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

8. April 2008

Proseminar Algorithmen und Datenstrukturen

Exercise sheet 3

Exercise 1 (Programming in C)

Read Chapter 4 from "Brian W. Kernighan, Dennis M. Ritchie: Programmieren in C." (You may skip Sections 4.6 and 4.7.)

Exercise 2 (Operator Precedence)

Consider the following C program.

```
#include <stdio.h>

int main() {
  // a)
  if (2>1 || 4>0 && -1>0) printf("true\n"); else printf("false\n");
  printf (('y'>'x' && 'a'>'A' || 'a'!='b') ? "true\n" : "false\n");

  // b)
  int a[2] = {0,0};
  int i=0;
  a[i]=i++;
  printf("a=[%i, %i]\n", a[0], a[1]);

  // c)
  int x=0;
  int y = ++x + x--;
  printf("y=%i\n", y);
```

```
  // d)
  int z = 8 << 4 + 2 >> 1;
  printf("b=%i\n", z);

  // e)
  int k = 0;
  printf("%i, %i\n", ++k, k+k);

  return 0;
}
```

What is the output produced by this program? In which cases does it depend on the compiler the program runs on?

Hint:

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

a) `true` in both cases

b) e.g. `a=[1, 0]`, `a=[0, 0]` or `a=[0, 1]`, depending on the compiler

c) `y=2` or `y=0`, depending on the compiler

d) `b=256`

e) `1, 0` or `1, 2`, depending on the compiler

Exercise 3 (Variable Scope)

Consider the following snippets of C code containing multiple variables with the same name. For each occurrence of the respective name, explain which variable is accessed and account for the resulting output.

a)
```
char a = 'a';
printf("Press a key: ");
if (getchar() == 'y')
  {char a = 'y'; printf("a = '%c'\n", a ); }
else
  {char a = 'n'; printf("a = '%c'\n", a ); }
printf("a = '%c'\n",a);
```

b)
```
int b = 10;
   {
      int b = 0;
      for(int b = 0; b < 3; b++ )
```

```
            printf("in loop: b = %d\n", b );
          printf("after loop: b = %d\n",b);
        }
        printf("after block: b = %d\n", b);
```

c)
```
int c = 0;
  do {
    int c = 0;
    ++c;
    printf("c = %d\n", c );
  } while( ++c < 3 );
```

Exercise 4 (The Longest Upsequence)

Consider a sequence of natural numbers $(n_1, n_2, \ldots, n_k)$. If one deletes i (not necessarily adjacent) numbers from the list, one has a subsequence of length $k - i$. This subsequence is called an *upsequence* if its values are in non-decreasing order. For example, the sequence $(1, 3, 4, 6, 2, 4)$ has a subsequence $(1, 3, 2)$, which is not an upsequence, and another subsequence $(1, 3, 6)$, which is an upsequence. In the lecture you saw the pseudo code of an algorithm computing the length of the longest upsequence of a sequence of natural numbers. Implement this algorithm in C.

```
#include <stdio.h>

int main() {
  int n = 8; /* array length */
  int X[] = {1, 3, 4, 6, 2, 4, 4, 0};
  int M[n]; for (int i=0; i<n; i++) M[i]=0;
  int i;

  int k = 0;
  M[0] = X[0];
  for(i = 1; i < n; i++){
    if (X[i] >= M[k]) {
      k++;
      M[k] = X[i];
      }
    else if (X[i] < M[0]) {
      M[0] = X[i];
      }
    else {
      /* find j such that M[j-1] <=X[i]<M[j] */
      /* Possibility 1: naive search
```

```
      int j = i;
      while (M[j-1] > X[i])
        j--; */
      /* Possibility 2: binary search */
      int b, j = 1;
      for (b = (k+1)/2; M[j-1] > X[i] || X[i] >= M[j]; b /= 2)
        if (M[j-1] > X[i])
          j -= b;
        else
          j += b;
      M[j] = X[i];
    }
  }
  printf("Longest upsequence has length %d\n", k + 1);
  return 0;
}
```

