Programming in C, course exam October 18, 2012, evaluation principles

Write on every paper the name of the exam, date, your name, your personal identification number or student number, and your signature. Enumerate the pages.

1. **a)** Write a function `void printRightBit(uint8_t byte)` which prints the rightmost bit of byte.

   Max 2 points: Nearly all got 2 points, some zero points.

   b) Write a function `void printLeftBit(uint8_t byte)` which prints the leftmost bit of byte.

   Max 3 points:

   A common mistake was such that a mask was used to transform the byte bits so that the leftmost bit was original and all the other bits were zero. After this, the byte was printed. So a large number was printed. Only two points was given.

   c) Consider the following structure:

```
struct person{
    char * name;
    int age;
};
```

   The following piece of code reads n names and ages from the keyboard, creates a structure for every name and places the names and ages into the corresponding structures.

```
struct person * ptr;
int i;
char  name[50];
int age;

ptr = malloc(n*sizeof(struct person)); /* n defined earlier */

for (i=0; i < n; i++)
{
    fgets(name,50,stdin);
    scanf("%d", &age);
    (ptr+i)->name = name;
    (ptr+i)->age = age;
}
```

   Is the code working properly?

Max 3 points: If it was noticed that there is no space reserved for name, 3 points. All the other answers zero.

2. **a)** Let p be a pointer i a variable of integer type. What does $p + i$ mean?

   Max 1 point: The answer shows, that the result is a pointer equivalent to `&p[i]`, (1 point) Explained answers without the notion were accepted too.

   b) Let p and q be pointers. How do you intepret $p - q$? When is the expression erroneous?

   Max 2 points:

   - The answer shows "p-q"represents a distance, size of range or a count of objects, (1 point)
   - The result is not a meaningful pointer to memory.
   - The range is exclusive, i.e. not including q.
   - Proof: `q = p; (p - q) == 0;  QED`.
   - The inclusive range would have been p-q+1
   - You didn't lose points if inclusiveness was the only mistake.
   - At least one valid reason for when they are invalid is given, (1 point)
   - When the pointers do not reference the same consistent block, e.g. an array.
   - They point somewhere else, are uninitialised or null
   - When the pointers are of different types
   - When q points to a latter area in memory than p
   - Remember, that the pointers are an unsigned integer type.

   c) Let p be a pointer and a an array. Where does p refer to after $p = a$ and after $p = \&a[0]$?

   Max 2 points: The answer shows that the first assignment makes p point to the beginning of the array, (1 point) The answer shows that the second assignment makes p point to the beginning of the array, (1 point) It's ok to have shown them separately, with both correct you get both points. It's even better to have pointed out that the assignments are equivalent (have the same result) and just shown the other.

   d) Let a be an array. What does $*(a + i)$ mean?

   Max 1 point: The answer shows, that `*(ai)`+ is equivalent to `a[i]`, (1 point) The answer is a dereferenced value, claiming it to be a pointer is incorrect. If it were an array of pointers, the pointers it contains are still it's values.

   e) Suppose that `sizeof(short) == 2` and `sizeof(int) == 4`. Consider an array

   `int arr[5] = { 0, 0, 0, 0, 0 };` Explain where the following statement stores values (both the location and the amount of bytes written).

   ```
   ((short*)arr)[7] = 128;
   ```

Max 2 points: The placement of bytes where the assignment is made was correct, (1 point) Both indexed from 0 and 1 were accepted, if it was clear which one was used e.g. from a picture.

The number of bytes assigned to was correct (2 bytes), (1 point) The original array was an int array, but the pointer was cast into a short, therefore, the assignment is made as short. An assignment doesn't necessarily mean that the new value is different from the old value. Even though the value 128 fits in one byte (if unsigned), claiming that the assignment is done to only one byte is incorrect.

3. Let's assume that the following definitions and functions are available.

```
typedef struct term{
   int deg;
   int coeff;
  struct term * prev;
  struct term * next;
} termNode;

termNode* makeTerm(int deg, int coeff);

termNode * addPolynomialsMod(termNode* P, termNode* Q, int n);
```

Thus it is assumed that a polynomial is represented as a doubly linked list in such a way that the terms are composed of nodes of type `termNode`. The terms are in decreasing order with respect to a degree. Thus the highest degree term is first. Write a function

```
termNode* mulPolynomialsMod(termNode* P, termNode* Q, int n)
```

that calculates the product of two polynomials modulo n. The zero polynomial is represented by the term `makeTerm(0,0)`. Parameter values `P == NULL` and `Q == NULL` are incorrect values which cause the function to return without calculating anything.

Max 9 points: 4 points from general idea of multiplying all the terms of p with all the terms of q with following conditions:

- -1 point from eternal loops (in a typical solution remembering to have `p = p-> next` and `q = q -> next`)
- -1 point from forgetting to reset q to the first term after looping it to the last term resulting that you multiply q only with the first term of p (other working solutions like going q backwards every other round are naturally also accepted)
- -1 point from clear seqmentation fault or not going through all the terms (for example forgetting the last one)
- -1 from other mistakes.