George S. Tselikis | Nikolaos D. Tselikas

C From Theory to Practice

Book's Indicative Exercises

Book's Indicative Exercises

S.1 What does the following program do?

```
#include <stdio.h>
int main(void)
{
	unsigned char ch = 3;

	ch = ((ch&1) << 7) | ((ch&2) << 5) | ((ch&4) << 3) | ((ch&8) << 1) | ((ch&16) >> 1) | ((ch&32) >> 3) | ((ch&64) >> 5) | ((ch&128) >> 7);
	printf("%d\n", ch);
	return 0;
}
```

S.2 Write a program that reads an annual income and calculates the tax according to the following table. There is a restriction; don't use `if` statement.

Income	Tax Rate (%)
0-5.000	0
5.001-20.000	15
> 20.000	30

For example, if the user enters `23000`, the tax is calculated as: `tax = (20000-5000)*0.15 + (23000-20000)*0.3`.

S.3 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
	int i = 10, j = 20, k = 0;

	if(i = 40)
		printf("One ");
	if(j = 50)
		printf("Two ");
	if(k = 60)
		printf("Three ");
	if(k = 0)
		printf("Four ");
	printf("%d %d %d\n", i, j, k);
	return 0;
}
```

```
2
```

S.4 Suppose that two PCs reside in the same IP network. Write a program that reads their IP addresses (version 4) and displays if they are configured correctly in order to communicate. The form of the IP address is `x.x.x.x`, where each `x` is an integer within [0, 255]. The value of the first octet of an IP address defines its class, as follows:

a. Class A: [0, 127]
b. Class B: [128, 191]
c. Class C: [192, 223]

If the two IP addresses indicate different classes, the PCs cannot communicate. If they belong in the same class, we compare their network octet(s). The octet(s) to be compared are defined according to their class, as follows:

a. Class A: first octet
b. Class B: first two octets
c. Class C: first three octets

If they are the same the PCs may communicate. For example, the PCs with IP addresses 192.168.1.1 and 192.168.1.2 may communicate, because they belong in the same class C and the first three octets that specify the network, that is, 192.168.1, are the same.

S.5 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
	int i, a[] = {10, 20, 30, 40, 50};
	double b[] = {2.2, 1.94, 0.5, -1, -2};

	for(i = 0; a[i] = b[i]; i++)
		printf("%d ", a[i]);
	return 0;
}
```

S.6 Write a program that simulates an on-line lottery game. Suppose that the winning numbers are `10` and their values are within [0, 100]. However, the program should be written in a way to control the winning numbers. In particular, the "cheat" is that `3` of the winning numbers in each lottery should have also been drawn in the previous one. The program should ask the user to play, and if the answer is no, the program terminates.

S.7 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
	int i;

	for(i = 1; i < 10; i++)
	{
		switch(i)
		{
			case 4:
			break;

			default:
				if(i == 3)
					break;
				else
					continue;
			printf("* ");
			break;
		}
		if(i == 6)
			break;
	}
	return 0;
}
```

S.8 Write a program that reads products' codes and stores them in an integer array of `50` places. The program should store a code in the array only if it is not already stored. If the array becomes full or the user enters `-1`, the program should display the stored codes and terminate.

S.9 Write a program that reads `6` integers and stores them in a `2×3` array (e.g., `a`). Then, it should read another `6` integers and store them in a second `3×2` array (e.g., `b`). The program should display the elements of a third `2×2` array (e.g., `c`), which is the result of `c = a×b`.

It is reminded from the linear algebra that the product of two matrices is produced by adding the products of the elements of each row of the first matrix with the corresponding elements of each column of the second matrix. Therefore, the outcome of a `N×M` matrix multiplied with a `M×N` matrix is a `N×N` matrix. For example, consider the following `a` (`2×3`) and `b` (`3×2`) matrices:

$$a = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 2 & 1 \end{bmatrix} \text{ and } b = \begin{bmatrix} 1 & 0 \\ 2 & -2 \\ 2 & 3 \end{bmatrix}$$

The dimension of `c = a×b` is `2×2`, and its elements are:

$$c = \begin{bmatrix} 1\times1+(-1)\times2+1\times2 & 1\times0+(-1)\times(-2)+1\times3 \\ 0\times1+2\times2+1\times2 & 0\times0+2\times(-2)+1\times3 \end{bmatrix} = \begin{bmatrix} 1 & 5 \\ 6 & -1 \end{bmatrix}$$

In math, the value of its element is the outcome of $c_{ij} = \sum_{k=1}^{M} a_{ik} \times b_{kj}$. Of course, in programming level, `k` ranges from `0` to `M-1`.

S.10 Is the following program correct? If yes, what does it output? If there is an error, could you think of a side-effect?

```
#include <stdio.h>
int main(void)
{
	int i, a[] = {i, i+1, i+2};

	for(i = 0; i < 4; i++)
	{
		a[i] = 0;
		printf("%d ", a[i]);
	}
	return 0;
}
```

S.11 Use the `ptr` pointer and complete the following program to read and store the grades of 50 students into `arr` and then display the array's values in reverse order. Use pointer arithmetic to process the array.

```
#include <stdio.h>

#define SIZE 50

int main(void)
{
	float *ptr, arr[SIZE];
	...
}
```

S.12 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
	int *ptr1, *ptr2, i = 10, j = 20;

	ptr1 = &i;
	*ptr1 = 150;

	ptr2 = &j;
	*ptr2 = 50;

	ptr2 = ptr1;
	*ptr2 = 250;

	ptr1 = ptr2;
	*ptr1 += *ptr2;
	printf("%d\n", i+j);
	return 0;
}
```

S.13 Use the `p1` and `p2` pointers and complete the following program to read the codes of 100 products and store them into `arr`. The program should store a code in the array only if it is not already stored. If the user enters `-1`, the insertion of codes should terminate. The program should display the codes of the stored products, before it terminates. Use pointer arithmetic to process the array.

```
#include <stdio.h>

#define SIZE 100

int main(void)
{
	int *p1, *p2, arr[SIZE];

	...
}
```

S.14 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
	int a[] = {0, 0, 1, 2, 3}, b[] = {0, 0, 4, 5, 6};
	int *ptr1 = a, *ptr2 = b;

	while(!*ptr1++ && !*ptr2++)
```

```
            ;
      printf("%d %d\n", *(b+(ptr1-a)), *(a+(ptr2-b)));
      return 0;
}
```

S.15 Write a program that assigns random integers to a `3×5` array and displays the minimum and the maximum value of each row and column. Use pointer arithmetic to process the array.

S.16 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
      char str1[] = "test", str2[] = "test";

      (str1 == str2) ? printf("One\n") : printf("Two\n");
      return 0;
}
```

S.17 Write a program that reads two characters and displays the characters between them. For example, if the user enters `af` or `fa`, the program should display `bcde`.

S.18 Write a program that displays all lowercase letters in one line, all uppercase letters in a second line, and all characters that represent the digits `0-9` in a third line. Use a single **`for`** loop.

S.19 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
      char *p, *q, s[] = "play";

      p = s+1;
      q = s;
      p[1] = 'x';
      *s = 'a';
      printf("%d %c\n", *q+2, *(q+2));
      return 0;
}
```

S.20 What is the output of the following program?

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

int main(void)
{
	char str[] = "Text", *p = str;
	int i;

	for(i = 0; i < strlen(str)-1; i++, p++)
		printf("%c", p[i]);
	return 0;
}
```

S.21 What is the output of the following program?

```
#include <stdio.h>
#include <string.h>
int main(void)
{
	char str[] = "noteasy";

	printf("%s\n", str+(*(str+3)-1)-str[strlen(str+3)]);
	return 0;
}
```

S.22 Write a program that simulates the popular word-guessing game "hangman". The program reads a secret word of less than 30 characters, which is the word to guess. This word is entered by the first player. Then, the second player enters letters one by one. The program should check each letter and inform the player. If the letter occurs in the word, the program should display the letters of the correct guesses in its positions and set the underscore character in the positions of the unknown letters. If it does not occur, the program should display a related message. The game is over if either the player finds the word or 7 incorrect guesses are made. Assume that the second player is not allowed to enter the same letter more than once.

