# Week 13. Final exam review

## 1. True or false?

A. I claim that linked lists have the following advantages over the arrays:

- They allow insertion in the middle in a constant time
- They allow access to the element at position *k* in a constant time
- They use less memory
- The search is faster because we are following the pointers

B. I claim that for the following variables:

```
char *a;  int *b;
```

- a and b store values of different types
- sizeof(a) ≠ sizeof(b)
- sizeof(*a) ≠ sizeof(*b)

C. I claim that for the following declarations:

```
char a [] = "abcde";  char *b="abcde";
```

- sizeof(a)=sizeof(b)
- a and b both are variables that store an address
- we can do both a=b and b=a
- the amount of memory used is the same for both declarations
- we can do both a[1] = ‘d’; and b[1]=’d’;
- we can pass both a and b as parameters to a function *func (char *c)*

## 2. What is legal?

```
int x, y;
int *px, *py, *p;
float *pf;

px = &x;
py = &y;
p = px + py;
p = px * py;
p = px + 10.0;
pf = px;
```

## 3. Linked lists

Given new data type *node*:

```
typedef struct node{
   int data;
   struct node * next;
}node;
```

- How do we declare a list of nodes?

- How do we insert a new node *new_node* after the k-th element (suppose k=2) of the list?
- How do we make a circular list of nodes?

- How can we reverse the order of elements in the list in one iteration?

- How can we add new element on top of the list in a void function?

```
node *get234List () {
   node * head;
   node * a = (node *)calloc (1, sizeof(node));
   a->data = 2;
   node * b = (node *)calloc (1, sizeof(node));
   b->data = 3;
   node *c = (node *)calloc (1, sizeof(node));
   c->data = 4;
   a->next = b;
   b->next = c;

   head = a;
   return head;
}

void addOnTop (node * head,  int value) {
   node * d = (node *)malloc (sizeof(node))
   d->data = value;
   if (head == NULL)
         head = d;
   else {
         d->next = head->next;
         head = d;
   }
}




int main() {
   node * head = get234List();
   addOnTop (head, 1);
   //what is the length of the list now?
}
```

## 4. Memory segments

Draw memory diagram and say where each variable is stored and to which memory segment it points to (in case it is a pointer):

### A. Fun

```
int fun (char a, char b) {
      a++;
      b++;
      return b;
}
char a=’a’; //value 97
char b=’b’;
int (*func) (char, char);

int main () {
      func = fun;
      char c = (char) func (a, b);
      printf (“%c %c %c\n”, a, b, c);
      //what is printed by the way?
}
```

### B. Even more fun

```
int * more_fun (char *a) {
      a = malloc (5);
      *a = ‘a’;
      *(a+1) = ‘b’;
      *(a+2) = 0;
      int result[] = {1,2};
      return result;
}

int main () {
      char *str;
      int *ip = more_fun (str);
      printf (“%d %s\n”, *ip, str);
}
```

## 5. Pass by value (even pointer variables)

```
   void init_array1 (char * a, int size) {
      a = (char *) malloc (size);
      strncpy(a, "new value", size-1);
      a[size-1] = '\0';
   }

   int main() {
      char * y = "abba";
      init_array1 (y, 8);
      fprintf (stdout, "Array after init1 - %s\n", y);
   }
```

## 6. File descriptors

A. If you want a parent process to read from a pipe and a child process to write to a pipe, which file descriptors do you leave open?

Parent: fds[0] or fds[1]

Child: fds[0] or fds[1]

B. You want to implement the following shell pipe in a C program

```
sort file1 | head
```

- Which process should be the parent and which one the child?
- How would you use *dup2* to set standard output of a child process to the writing end of a pipe, and standard input of a parent process to the reading end of the pipe?

Parent file descriptors: 0, 1, 2, fds[0], fds[1]

dup2(________, ________)

Child file descriptors: 0, 1, 2, fds[0], fds[1]

dup2(________, ________)

C. Sockets

Server code:

```
int a= socket(family, type, protocol);

int b= accept(a, &clientAddr, &addrLen);
```

Client code:

```
int c= socket(family, type, protocol);

int d=connect(c, &foreignAddr, addrlen);
```

Which of the file descriptors a,b,c (or d) are used for sending data between server and client?

## 7. Handling signals

- How can we make our program to ignore an interrupt signal?

- How can we make sure that our signal handler is not interrupted in the middle by an interrupt signal?

- How can we make sure that the important section of code gets uninterrupted by any signal?