution. However, inline functions must be kept very short, or they will generate a lot of code overhead. Usually, inline functions are declared and defined directly in the program headers.

```
inline char read_8b ( void *address ) {
        return *( (char *) address );
}

...

char a = read_8b ( 0x0000F3E6 );
```

C Code Portability

Sometime it might be necessary to **port** an existing program to other hardware platform (for example when upgrading hardware).

A C program is inherently portable, however, certain precautions and requirements should be noted.

- The hardware dependent functions should be strictly separated from the rest of the program, that is from the algorithms (this is often not so easily done).
- The hardware dependent functions should use a standard interface and provide a standard set of services, independently on the special hw features.
- The program should be endian-independent, i.e. bitwise operations should be written carefully in a general manner to allow easy little/big endian conversion.
- The program should use special integer data types (provided by the *types* library) that have hw independent bit length (note that a "standard" integer bit length may vary depending on the hw).
- Any special platform-dependent compiler features should be avoided.

Lecture Summary – Essential Things to Remember

- Complex C projects are usually decomposed into multiple files.
- A **makefile** is a batch file containing the compilation and linking directives for a **project**. Basically, a makefile contains the procedural information about how to compile and link a project. **Directions** and **prerequisites** are given for each automatically generated file (**target**), allowing the program **make** to create it.
- A **structure** data type is used to declare an encapsulated, non-homogenous data type.
- A **union** data type is analogous to structure, but memory is allocated only according the length of the longest data item.
- A **enum** data type is a user-defined index data type.
- The **inline functions** are special functions, which are not compiled as subroutines, but are directly copied by the preprocessor to the places where the function is being called.