S.23 What is the output of the following program?

```
#include <stdio.h>
#include <string.h>
int main(void)
{
	char str[] = "test";

	printf("%d %s\n", *strcpy(str, "n")**strcpy(str+2, "xt"),
str);
	return 0;
}
```

S.24 The data compression algorithm RLE (Run Length Encoding) is based on the assumption that a symbol within the data stream may be repeated many times in a row. This repetitive sequence can be replaced by an integer that declares the number of the repetitions and the symbol itself. Write a program that reads a string of less than 100 characters and uses the RLE algorithm to compress it. Don't compress digits and characters that appear once.

For example, the string: `fffmmmm1234jjjjjjjjjx`
should be compressed to: `3f4m123410jx`

S.25 What is the output of the following program?

```
#include <stdio.h>

void test(int *arg);

int var = 100;

int main(void)
{
	int *ptr, i = 30;

	ptr = &i;
	test(ptr);
	printf("%d\n", *ptr);
	return 0;
}

void test(int *arg)
{
	arg = &var;
}
```

S.26 Write a **void** function that takes as parameters an array that contains the prices of some products in a shop and their number and uses proper variables to return the lowest, the highest, and the average of the prices. Write a program that reads the prices of less than 100 products and stores them in an array. If the user enters `-1`, the insertion of prices should terminate. The program should use the function to display the lowest, the highest, and the average of the prices.

S.27 What does the following program do?

```
#include <stdio.h>

int unknown(int num1, int num2);
```

```
int main(void)
{
	int num1, num2, sign;

	printf("Enter numbers: ");
	scanf("%d%d", &num1, &num2);

	sign = 1;
	if((num1 < 0) && (num2 > 0))
	{
		num1 = -num1;
		sign = -1;
	}
	else if((num1 > 0) && (num2 < 0))
	{
		num2 = -num2;
		sign = -1;
	}
	else if((num1 < 0) && (num2 < 0))
	{
		num1 = -num1;
		num2 = -num2;
	}
	if(num1 > num2)
		printf("%d\n", sign*unknown(num1, num2));
	else
		printf("%d\n", sign*unknown(num2, num1));
	return 0;
}

int unknown(int n1, int n2)
{
	if(n2 == 1)
		return n1;
	else
		return n1 + unknown(n1, n2-1);
}
```

S.28 Write a program that reads the names of 50 countries (less than 100 characters each) and the number of tourists visited them on monthly basis. The program should use proper arrays to store the data. Then, the program should read the name of a country and display the annual number of tourists who visited that country. The program should display the 5 most visited countries (check if more than one country ties in the fifth place) before it ends. Use the bubble sort algorithm to sort the arrays.

Indicative Exercises

S.29 Define the structure type `book` with members: title, code, and price. Write a function that takes as parameters two pointers to structures of type `book` and return the structure with the higher price. Examine the case of the same prices and write a related message. Write a program that reads and stores the data of `10` books into an array of structures. Then, the program should read two numbers in [1, 10] that represent two structure indexes in the array. The program should check the validity of the input numbers and use the function to display the data of the structure with the higher price.

S.30 Image-editing programs often need to rotate an image by 90°. An image can be treated as a two-dimensional array whose elements represent the pixels of the image. For example, the rotation of the original image (e.g., `p[M][N]`) to the right produces a new image (e.g., `r[M][N]`), as shown here:

$$p = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,N-2} & p_{0,N-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{M-2,0} & p_{M-2,1} & \cdots & p_{M-2,N-2} & p_{M-2,N-1} \\ p_{M-1,0} & p_{M-1,1} & \cdots & p_{M-1,N-2} & p_{M-1,N-1} \end{bmatrix} \quad r = \begin{bmatrix} p_{M-1,0} & p_{M-2,0} & \cdots & p_{1,0} & p_{0,0} \\ p_{M-1,1} & p_{M-2,1} & \cdots & p_{1,1} & p_{0,1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{M-1,N-2} & p_{M-2,N-2} & \cdots & p_{1,N-2} & p_{0,N-2} \\ p_{M-1,N-1} & p_{M-2,N-1} & \cdots & p_{1,N-1} & p_{0,N-1} \end{bmatrix}$$

In particular, the first row of the original image becomes the last column of the new image, the second row becomes the last but one column, up to the last row, which becomes the first column. For example, the image:

$$p = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix} \quad \text{is transformed to:} \quad r = \begin{bmatrix} 11 & 6 & 1 \\ 12 & 7 & 2 \\ 13 & 8 & 3 \\ 14 & 9 & 4 \\ 15 & 10 & 5 \end{bmatrix}$$

The color of each pixel follows the RGB color model, in which the red, green, and blue colors are mixed together to reproduce a wide range of colors. The color is expressed as an RGB triplet (r, g, b), in which each component value varies from 0 to 255.

Define the structure type `pixel` with three integer members named `red`, `green`, and `blue`. Write a program that creates a two-dimensional image (e.g., `3×5`) whose elements are structures of type `pixel`. Initialize the members of

each structure with random values within [0, 255]. Then, the program should display the original image, rotate the image by 90 degrees right, and display the rotated image (e.g., `5×3`).

Hint: Use a second array to store the rotated image.

S.31 Write a program that reads its command line arguments and allocates memory to concatenate them into one string and display that string.

S.32 Write a function that takes as parameters two arrays of doubles (e.g., `a1` and `a2`) of the same size and their number of elements, and allocates memory to store the elements of `a1` that are not contained in `a2`. The function should return a pointer to that memory. Write a program that reads pairs of doubles and stores them into two arrays of 100 elements (e.g., `p1` and `p2`). If either input value is `-1`, the insertion of numbers should end. The program should use the function to display the elements of `p1` that are not contained in `p2`.

S.33 Write a program that reads products' codes (less than 20 characters each) and their prices and stores them in a text file, as follows:

```
C101  17.5
C102  32.8
...
```

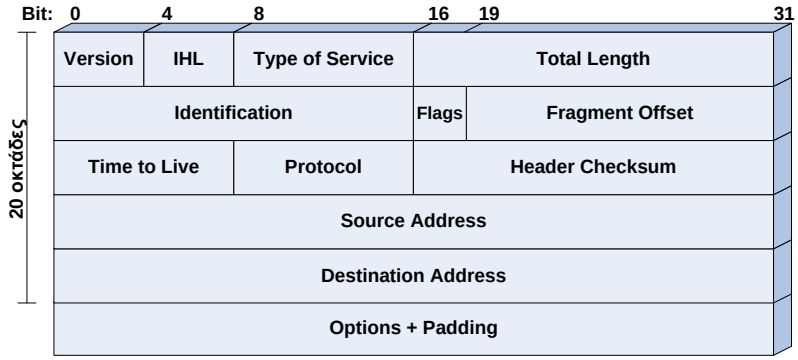
If the user enters `-1` for price, the insertion of products should terminate. Then, the program should read a product's code and search the file to find and display its price.

S.34 Define a structure of type `band` with members: name, category, singer (all less than 100 characters), and records. Suppose that the `test.bin` binary file contains such structures. The number of bands is declared in the beginning of the file. Write a program that reads the file and uses the structure type to store the data in a dynamically allocated memory. Then, the program should read the name of a band, a new singer, and replace the existing singer with the new one.

