Softprocessor SP4: Arrays, Pointer and Bit-manipulation

July 10, 2011

Abstract

Contents of this lecture are arrays, pointer and bit-manipulation.

1 Arrays

An array is a summary of same elements. The declaration is similar to a single element, but beside the variable name the element number is given:

```
char str[10]; // array with 10 signed 8 bit numbers
int Liste[20];// array with 20 signed 32 bit umbers
```

The compiler reserves a continous area in the memory. Single array elements will be addressed by an index (Abb. 1). The element address is calculated always as follows:

$$a\,(i) = a\,(0) + k \cdot i$$

(i – index; $a\,(i)$ – adress of element i; k – byte number of an array element).

A string will be saved as an array of characters (8 bit numbers). After the last character the character value »0« is the end identifier. When working with strings always take care of one aspect: the string has to be shorter than the array which should contain them. Otherwise write and read operations will overwrite or read bytes after the array which maybe contain other information.

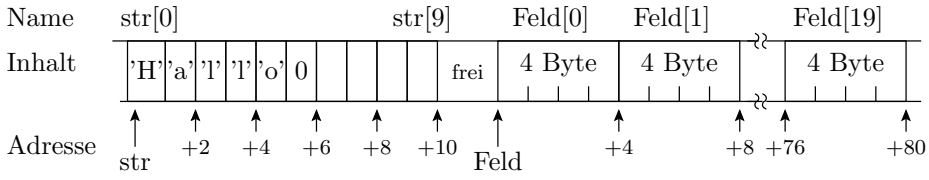

Figure 1: Arrangement of arrays in the memory (Name = name, Inhalt = content, Feld = array element, Adresse = address, frei = empty)

Aufgabe 4.1: String mirroring

Write a main program which declares a string variable with the length of 100. In an infinite loop always the following should happen:

- waiting for the input of a line closed with a line break,
- returning of all characters to the PC (aka echo), writing of the characters into the string variable,
- if there is a line break the string shall be closed with the character value 0 [1] and
- the length of the string followed by an double point and the content of the memory shall be returned mirrored to the PC.

Test example:

Input:	`123␣Hallo␣Welt!␣↩`
Output:	`123␣Hallo␣Welt!␣` `15:␣!tleW␣ollaH␣321`

Aufgabe 4.2: Array sorting

Write a main program which declares an array for unsigned 16 bit numbers with 20 elements. In the infinite loop always the following should happen:

- waiting for the input of maximal 20 number sequences seperated by spaces ending by line break. The number range is from 0 to$2^{16} - 1 = 65535$.
- converting of the numbers with the function »term_read_unsigned« from the file »utils.c« from the lecture before in number values and saving them one after another in the array
- sorting the numbers saved in the array in ascending oder with Bubble Sort and
- converting the sorted numbers one after another with the function »term_write_unsigned« from thr file »utils.c« in character sequences and returning them seperated with spaces to the PC.

Test example:

Input:	`123␣11␣17␣122␣87␣12↩`
Output:	`11␣12␣17␣87␣122␣123`

Advices:

[1] The characters of the line break shall not copied to the string.

- It is allowed to modify your function »term_read_unsigned« from »utils.h« so that it returns the input sequence as echo.
- You need two additional counter variables, such as one for the number of saved values.
- Bubble Sort consists of two nested loops. The inner loop runs from the first to the before last element of the array, reads the selected and the following element, compares them and changes them if the following element is greater than the actual. The outer loop repeates the inner loop as long as there are changes inside. Figure 2 shows the sorting sequence for the example.

2 Pointer

Variables, constants, but also functions are all data objects in the memory arrangened from a certain addresses.. A Pointer is a variable which contains an address. At declaration a type of values on which the pointer is allowed to point is declared, e. g. »int«:

```
int *ptr, *ptr1;
```

(* – operator for »value of a pointer«). The address (the value of the pointer) of a variable will created with »&«. For efficient programming when working

	123	↔	11						
	11		123	↔	17				
			17		123	↔	122		
					122		123	↔	78
							78		123 ↔ 12
Bubble 1: 5× tauschen									12 123
	11	→	17	→	122	↔	78		
					78		122	↔	12
Bubble 2: 2× tauschen							12		122 → 123
	11	→|	17	→	78	↔	12		
Bubble 3: 1× tauschen					12		78	→	122 → 123
	11	→	17	↔	12				
Bubble 4: 1× tauschen			12		17	→	78	→	122 → 123
	11	→	12	→	17	→	78	→	122 → 123
Bubble 5: kein Tausch, Abbruch äußere Schleife									

Figure 2: Sorting sequence for the example (tauschen = changing, Bubble 5: no change, break of the outer loop)

with arrays pointer will be used. As follows an array with a pointer pointing on the address of the first array element is declared:

```
int Feld[20];
ptr = &Feld[0]; // identical with ptr = Feld
```

The array name without index value is identical to the pointer value from the beginning of the array. For calculation of the address from other array elements the index value will be added to the address of the array beginning:

```
ptr1 = ptr + 4; // addition of 4×element size
```

In the example the elements are from type »int«, so the size is 4 byte. The address increases by 16 (byte storing spaces) (Abb. 1). To get the value of element 4 »&(ptr + 4)«, »&(Feld+4)« or »Feld[4]« are possible.

