Universität Innsbruck - Institut für Informatik
Prof. Clemens Ballarin, Robert Binna, Friedrich Neurauter und Francois Scharffe

1. April 2008

Proseminar Algorithmen und Datenstrukturen

Exercise sheet 2

Exercise 1 (Programming in C)

Read Chapter 3 from "Brian W. Kernighan, Dennis M. Ritchie: Programmieren in C."

Exercise 2 (Loops - While and For)

a) Write a C program that uses a for-loop to calculate the following sum:

$$\sum_{i=1}^{n} i$$

The parameter n should be entered by the user at run-time. The result of the calculation should be printed to the terminal.

```
#include <stdio.h>

int main()
{
  int i = 0, n = 0;
  int res = 0;

  printf("\n\n1 + 2 + ... + n = ?\n\n");
  printf("Please enter n: ");
  scanf("%d", &n);

  for(i=1;i<=n;++i)
    res += i;
```

```
  printf("\nThe result of 1 + 2 + ... + %d is %d \n", n, res);
  return 0;
}
```

b) Extend your program in the following way:

After calculating $\sum_{i=1}^{n} i$ for a specific n the user should be asked whether he would like to do another round, calculating the sum for a new n, or whether he would like to quit the program. A do-while-loop should be useful for this task.

```
#include <stdio.h>

int main()
{
  int i = 0, n = 0;
  int res = 0;
  char c = '\0';

  do
  {
    res = 0;

    printf("\n\n1 + 2 + ... + n = ?\n\n");
    printf("Please enter n: ");
    scanf("%d", &n);
    getchar(); //throw away the '\n' that is still in the input buffer

    for(i=1;i<=n;++i)
      res += i;
    printf("\nThe result of 1 + 2 + ... + %d is %d \n\n", n, res);

    printf("Do you want to continue? [y/n] ");
    scanf("%c", &c);
    getchar(); //throw away the '\n' that is still in the input buffer
  } while ((c == 'y') || (c == 'Y'));

  return 0;
}
```

This program works, yet it is not the best of solutions. Therefore examine what happens if the user enters something unexpected for the parameter n rather than an integer number. This awkward behaviour is caused by the way `scanf` reads input from the standard input stream. Check out the following link for more information on this topic:
http://www.galileocomputing.de/openbook/c_von_a_bis_z/c_005_001.htm

Hints:

- Useful standard library functions: `printf`, `scanf`, `getchar`
- To display the help text for library functions type `man function-name`, eg. `man scanf`, into your Linux terminal. In certain cases (with `printf`, for example) you will have to type `man 3 function-name` to get the desired information.

Exercise 3 (Leibniz' Formula)

Consider Leibniz' formula

$$4 \cdot \sum_{i=0}^{\infty} \frac{(-1)^i}{2 \cdot i + 1} = \pi$$

for computing π. By changing ∞ to some natural number n this sum becomes finite, hence computable, and allows us to approximate π. Now write a C program that approximates π such that the difference between the approximation and the real value of π is smaller than $1e-4$ $(= 0.0001)$.

Hints:

- Use the datatype `double`.
- Use `printf(``... %.13f ...'', ...)` to print `double` values.
- The header file `math.h` defines the constant `M_PI` for π.
- if your program does not compute the desired value, and you can't find the error, then you will most likely have a datatype related problem. Make sure your division operation involves `double` values as opposed to `int` values.

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

int main()
{
  double res = 0.0, res_old = 0.0;
  long sign = 1;
  long i = 0;

  do
  {
    res_old = res;
    res += sign * 1.0/(2.0 * i + 1.0);
    sign *= -1;
    ++i;
```

```
  } while (fabs(res - res_old) >= 1e-4/4.0);
  /* if the absolute difference between two consecutive
  iterations is smaller than 1e-4 (=0.0001) then the
  difference between the approximation and the real value
  of pi is certainly also smaller than 1e-4. Note: the division
  by 4 is due to the fact that pi = 4 * res
  */
  printf("After %ld iterations pi is approximately %.13f\n\n", i, 4 * res);
  printf("The error we make is: %.13f\n\n", fabs(M_PI - 4 * res));
  return 0;
}
```

Exercise 4 (Arrays)

a) Declare the following array inside of your `main()` function:

```
int arr[10]
```

Now write some code (using a for-loop) that prints every element of the array. Then move the declaration of the array outside of your `main()` function and print it again. What is the difference? Can you explain what happens?

Hint:

- The initialization of *global* and *local* variables is different.

```
#include <stdio.h>

//int arr[10];

int main()
{
  int arr[10];
  int i = 0;

  for(i=0;i<10;++i)
    printf("arr[%d] = %d\n", i, arr[i]);
  return 0;
}
```

Global variables are initialized at the start of the program (the actual initialization value depends on the data type, eg. 0 for `int`) whereas local variables are not.

b) Write a C program that computes the mean value of the following array:

`int arr[10]`

At the beginning of the program the user should be asked for 10 integer numbers, which should then be stored in the array. Afterwards their mean value should be computed and printed.

```
#include <stdio.h>

int main()
{
  int arr[10];
  int i = 0, sum = 0;

  for(i=0;i<10;++i)
  {
    printf("\nEnter an integer number for arr[%d] : ", i);
    scanf("%d", &arr[i]);
    sum += arr[i];
  }
  printf("\nThe mean value is %f\n", sum/10.0);
  return 0;
}
```

Exercise 5 (Data Types)

Consider the following C program:

```
#include <stdio.h>

int main()
{
  char c = 'a';
  float f = 0.1f;
  double d = 0.1;
  short s = 1;
  int i = 1;
  long l = 1;

  printf("sizeof char is %d byte(s).\n", sizeof(c));
  // this is equivalent: printf("sizeof char is %d byte(s).\n", sizeof(char));
  printf("sizeof float is %d byte(s).\n", sizeof(f));
  printf("sizeof double is %d byte(s).\n", sizeof(d));
  printf("sizeof short is %d byte(s).\n", sizeof(s));
  printf("sizeof int is %d byte(s).\n", sizeof(i));
```

```
    printf("sizeof long is %d byte(s).\n", sizeof(l));

    return 0;
}
```

What does it do? Interpret the output of the program! Does the output depend on the computer the program runs on?

Hint:

- Chapter 2 from "Brian W. Kernighan, Dennis M. Ritchie: Programmieren in C."