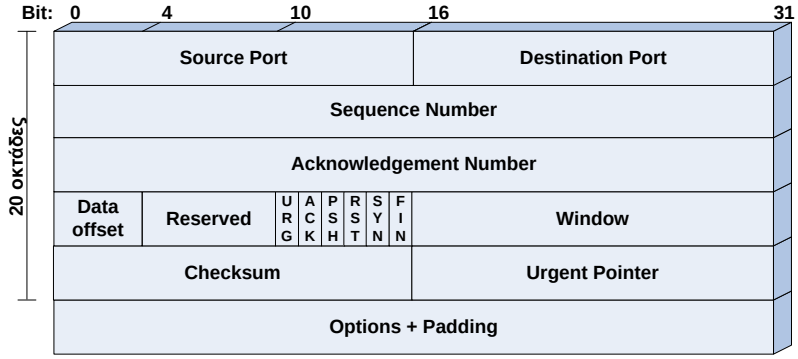
S.35 What is the output of the following program?

```
#include <stdio.h>

#define no_main(type, name, text, num) type name(void)
{printf(text); return num;}

no_main(int, main, "No main()", 0)
```

S.36 In order a system to connect to an Internet address, its network card must transmit an IP packet that encapsulates a special TCP segment. The IPv4 header format is depicted below.



The TCP header format is depicted below.



Write a program that reads the source IP address in `x.x.x.x` format (each `x` is an integer in [0, 255]), the destination IP address in `x.x.x.x` format, the TCP destination port (integer in [1, 65535]) and creates an IP packet that encapsulates the proper TCP segment. The program must store the content of the IP packet in hexadecimal format in a user selected text file. Each line must contain 16 bytes. Set the following values in the IPv4 header:

a. `Version` = `4`.
b. `IHL` = `5`.
c. `Total Length` = total length of the IP packet, including the TCP data.
d. `Protocol` = `6`.
e. `Time to Live` = `255`.

f. `Destination Address` = destination IP address.
g. `Source Address` = source IP address.

Set the following values in the TCP header:

a. `Destination Port` = destination TCP port.
b. `Source Port` = `1500`.
c. `Window` = the maximum allowed value.
d. `SYN bit` = `1`.

Set the rest fields to `0` and assume that there are no `Options` fields.

The program has one restriction; if the destination IP address starts from 130.140 or 160.170 and the TCP destination port is 80, do not create the IP packet and display a related message.

S.37 What is the output of the following program?

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

void test(char a, char str[], char *ptr);

int main(void)
{
	char *tmp, txt[20] = "abcde";

	tmp = txt;
	test(*(tmp+1), txt+3, &txt[1]);
	printf("%s\n", txt);
	return 0;
}

void test(char a, char str[], char *ptr)
{
	strcpy(str, "1234");
	str[0] = a;
	ptr[2] = *str + 5;
}
```

S.38 What is the output of the following program?

```
#include <stdio.h>
int main(void)
{
	char *arr[] = {"TEXT", "SHOW", "OPTIM", "DAY"};
```

```
        char **ptr1;

        ptr1 = arr;
        printf("%s ", *++ptr1);
        printf("%s", *++ptr1+2);
        printf("%c\n", **++ptr1+1);
        return 0;
}
```

S.39 Write a program that simulates a cinema's ticket office. Assume that the cinema has 30 rows of 20 seats each. The program should display a menu to perform the following operations:

1. Buy a ticket. The program should let the spectator to select the row and the seat. If no specific seat is asked, the program should select a random seat. The ticket's price is $6.
2. Ticket cancellation. The program should read the row and the seat and cancel the reservation. The refund is $5.
3. Display the total income and a diagram to show the reserved and free seats.
4. Program termination.

Indicative Answers

S.1 **Answer:** Suppose that `ch` is coded in binary as $x_8x_7x_6x_5x_4x_3x_2x_1$, where each x_i is either `1` or `0`. Therefore, the value of the expression `(ch&1) << 7` is:

```
   x8x7x6x5x4x3x2x1
&  0 0 0 0 0 0 0 1
   -----------------
   0 0 0 0 0 0 0 x1 << 7 = x1 0 0 0 0 0 0 0
```

Similarly, the value of `(ch&2) << 5` is:

```
   x8x7x6x5x4x3x2x1
&  0 0 0 0 0 0 1 0
   -----------------
   0 0 0 0 0 0 x20  << 5 = 0 x2 0 0 0 0 0 0
```

In a similar way:

The value of `(ch&4) << 3` is: `0 0 x3 0 0 0 0 0`
The value of `(ch&8) << 1` is: `0 0 0 x4 0 0 0 0`
The value of `(ch&16) >> 1` is: `0 0 0 0 x5 0 0 0`
The value of `(ch&32) >> 3` is: `0 0 0 0 0 x6 0 0`
The value of `(ch&64) >> 5` is: `0 0 0 0 0 0 x7 0`
The value of `(ch&128) >> 7` is: `0 0 0 0 0 0 0 x8`

Therefore, the entire expression is evaluated as:

```
(x1 0 0 0 0 0 0 0) | (0 x2 0 0 0 0 0 0) | (0 0 x3 0 0 0 0 0) |
(0 0 0 x4 0 0 0 0) | (0 0 0 0 x5 0 0 0) | (0 0 0 0 0 x6 0 0) |
(0 0 0 0 0 0 x7 0) | (0 0 0 0 0 0 0 x8) = x1x2x3x4x5x6x7x8
```

In fact, the program reverses the bits of `ch`, without using a second variable. Therefore, `ch` (00000011_2) becomes `192` (11000000_2), and the program displays `192`.

S.2 **Answer:**

```
#include <stdio.h>
int main(void)
{
      float tax, a;
```

```
	printf("Enter income: ");
	scanf("%f", &a);

	tax = (a > 5000 && a <= 20000)*(a-5000)*0.15 + (a > 
20000)*((a-20000)*0.3 + 15.000*0.15);
	printf("Tax = %.2f\n", tax);
	return 0;
}
```

S.3 **Answer:** This is an example of how the wrong use of the = operator instead of the == operator can produce unexpected results. In particular, the **if** conditions don't test the variables for equality, but the variables are assigned with the respective values instead. As a result, `i` becomes `40`, `j` becomes `50`, while `k` first becomes `60`, and then `0`. Since the first three **if** conditions are true and the fourth is false (since `0` is assigned to `k`), the program displays: `One Two Three 40 50 0`

S.4 **Answer:**

```
#include <stdio.h>
int main(void)
{
	int a1, a2, a3, a4, b1, b2, b3, b4;

	printf("Enter first IP address: ");
	scanf("%d.%d.%d.%d", &a1, &a2, &a3, &a4);

	printf("Enter second IP address: ");
	scanf("%d.%d.%d.%d", &b1, &b2, &b3, &b4);

	if(a1 < 128)
	{
		if(a1 == b1)
			printf("Class A: Correct Configuration\n");
		else
			printf("Class A: Wrong Configuration\n");
	}
	else if(a1 < 192)
	{
		if(a1 == b1 && a2 == b2)
			printf("Class B: Correct Configuration\n");
		else
			printf("Class B: Wrong Configuration\n");
	}
	else if(a1 < 224)
	{
		if(a1 == b1 && a2 == b2 && a3 == b3)
			printf("Class C: Correct Configuration\n");
```