Exercise 5 (Binomial Coefficient)

a) Use a *for* loop to implement a function `unsigned long fac(unsigned long n)` computing the factorial $n! = \prod_{i=1}^{n} i$ for a natural number $n \geq 0$, where $0! = 1$.

b) Use your factorial function to write a function `unsigned long bc(unsigned long n, unsigned long k)` with two arguments that computes the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

for two natural numbers $n, k \geq 0$.

c) For a second version of `bc`, employ the formula

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

to obtain a recursive implementation, where $\binom{n}{k} = 0$ if $k > n$ and $\binom{n}{0} = 1$.

d) Write a `main()` function that prompts the user to input two natural numbers n and k and outputs `bc(n, k)`. Which version of your `bc` function can compute the number of guesses possible in the National Austrian Lottery ("6 aus 45")?

```
#include <stdio.h>

unsigned long fac(unsigned long n);
```

```
unsigned long bc(unsigned long n, unsigned long k);
unsigned long bc2(unsigned long n, unsigned long k);

int main(){
  int n, k;

  printf("Enter n: ");
  scanf("%d", &n);

  printf("Enter k: ");
  scanf("%d", &k);

  printf("recursively computed: bc(n, k) = %d \n", bc2(n, k));
  printf("iteratively computed: bc(n, k) = %d \n", bc(n, k));

  return 0;
}

unsigned long fac(unsigned long n){
  unsigned long i, r = 1;
  for (i = 2; i <= n; i++)
    r = r * i;
  return r;
}

unsigned long bc(unsigned long n, unsigned long k){
  return fac(n) / (fac(k) * fac(n-k));
}

unsigned long bc2(unsigned long n, unsigned long k){
  if (k > n)
    return 0;
  else if ((n == k) || (k == 0))
    return 1;
  else
    return bc2(n-1, k) + bc2(n-1, k-1);
}
```

Exercise 6 (Binary Representation)

Functions are an important programming construct to structure code and split larger problems into smaller ones. They also allow to build upon code already present, as opposed to start from scratch over and over again.

The aim of the following exercise is to provide functions transforming between a natural number and its binary representation. These functions can then be integrated in other programs.

a) In the program below, the function `to_binary(int n, int bits[], int len)` should compute the binary representation of a natural number `n` using `len` bits. (Arrays can not be directly returned by functions in C. Instead, the result can be stored in the `bits` array which is passed as an argument.) Moreover, 0 should be returned if the computation was successful and 1 if `n` was too big such that an overflow occurred, i.e. $n \geq 2^{len}$. Complete the function code respectively!

```
#include <stdio.h>
#define MAXBITS 16 /* maximal number of bits used */

int bitarray[MAXBITS]; /* pointer to array of bits */

/* function head */
int to_binary(int n, int bits[], int len);

int main(){
  int number = 17;
  int ok = to_binary(number, bitarray, MAXBITS);
  /* output bitarray here if ok == 0 */
  return ok;
}

/* function implementation */
int to_binary(int n, int bits[], int len){
  /* to be completed: fill bits array */
}

int to_binary(int n, int bits[], int len){
  int k = 0;
  while (k < len)
    {
    bits[len - k - 1] = n % 2;
    n = n / 2;
    k++;
    }
  return (n > 0);
}
```

b) Add a function `void print_array(int bits[], int len)` which can be used to output the computed binary representation.

```
void print_array(int bits[], int len){
  printf("[");
  for (int i=0; i < len; i++)
   printf("%d", bits[i]);
  printf("]\n");
}
```

c) Write a function `int to_decimal(int bits[], int len)` which returns the natural number encoded in the array `bits` of length `len`.

```
int to_decimal(int bits[], int len){
  int k = 1; /* k contains powers of 2 */
  int d = 0; /* result */
  int i;
  for (i = len-1; i >= 0; i--) {
    d = d + k * bits[i];
    k = k * 2;
    }
  return d;
}
```

Hints:

- The algorithm presented in the lecture might be helpful.
- You can also use `unsigned` types in this exercise.