String variables are arrays with characters (char) as elements. The values will be saved as sequences of character values ending with a real zero as ending delimiter (Abb. 1). Loops of the type »Repeat for every character« will be programmed as »Repeat, until the chracter value is zero«. In the following example the string constant »Text«, one string variable »Feld« and two pointers are declared. The pointer gets the starting addresses of both arrays. In the loop every character value from »Text« will be copied to the same position in »Feld« until the pointer from »Text« points on the value »0«. After the loop the additional ending zero will be added to the copied string in the string variable »Feld«:

```
char Text = "Hallo";
char Feld[20], *dest, *src;
src  = Text;
dest = Feld;
while(&src != 0)
{
 *dest = *src;
 src++;
 dest++;
}
*dest = 0;
```

Parameter passing to subprograms up to now was as readable »values«. The values were copied to local variables of the subprogram. Value assignments of the local variables don't change the variable values of the calling program. The only return value is the function value returned with the return statement. In the same kind as before the values, also pointer variables of data objects can be passed. With the passing of addresses the subprogram gets the possibilty to access data objects of the calling program read- and writeable. With the keyword »const« the access is limited to »read only« again. The following subprogram for copying of two strings gets two pointer passed, one on the beginning of the array from the data source and one on the beginning of the data destination.

It copies like the example before, all characters including the ending zero from the character array the pointer »src« is pointing on, into the array »dest« is pointing on:

```
char *strcpy(char *dest, const char *src)
{
 while(&src != 0)
 {
  *dest = *src;
  src++;
  dest++;
 }
 *dest = 0;
 return dest;
}
```

Aufgabe 4.3: Basic functions for text processing

Write a program to the following header »string.h« with these functions for text processing:

```
#ifndef STRING_H
#define STRING_H
char *strcpy(char *dest, const char *src);
char *strcat(char *dest, const char *src);
int strlen(const char *str);
char *append_unsignd(char *dest, int Zahl);
void gets(char *str);
void puts(char *str);
#endif
```

The function »strcpy« was explained before. The fuction »strcat« should concatenate the source string with the destination string. In the first loop the pointer from the destination string has to set on the ending zero. For the following copy process in he easiest case »strcpy« with the destination counter on the ending zero will be called. The function »strlen« should return the length of the string. The function »append_unsigned« should return the passed number value converted in a string and copy it characterwise at the end of the string. The function »gets« should write the via UART received characters one after another in the character array »str« is pointing on. If an line break is received the string should be closed with »0« and the function should return to the calling program. The function »puts« should send the string »str« is pointing on bytewise via UART. If the closing zero is reached the function should send a line break and end itself.

The test frame is the usual main program with infinite loop. In the infinite loop after the request of two input texts, both strings shall be written one behind the other in one string. The constant text »Length:« and the text representation of the text length, determined with »strlen«, shall be attached:

```
...
char *Feld[100], tmp[80];
while(1)
{
 puts("Inputtext 1: \n");
 gets(tmp);
 strcpy(Feld, tmp);
 puts("Inputtext 2: \n");
 gets(tmp);
 strcat(Feld, tmp);
 strcat(Feld, " Text length: ");
 append_unsigned(Feld, strlen(Feld));
 puts(Feld);
}
```

Example:

Inputtext 1:	The␣sun␣↩
Inputtext 2:	shines␣bright.↩
	The␣sun␣shines␣bright. text length: 22

Check questions:

- What happens with the implemented example functions if the string, generated with »strcpy« or »strcat« is longer than the array, in which it is saved?
- How have the function to be completed, to avoid this error?

Aufgabe 4.4: Pointer and references

Write a main program starting with the following declarations and value assignments:

```
int main()
{
  unsigned int x, y, z;
  x = 12; y = 5; z = 88;
  ...
```

and returning the following output text after every character input:

```
variable | value | address
--------------------------
    x    |   ?   |    ?
    y    |   ?   |    ?
    z    |   ?   |    ?
```

Instead of »?« the value and the address of the variable shall be written.

3 Bit manipulation

At input, output and other programming tasks close to the hardware often single bits or parts of bit vectors are handled. Therefore are bitwise logic operations and shifting operations:

negation	AND	OR	EXOR	shift left	shift right
~a	a & b	a | b	a ^ b	a << n	a >> n

(a, b – constants or variables with the same type; n – shifting in bit).

To switch on the LED with the number $n \in \{0, 1, \ldots 7\}$ a variable for the output value is needed. To this value the by n places left shifted »1« has to be OR-interconnected and the resulting value is the output to the LEDs:

```
char LED;
int n;
...
LED = LED | (1 << n); // setting of Bit n
XGpio_mWriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0, LED);
```

For inverting, the OR-interconnection will be replaced by an EXOR-interconnection and for switching off the AND-interconnection will be replaced by the negated value:

```
LED = LED ^ (1 << n); // inverting of bit n
LED = LED & ~(1 << n); // deleting of bit n
```

Aufgabe 4.5: Bit manipulation

Write a program with an infinite loop waiting for »E« or »A« from the UART followed by a digit between »0« and »7« to switch the corresponding LED on or off.

Test example:

Testschritt	Eingabe	Leuchtdioden
0		○○○○○○○○
1	E2	○○○○○●○○
2	E5	○○●○○●○○
3	E4	○○●●○●○○
4	A5	○○○●○●○○
5	E7	●○○●○●○○
6	A2	●○○●○○○○

(Testschritt = Step, Eingabe = Input, Leuchtdioden = LEDs)