```
            else
                  printf("Class C: Wrong Configuration\n");
      }
      else
            printf("Error: Wrong class\n");
      return 0;
}
```

Comments: Notice that we put on purpose the dot in `scanf()` so that the user inputs the IP address in its familiar form.

S.5 **Answer:** The condition `a[i] = b[i];` is equivalent to `(a[i] = b[i]) != 0;`, which means that the elements of `b` are copied to the respective elements of `a` as long as `a[i]` does not become `0`. If it does, the loop terminates. Since the type of `a` is **`int`**, only the integer parts of the `b` elements will be stored into the respective `a` elements. Therefore, when the value `0.5` is copied, `a[2]` becomes `0` and the loop terminates. As a result, the program outputs: `2 1`

S.6 **Answer:**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SIZE 10

int main(void)
{
      int i, pos1, pos2, pos3, ans, arr[SIZE];

      srand(time(NULL));
      /* First lottery. */
      for(i = 0; i < SIZE; i++)
      {
            arr[i] = rand()%101;
            printf("%d ", arr[i]);
      }
      while(1)
      {
            printf("\nContinue to play? (0:No): ");
            scanf("%d", &ans);
            if(ans == 0)
                  return 0;
            /* We choose three random places. The numbers stored
in these places will be drawn in the next lottery. */
            pos1 = rand()%SIZE;
            do
            {
```

```
                pos2 = rand()%SIZE;
            } while(pos1 == pos2);
            do
            {
                pos3 = rand()%SIZE;
            } while((pos1 == pos3) || (pos2 == pos3));

            /* Next lottery. */
            for(i = 0; i < SIZE; i++)
            {
                if((i == pos1) || (i == pos2) || (i == pos3))
                {
                    printf("%d ", arr[i]);
                    continue;
                }
                arr[i] = rand()%101;
                printf("%d ", arr[i]);
            }
        }
        return 0;
}
```

Comments: The primary goal of this program is to show you that a betting software can be written in a way to manipulate the winning results. Therefore, **stay away** from betting sites that advertise big profits in on-line lotteries. The big profits they promise are not for you, but for their owners.

S.7 **Answer:** Unlike **break**, the **continue** statement does not apply to **switch**. A **continue** statement inside a **switch**, which is inside a loop triggers the next loop iteration. Since the **continue** statement causes the next iteration to begin, the program displays nothing.

S.8 **Answer:**

```
#include <stdio.h>

#define SIZE 50

int main(void)
{
        int i, j, pos, num, found, code[SIZE];

        pos = 0;
        while(pos < SIZE)
        {
                printf("Enter code: ");
                scanf("%d", &num);
                if(num == -1)
                        break;
```

```
            found = 0;
            /* The pos variable indicates how many codes have
been stored in the array. The loop checks if the input code is
already stored. If it is, found becomes 1 and the loop
terminates. */
            for(j = 0; j < pos; j++)
            {
                  if(code[j] == num)
                  {
                        printf("Error: Code %d exists. ", num);
                        found = 1;
                        break;
                  }
            }
            /* If the code is not stored, we store it and the
index position is incremented. */
            if(found == 0)
            {
                  code[pos] = num;
                  pos++;
            }
      }
      printf("\nCodes: ");
      for(i = 0; i < pos; i++)
            printf("%d ", code[i]);
      return 0;
}
```

S.9 Answer:

```
#include <stdio.h>

#define N 2
#define M 3

int main(void)
{
      int i, j, k, a[N][M], b[M][N], c[N][N] = {0};
      for(i = 0; i < N; i++)
      {
            for(j = 0; j < M; j++)
            {
                  printf("Enter element a[%d][%d]: ", i, j);
                  scanf("%d", &a[i][j]);
            }
      }
      for(i = 0; i < M; i++)
      {
            for(j = 0; j < N; j++)
```

```
            {
                  printf("Enter element b[%d][%d]: ", i, j);
                  scanf("%d", &b[i][j]);
            }
      }
      for(i = 0; i < N; i++)
            for(j = 0; j < N; j++)
                  for(k = 0; k < M; k++)
                        c[i][j] += a[i][k] * b[k][j];

      printf("\nArray c = a x b (%dx%d)\n", N, N);
      printf("-----------------\n");
      for(i = 0; i < N; i++)
      {
            for(j = 0; j < N; j++)
                  printf("%5d", c[i][j]);
            printf("\n");
      }
      return 0;
}
```

S.10 **Answer:** Did you answer that the initialization of `a` array is wrong? Nothing is wrong there; `i` has an arbitrary value, so what? The error is the number of repetitions. Since `a` contains three elements, it should be `3`, not `4`. In particular, when `i` becomes `3`, the statement `a[i] = 0;` assigns a value in a memory location out of `a`. If `i` is stored in that location, `i` becomes `0` and the loop becomes infinite.

Be careful, and be careful again with the array bounds, if you want to have a peaceful sleep.

S.11 **Answer:**

```
#include <stdio.h>

#define SIZE 50

int main(void)
{
      float *ptr, arr[SIZE];

      ptr = arr;
      while(ptr < arr+SIZE)
      {
            printf("Enter grade: ");
            scanf("%f", ptr);
            ptr++;
      }
      ptr--;
```

```
	while(ptr >= arr)
	{
		printf("%f\n", *ptr);
		ptr--;
	}
	return 0;
}
```

S.12 **Answer:** Since `ptr1` points to the address of `i`, the statement `*ptr1 = 150;` is equivalent to `i = 150;` Similarly, the statement `*ptr2 = 50;` is equivalent to `j = 50;` The statement `ptr2 = ptr1;` makes `ptr2` point to the same address that `ptr1` points to, that is, the address of `i`. Therefore, the statement `*ptr2 = 250;` is equivalent to `i = 250`.

The next statement does not change the value of `ptr1`, that is, it still points to the address of `i`. Therefore, the statement `*ptr1 += *ptr2;` is equivalent to `i += i`, that is, `i = i+i = 250+250 = 500`. As a result, the program outputs `550`.

S.13 **Answer:**

```
#include <stdio.h>

#define SIZE 100

int main(void)
{
	int *p1, *p2, arr[SIZE];

	p1 = arr;
	while(p1 < arr+SIZE)
	{
		printf("Enter code_%d: ", p1-arr+1);
		scanf("%d", p1);
		if(*p1 == -1)
			break;

		for(p2 = arr; p2 < p1; p2++) /* Traverse the array
to check if the input code is already stored. */
		{
			if(*p1 == *p2)
			{
				printf("Error: Code %d exists\n", *p1);
				break;
			}
		}
		/* If the code is not stored, increase the pointer.
*/
		if(p2 == p1)
```

```
                p1++;
        }
        /* Display the codes. */
        for(p2 = arr; p2 < p1; p2++)
                printf("C: %d\n", *p2);
        return 0;
}
```

S.14 **Answer:** Yes, you need pen and paper to decipher it. It is a tough one, indeed; let's trace the iterations:

First iteration. Notice that in `!*ptr1++` the `!` operator is applied first and then `ptr1` is increased. Since `ptr1` points to `a`, `*ptr1` is equal to `a[0]`, that is, `0`. The `!` operator makes it `1`. Next, `ptr1` is increased and points to `a[1]`. Similarly, the value of `!*ptr2` is `1` and `ptr2` points to `b[1]`. Since both terms are true, the loop continues.

Second iteration. Like before, the values of `!*ptr1` and `!*ptr2` are `1`. `ptr1` points to `a[2]` and `ptr2` points to `b[2]`.

Third iteration. Since `ptr1` points to `a[2]`, `!*ptr1` is `0` and `ptr1` is increased and points to `a[3]`. Here is the key to the answer. Recall from Chapter 4 and the description of the `&&` operator that if an operand is false the rest operands are not evaluated and the value of the expression becomes `0`. Therefore, the loop terminates. Since the `!*ptr2++` term is not evaluated, `ptr2` is not increased.

Since `ptr1` points to `a[3]` the result of `ptr1-a` is `3`. Therefore, the expression `*(b+(ptr1-a))` is equivalent to `*(b+3)`, that is, `b[3]`. Similarly, since `ptr2` points to `b[2]` the expression `*(a+(ptr2-b))` is equivalent to `*(a+2)`, that is, `a[2]`. Therefore, the program outputs: `5  1`

And to make it even worse, what would be the output if we replace the `&&` operator with the `||` operator and write:

```
while(!*ptr1++ || !*ptr2++);
```

If you find it, please let us know…

S.15 **Answer:**

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

```
#include <stdlib.h>
#include <time.h>

#define ROWS 3
#define COLS 5

int main(void)
{
	int i, j, min, max, arr[ROWS][COLS];

	srand(time(NULL));
	for(i = 0; i < ROWS; i++)
	{
		min = max = *arr[i] = rand();
		for(j = 1; j < COLS; j++)
		{
			*(arr[i]+j) = rand();
			if(*(arr[i]+j) < min)
				min = *(arr[i]+j);
			if(*(arr[i]+j) > max)
				max = *(arr[i]+j);
		}
		printf("Row_%d: Min=%d Max=%d\n", i+1, min, max);
	}
	for(i = 0; i < COLS; i++)
	{
		min = max = *(arr[0]+i);
		for(j = 1; j < ROWS; j++)
		{
			if(*(arr[j]+i) < min)
				min = *(arr[j]+i);
			if(*(arr[j]+i) > max)
				max = *(arr[j]+i);
		}
		printf("Col_%d: Min=%d Max=%d\n", i+1, min, max);
	}
	/* Display the array to verify the results. */
	for(i = 0; i < ROWS; i++)
	{
		for(j = 0; j < COLS; j++)
			printf("%10d", *(arr[i]+j));
		printf("\n");
	}
	return 0;
}
```

S.16 **Answer:** Since the names of the two arrays are used as pointers, the expression `str1 == str2` checks if the pointers point to the same address, not if the arrays are the same. Do `str1` and `str2` point to the same address?

Of course not; `str1` and `str2` have the same content, they are stored in different memory though. Therefore, the program displays `Two`.

What would be the output if we write:

```
(*str1 == *str2) ? printf("One\n") : printf("Two\n");
```

Since `str1` can be used as a pointer to its first element, `*str1` is equal to `'t'`. Similarly, `*str2` is equal to `'t'`. Therefore, the program would display `One`.

S.17 Answer:

```
#include <stdio.h>
int main(void)
{
	char ch1, ch2;

	printf("Enter characters: ");
	scanf("%c%c", &ch1, &ch2);

	if(ch1 < ch2)
	{
		ch1++;
		while(ch1 != ch2)
		{
			printf("%c", ch1);
			ch1++;
		}
	}
	else
	{
		ch2++;
		while(ch2 != ch1)
		{
			printf("%c", ch2);
			ch2++;
		}
	}
	return 0;
}
```

S.18 Answer:

```
#include <stdio.h>
int main(void)
{
	char ch, end_ch;
```

```
	end_ch = 'z';
	for(ch = 'a'; ch <= end_ch; ch++)
	{
		printf("%c ", ch);
		if(ch == 'z')
		{
			ch = 'A'-1; /* Subtract 1, so that the ch++ statement in the next iteration makes it 'A'. */
			end_ch = 'Z'; /* Change the end character, so that the loop displays the uppercase letters. */
			printf("\n");
		}
		else if(ch == 'Z')
		{
			ch = '0'-1;
			end_ch = '9';
			printf("\n");
		}
	}
	return 0;
}
```

S.19 **Answer:** Since `p` points to the address of the second element of the `s` array, `s[2]` changes to `'x'`. Also, `s[0]` becomes `'a'`. Because `q` points to `s[0]` and the `*` operator has greater precedence than the `+` operator, we have `*q+2 = 'a'+2`. Therefore, the program outputs the ASCII value of the character two places after `'a'`. This is `'c'` and the program outputs `99`. Since the value of `*(q+2)` is equal to `s[2]`, the program outputs `x`, as well.

S.20 **Answer:** Since `strlen()` returns `4`, the loop is executed three times. Let's trace the iterations:

First iteration (`i = 0`): The value of `p[0]` is displayed, that is, `T`.
Second iteration (`i++ = 1`): Since `p` is incremented by one, `p` points to the second character of the string, that is, `'e'`. Since we handle `p` as an array, `p[0]` is `'e'` and `p[1]` is `'x'`. Therefore, the program outputs `x`.
Third iteration (`i++ = 2`): Now, `p` points to the third character, that is, `'x'`. Therefore `p[0]` is `'x'`, `p[1]` is `'t'` and `p[2]` is equal to `'\0'`. As a result, the program displays nothing.

To sum up, the program displays: `Tx`

S.21 **Answer:** Since the expression `*(str+3)` is equivalent to `str[3]`, that is, `'e'`, the value of `*(str+3)-1` is equal to `str[3]-1`, that is, `'e'-1 = 'd'`. `strlen()` returns the length of the string after the third character, that is, `4`.

Therefore, the expression is equivalent to `str+'d'-'a' = str+3` and the program outputs `easy`.

S.22 Answer:

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

#define ERROR_TRIES 7

int read_text(char str[], int size, int flag);

int main(void)
{
	char ch, secret[30], hide[30] = {0};
	int i, found, len, error, correct;

	printf("Enter secret word: ");
	len = read_text(secret, sizeof(secret), 1);

	for(i = 0; i < len; i++)
		hide[i] = '_';

	error = correct = 0;
	while(error < ERROR_TRIES)
	{
		no_main
		ch = getchar();
		found = 0;
		for(i = 0; i < len; i++)
		{
			if(secret[i] == ch)
			{
				hide[i] = ch;
				found = 1;
				correct++;
				if(correct == len)
				{
					printf("Secret word is found
!!!\n");
					return 0;
				}
			}
		}
		if(found == 0)
		{
			error++;
			printf("Error, %c does not exist. You've got
%d more attempts\n", ch, ERROR_TRIES - error);
```

```
            }
            else
                  printf("%s\n", hide);
            getchar(); /* Get '\n' from the previous getchar().
*/
      }
      printf("Sorry, the secret word was %s\n", secret);
      return 0;
}
```

S.23 **Answer:** The first `strcpy()` copies the characters of the string `"n"`, that is, the characters `'n'` and `'\0'` to `str[0]` and `str[1]`, respectively, and returns a pointer to `str`. Therefore, the expression `*strcpy(str, "n")` can be replaced by `*str`, that is, `'n'`. The second `strcpy()` copies the string `"xt"` in the third position of `str` and returns the `str+2` pointer. Therefore, the expression `*strcpy(str+2, "xt")` can be replaced by `*(str+2)`, that is, `'x'`.

As a result, the program displays the product of the ASCII codes of `'n'` and `'x'`, that is, `13200`. However, it does not display `next` as you might expect, but just `n`, because the first `strcpy()` replaced `'e'` with `'\0'`.

S.24 **Answer:**

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

int read_text(char str[], int size, int flag);

int main(void)
{
      char str[100];
      int i, len, cnt;

      printf("Original text  : ");
      len = read_text(str, sizeof(str), 1);

      printf("Compressed text: ");
      i = 0;
      while(i < len)
      {
            cnt = 1;
            if(str[i] < '0' || str[i] > '9') /* Digits are not
compressed. */
            {
```

```
                while(str[i+cnt] == str[i]) /* Check if the
current character, that is str[i], is repeated in the next
places. */
                        cnt++;
                if(cnt == 1)
                        printf("%c", str[i]);
                else
                        printf("%d%c", cnt, str[i]);
        }
        else
                printf("%c", str[i]);
        i += cnt;
    }
    return 0;
}
```

S.25 **Answer:** That's a tricky one. Since the value of `ptr` and not its address is passed to `test()`, any change in the value of `arg` does not affect `ptr`. Therefore, the program displays `30`.

S.26 **Answer:**

```
#include <stdio.h>

void stat_arr(float arr[], int size, float *min, float *max,
float *avg);

int main(void)
{
    int i;
    float min, max, avg, arr[100];

    for(i = 0; i < 100; i++)
    {
        printf("Enter price: ");
        scanf("%f", &arr[i]);
        if(arr[i] == -1)
            break;
    }
    if(i == 0)
        return 0;
    /* The variable i indicates how many prices were stored
into the array. For example, if the user does not enter the
value -1, i would be equal to 100. */
    stat_arr(arr, i, &min, &max, &avg);
    printf("Max=%.2f Min=%.2f Avg=%.2f\n", max, min, avg);
    return 0;
}
```

```
void stat_arr(float arr[], int size, float *min, float *max,
float *avg)
{
	int i;
	float sum;

	sum = *min = *max = arr[0];
	for(i = 1; i < size; i++)
	{
		if(arr[i] > *max)
			*max = arr[i];
		if(arr[i] < *min)
			*min = arr[i];
		sum += arr[i];
	}
	*avg = sum/size;
}
```

S.27 **Answer:** First, let's see what the function is doing. Suppose that it is called with arguments `10` and `4`. The function returns:

```
n1 + unknown(n1, n2-1 = 3) =
n1 + n1 + unknown(n1, n2-1 = 2) =
n1 + n1 + n1 + unknown(n1, n2-1 = 1)
```

The last call of `unknown(n1, 1)` returns `n1`, because `n2 = 1`. Therefore, the function returns:

```
n1+n1+n1+n1 = 4*n1 = n2*n1
```

The **`if-else-if`** series finds the sign of the product and the absolute values of the integers. Next, the **`if-else`** statement is used, in order to make the less recursive calls. As a result, the purpose of this program is to calculate the product of the input numbers through the use of a recursive function.

S.28 **Answer:**

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

#define CNTRS    50
#define MONTHS   12

void bubble_sort(char str[][100], int arr[]);
int read_text(char str[], int size, int flag);
```

```
int main(void)
{
	char cntr[CNTRS][100], str[100];
	int i, j, tmp, flag, tour[CNTRS] = {0};

	for(i = 0; i < CNTRS; i++)
	{
		printf("Enter name of country_%d: ", i+1);
		read_text(cntr[i], sizeof(cntr[i]), 1);

		for(j = 0; j < MONTHS; j++)
		{
			printf("Enter tourists of month_%d: ", j+1);
			scanf("%d", &tmp);
			tour[i] += tmp; /* This array holds the annual
number of tourists for each country. */
		}
		getchar();
	}
	printf("Enter country to search: ");
	read_text(str, sizeof(str), 1);

	flag = 0;
	for(i = 0; i < CNTRS; i++)
	{
		if(strcmp(str, cntr[i]) == 0)
		{
			flag = 1;
			printf("%d tourists visited %s\n", tour[i],
str);
			break;
		}
	}
	if(flag == 0)
		printf("%s not registered\n", str);

	bubble_sort(cntr, tour); /* Sort the tourist array and
update the countries array in parallel. */
	printf("\n***** Tourists in decrease order *****\n");
	for(i = 0; i < 5; i++)
		printf("%d.%s\t%d\n", i+1, cntr[i], tour[i]);
	/* Check if more than one country ties in the fifth place.
*/
	while((tour[i] == tour[4]) && i < CNTRS)
	{
		printf("%d.%s\t%d\n", i+1, cntr[i], tour[i]);
		i++;
	}
	return 0;
}
```

```
void bubble_sort(char str[][100], int arr[])
{
	char temp[100];
	int i, j, k, reorder;

	for(i = 1; i < CNTRS; i++)
	{
		reorder = 0;
		for(j = CNTRS-1; j >= i; j--)
		{
			if(arr[j] > arr[j-1]) /* Parallel swapping of
the tourist numbers and the respective countries. */
			{
				k = arr[j];
				arr[j] = arr[j-1];
				arr[j-1] = k;

				strcpy(temp, str[j]);
				strcpy(str[j], str[j-1]);
				strcpy(str[j-1], temp);
				reorder = 1;
			}
		}
		if(reorder == 0)
			return;
	}
}
```

S.29 Answer:

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

#define SIZE 10

struct book
{
	char title[50];
	int code;
	float prc;
};

struct book max_prc(struct book *b1, struct book *b2);
int read_text(char str[], int size, int flag);

int main(void)
{
	int i, j;
```

```
	struct book tmp, b[SIZE];

	for(i = 0; i < SIZE; i++)
	{
		printf("\nEnter title: ");
		read_text(b[i].title, sizeof(b[i].title), 1);

		printf("Enter code: ");
		scanf("%d", &b[i].code);

		printf("Enter price: ");
		scanf("%f", &b[i].prc);

		getchar();
	}
	do
	{
		printf("Enter two book numbers [1-%d]: ", SIZE);
		scanf("%d%d", &i, &j);
	} while((i == j) || i > SIZE || j > SIZE || i < 1 || j <
1);

	i--; /* Subtract 1 since indexing starts from 0. */
	j--;
	tmp = max_prc(&b[i], &b[j]);
	if(tmp.prc != 0)
		printf("\nN:  %s  C:  %d  P:  %.2f\n",  tmp.title,
tmp.code, tmp.prc);
	else
		printf("\nBoth books have the same price: %.2f\n",
b[i].prc);
	return 0;
}

struct book max_prc(struct book *b1, struct book *b2)
{
	struct book tmp = {0};

	if(b1->prc > b2->prc)
		return *b1;
	else if(b1->prc < b2->prc)
		return *b2;
	else
		return tmp;
}
```

S.30 Answer:

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

```
#include <time.h>

#define ROWS    3
#define COLS    5

struct pixel /* RGB format (Red-Green-Blue). */
{
      unsigned char red; /* Value in [0, 255]. */
      unsigned char green;
      unsigned char blue;
};

void rotate_right_90(struct pixel img[][COLS], struct pixel
tmp[][ROWS]);

int main(void)
{
      int i, j;
      struct pixel img[ROWS][COLS], tmp[COLS][ROWS];

      srand(time(NULL));
      /* Create random colors. */
      for(i = 0; i < ROWS; i++)
      {
            for(j = 0; j < COLS; j++)
            {
                  img[i][j].red = rand()%256;
                  img[i][j].green = rand()%256;
                  img[i][j].blue = rand()%256;
            }
      }
      printf("*** Original Image ***\n\n");
      for(i = 0; i < ROWS; i++)
      {
            for(j = 0; j < COLS; j++)
            {
                  printf("(%3d,%3d,%3d)     ",     img[i][j].red,
img[i][j].green, img[i][j].blue);
            }
            printf("\n");
      }
      rotate_right_90(img, tmp);

      printf("\n*** Rotated Image ***\n\n");
      for(i = 0; i < COLS; i++)
      {
            for(j = 0; j < ROWS; j++)
            {
                  printf("(%3d,%3d,%3d)     ",     tmp[i][j].red,
tmp[i][j].green, tmp[i][j].blue);
```

```
            }
            printf("\n");
      }
      return 0;
}

void rotate_right_90(struct pixel img[][COLS], struct pixel
tmp[][ROWS])
{
      int i, j, k = 0;

      for(i = ROWS-1; i >= 0; i--)
      {
            for(j = 0; j < COLS; j++)
            {
                  tmp[j][i] = img[k][j];
            }
            k++;
      }
}
```

S.31 Answer:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])
{
      char *tot_str;
      int i, tot_chars;

      if(argc == 1) /* Check if the command line contains only
the name of the program. */
      {
            printf("Missing arguments ...\n");
            exit(EXIT_FAILURE);
      }
      tot_chars = 0; /* It counts the characters of all
arguments. */
      for(i = 1; i < argc; i++) /* Remember that argv[1] points
to the first argument, argv[2] to the second one, and so on.
argv[0] points to the name of the program. */
            tot_chars += strlen(argv[i]);

      tot_str = (char*) malloc(tot_chars+1); /* Allocate an
extra place for the null character. */
      if(tot_str == NULL)
      {
            printf("Error: Not available memory\n");
            exit(EXIT_FAILURE);
```

```
	}
	*tot_str = '\0'; /* Store the null character. */
	for(i = 1; i < argc; i++)
		strcat(tot_str, argv[i]);

	printf("The merged string is: %s\n", tot_str);
	free(tot_str);
	return 0;
}
```

S.32 Answer:

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

#define SIZE 100

double *find_diff(double a1[], double a2[], int size, int
*items); /* items indicates how many elements are stored in the
memory. A pointer is passed, so that the function may change its
value. */

int main(void)
{
	int i, elems;
	double *p3, j, k, p1[SIZE], p2[SIZE];

	for(i = 0; i < SIZE; i++)
	{
		printf("Enter numbers: ");
		scanf("%lf%lf", &j, &k);
		if((j == -1) || (k == -1))
			break;
		p1[i] = j;
		p2[i] = k;
	}
	elems = 0;
	p3 = find_diff(p1, p2, i, &elems);
	if(elems == 0)
		printf("\n***** No different elements *****\n");
	else
	{
		for(i = 0; i < elems; i++)
			printf("%f\n", p3[i]);
	}
	free(p3);
	return 0;
}

double *find_diff(double a1[], double a2[], int size,int *items)
```

```
{
	int i, j, found;
	double *mem;

	mem = (double*) malloc(size * sizeof(double));
	if(mem == NULL)
	{
		printf("Error: Not available memory\n");
		exit(EXIT_FAILURE);
	}
	for(i = 0; i < size; i++)
	{
		found = 0; /* This variable indicates whether an element of the first array exists in the second, or not. The value 0 means that it does not exist. */
		for(j = 0; j < size; j++)
		{
			if(a2[j] == a1[i])
			{
				found = 1;
				break; /* Since this element exists, we stop searching. */
			}
		}
		/* If it does not exist, it is stored in the memory. */
		if(found == 0)
		{
			mem[*items] = a1[i];
			(*items)++;
		}
	}
	return mem;
}
```

S.33 Answer:

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

int read_text(char str[], int size, int flag);

int main(void)
{
	FILE *fp;
	char flag, str[20], prod[20];
	double prc;
```

```
	fp = fopen("test.txt", "w+"); /* Open file for reading and
writing. */
	if(fp == NULL)
	{
		printf("Error: fopen() failed\n");
		exit(EXIT_FAILURE);
	}
	while(1)
	{
		printf("Enter price: ");
		scanf("%lf", &prc);
		if(prc == -1)
			break;
		getchar();
		printf("Enter product code: ");
		read_text(str, sizeof(str), 1);
		fprintf(fp, "%s %f\n", str, prc);
	}
	getchar();
	printf("Enter product code to search for: ");
	read_text(prod, sizeof(prod), 1);

	flag = 0;
	fseek(fp, 0, SEEK_SET);
	while(1)
	{
		if(fscanf(fp, "%s%lf", str, &prc) != 2)
			break;
		if(strcmp(str, prod) == 0)
		{
			flag = 1;
			break; /* Since the product is found exit from
the loop. */
		}
	}
	if(flag == 0)
		printf("The %s product is not listed\n", prod);
	else
		printf("The price for product %s is %f\n", prod,
prc);
	fclose(fp);
	return 0;
}
```

S.34 Answer:

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

```
#define LEN 100

typedef struct
{
	char name[LEN];
	char category[LEN];
	char singer[LEN];
	int records;
} band;

int read_text(char str[], int size, int flag);

int main(void)
{
	FILE *fp;
	band *band_arr;
	char found, name[LEN], singer[LEN];
	int i, band_num;

	fp = fopen("test.bin", "r+b");
	if(fp == NULL)
	{
		printf("Error: fopen() failed\n");
		exit(EXIT_FAILURE);
	}
	if(fread(&band_num, sizeof(int), 1, fp) != 1)
	{
		fclose(fp);
		printf("Error: fread() failed\n");
		exit(EXIT_FAILURE);
	}
	band_arr = (band*) malloc(sizeof(band) * band_num);
	if(band_arr == NULL)
	{
		fclose(fp);
		printf("Error: Not available memory\n");
		exit(EXIT_FAILURE);
	}
	if(fread(band_arr, sizeof(band), band_num, fp) == band_num)
	{
		printf("Enter band name: ");
		read_text(name, sizeof(name), 1);

		printf("Enter new singer: ");
		read_text(singer, sizeof(singer), 1);

		found = 0;
		for(i = 0; i < band_num; i++)
			if(strcmp(band_arr[i].name, name) == 0)
			{
```

```
                        fseek(fp, i*sizeof(band), SEEK_SET); /*
If the band is found, move the file pointer to the beginning of
the structure. */
                        strcpy(band_arr[i].singer, singer); /*
Change the singer and write the structure in the current
position. */
                        fwrite(&band_arr[i], sizeof(band), 1,
fp);
                        printf("\n%Singer of band %s is changed
to %s\n", name, singer);
                        found = 1;
                        break;
                  }
      }
      else
            printf("Error: fread() failed to read bands\n");

      if(found == 0)
            printf("\n%s band isn't found\n\n", name);

      free(band_arr);
      fclose(fp);
      return 0;
}
```

S.35 **Answer:** Here is a weird program with no `main()` included. The program works, though. The preprocessor replaces `type` with **`int`**, `name` with `main`, `text` with `"No main()"` and `num` with `0`. Therefore, the preprocessor expands `no_main()` to:

```
int main(void) {printf("No main()"); return 0;}
```

and the program displays: `No main()`

Notice that if we were using **`void`** instead of **`int`** the compilation would fail, because a **`void`** function cannot return a value.

S.36 **Answer:**

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

typedef unsigned char BYTE;

void Build_Pkt(int IP_src[], int IP_dst[], int port);
void Save_Pkt(BYTE pkt[], int len);

int main(void)
```

```
{
   int IP_src[4], IP_dst[4], TCP_dst_port;

   do
   {
      printf("Enter dst port [1-65535]: ");
      scanf("%d", &TCP_dst_port);
   } while(TCP_dst_port < 1 || TCP_dst_port > 65535);

   printf("Enter dst IP (x.x.x.x): ");
   scanf("%d.%d.%d.%d",   &IP_dst[0],   &IP_dst[1],   &IP_dst[2],
&IP_dst[3]);

   if(TCP_dst_port == 80)
   {
      if(IP_dst[0] == 130 && IP_dst[1] == 140)
      {
            printf("It  isn't  allowed  to  connect  to  network
130.140.x.x\n");
            return 0;
      }
      else if(IP_dst[0] == 160 && IP_dst[1] == 170)
      {
            printf("It  isn't  allowed  to  connect  to  network
160.170.x.x\n");
            return 0;
      }
   }
   printf("Enter src IP (x.x.x.x): ");
   scanf("%d.%d.%d.%d",   &IP_src[0],   &IP_src[1],   &IP_src[2],
&IP_src[3]);

   Build_Pkt(IP_src, IP_dst, TCP_dst_port);
   return 0;
}

void Build_Pkt(int IP_src[], int IP_dst[], int port)
{
   BYTE pkt[40] = {0}; /* Initialize all fields to 0. */
   int i, j;

   pkt[0] = 0x45; /* Version, IHL. */
   pkt[8] = 255; /* Time to Live. */
   pkt[9] = 6; /* Protocol = TCP. */
   for(i = 12, j = 0; i < 16; i++, j++)
      pkt[i] = IP_src[j]; /* IP Source. */
   for(i = 16, j = 0; i < 20; i++, j++)
      pkt[i] = IP_dst[j]; /* IP Destination. */
   pkt[20] = 1500 >> 8; /* TCP Source Port. */
   pkt[21] = 1500 & 0xFF;
   pkt[22] = port >> 8; /* TCP Dest Port. */
```

```
   pkt[23] = port & 0xFF;
   pkt[33] = 2; /* SYN bit. */
   pkt[34] = 0xFF; /* The maximum value for the Window field is
all 16 bits set to 1. */
   pkt[35] = 0xFF;
   /* The values of the CheckSum and Urgent Pointer are set in
positions 36-40, therefore the total length of the IP packet is
40 bytes. */
   pkt[2] = 0; /* IP Total Length. */
   pkt[3] = 40;

   Save_Pkt(pkt, 40);
}

void Save_Pkt(BYTE pkt[], int len)
{
      FILE *fp;
      char name[100];
      int i;

      printf("Enter file name: ");
      scanf("%s", name);

      fp = fopen(name, "w");
      if(fp == NULL)
      {
            printf("Error: fopen() failed\n");
            exit(EXIT_FAILURE);
      }
      for(i = 0; i < len; i++)
      {
            if(i%16 == 0)
                  putc('\n', fp);
            fprintf(fp, "%02X ", pkt[i]);
      }
      fclose(fp);
}
```

Comments: Do you have any idea about what this program really does? This program is an oversimplified version of a popular application, almost certainly installed in your computer, the *firewall*. Like this program, a firewall may prevent communication to specific IP addresses and specific applications (e.g., web servers listen to TCP port 80). In fact, the main part of a firewall is nothing more than a sequence of **`if-else`** statements.

In a real networking application, this IP packet is encapsulated in a MAC frame (see C.11.38), the hardware of the network card encodes the bits of the frame in digital signals (e.g., two different voltage levels to represent `0` and `1`) and

forwards the frame to the connected router. The router checks the IP destination address, finds the best path to that destination and forwards the frame to the next router. Yes, we understand, it is all Greek to you, we just tried to introduce you in the fascinating world of Computer Networking.

S.37 **Answer:** When `test()` is called, we have:

a. `a = *(tmp+1) = *(txt+1) = txt[1] = 'b'`.
b. `str = txt+3`, so, `str[0]` is equal to `txt[3]`.
c. `ptr = &txt[1] = txt+1`. Since `ptr` points to `txt[1]`, `ptr[0]` is equal to `txt[1]`, `ptr[1]` is equal to `txt[2]`, and `ptr[2]` is equal to `txt[3]`.

`strcpy()` copies the string `"1234"` into the memory that `str` points to. Since `str` points to `txt[3]`, the content of `txt` becomes `"abc1234"`.

The statement `str[0] = a;` is equivalent to `txt[3] = a`. Since the value of `a` is equal to `'b'`, `txt[3]` becomes `'b'` and the content of `txt` changes to `"abcb234"`.

As said, the values of `str[0]` and `ptr[2]` are equal to `txt[3]`. Therefore, `*str` is equal to `'b'` and the statement `ptr[2] = *str+5;` is equivalent to `txt[3] = 'b'+5;` meaning that `txt[3]` becomes equal to the character located five places after `'b'`, that is, `'g'`.

Therefore, the program displays: `abcg234`.

S.38 **Answer:** The `arr` array is declared as an array of pointers to strings. In particular, `arr[0]` points to `"TEXT"`, `arr[1]` points to `"SHOW"`, and so on. The statement `ptr1 = arr;` makes `ptr1` to point to the address of `arr[0]`.

The expression `++ptr1` makes `ptr1` to point to `arr[1]`. Since `*++ptr1` is equivalent to `arr[1]`, the program displays `SHOW`.

Similarly, the expression `++ptr1+2` makes first `ptr1` to point to `arr[2]`. Since `*++ptr1` is equivalent to `arr[2]`, `printf()` is equivalent to `printf("%s", ptr[2]+2);` and the program skips the first two characters of `"OPTIM"` and displays `TIM`.

Like before, the expression `++ptr1+1` makes first `ptr1` to point to `arr[3]`. Therefore, `**++ptr1` is equivalent to `*arr[3]`. What is the value of

*arr[3]? Since arr[3] points to the first character of "DAY", *arr[3] is equal to 'D'. Therefore, the value of *arr[3]+1 is 'E' and the program displays E.

As a result, the program displays: SHOW TIME

S.39 Answer:

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

#define ROWS 30
#define COLS 20

int main(void)
{
	int	i,	j,	sel,	row,	col,	rsvd_seats,	cost,
seats[ROWS][COLS] = {0}; /* We use the seats array to manage the
cinema's seats. If an element is 0, it implies that the seat is
free. */
	rsvd_seats = cost = 0;
	while(1)
	{
		printf("\nMenu selections\n");
		printf("---------------\n");

		printf("1. Buy Ticket\n");
		printf("2. Ticket Refund\n");
		printf("3. Show Information\n");
		printf("4. Exit\n");

		printf("\nEnter choice: ");
		scanf("%d", &sel);

		switch(sel)
		{
			case 1:
				if(rsvd_seats == ROWS*COLS)
				{
					printf("\nSorry,	no	free
seats\n");
					break;
				}
				printf("\nWould you like a specific seat
(No: 0)? ");
				scanf("%d", &sel);
				if(sel == 0)
				{
					do
					{
```

```
                                    row = rand() % ROWS; /* Use
rand() to select a random seat. */
                                    col = rand() % COLS;
                              } while(seats[row][col] == 1);
                        }
                        else
                        {
                              do
                              {
                                    printf("\nEnter row [1-%d]:
", ROWS);
                                    scanf("%d", &row);
                              } while(row < 1 || row > ROWS);

                              do
                              {
                                    printf("Enter seat [1-%d]:
", COLS);
                                    scanf("%d", &col);
                              } while(col < 1 || col > COLS);

                              row--;  /* Subtract 1, since
indexing starts from 0. */
                              col--;
                        }
                        if(seats[row][col] == 1)
                              printf("\nSorry, seat in row_%d
and column_%d is reserved\n", row+1, col+1);
                        else
                        {
                              seats[row][col] = 1;
                              cost += 6;
                              rsvd_seats++;
                        }
                  break;

                  case 2:
                        if(rsvd_seats == 0)
                        {
                              printf("\nAll seats are free\n");
                              break;
                        }
                        do
                        {
                              printf("\nEnter row [1-%d]: ",
ROWS);
                              scanf("%d", &row);
                        } while(row < 1 || row > ROWS);

                        do
                        {
```

```
                                    printf("Enter   seat   [1-%d]:   ",
COLS);
                                    scanf("%d", &col);
                              } while(col < 1 || col > COLS);

                              row--;
                              col--;
                              if(seats[row][col] != 1)
                                    printf("\nSeat   in   row_%d   and
column_%d is not reserved\n", row+1, col+1);
                              else
                              {
                                    seats[row][col] = 0;
                                    cost -= 5;
                                    rsvd_seats--;
                              }
                        break;

                        case 3:
                              printf("\nFree   seats:   %d,   Income:
%d\n\n", ROWS*COLS - rsvd_seats, cost);
                              for(i = 0; i < ROWS; i++)
                              {
                                    for(j = 0; j < COLS; j++)
                                    {
                                          if(seats[i][j] == 1)
                                                printf("%2s", "X");
                                          else
                                                printf("%2s", "#");
                                    }
                                    printf("\n");
                              }
                        break;

                        case 4:
                        return 0;

                        default:
                              printf("\nWrong choice\n");
                        break;
                  }
            }
            return 0;
}
```