# Vložení řetězce s realokací paměti

Napište funkci

```
int insertStr (char *si, char **st, int k)
```

pro vložení řetězce `si` do řetězce `st` na pozici `k`.

Například, je-li v `st` řetězec `"abcd"`, pak po volání

```
insertStr("XY", st, 3)
```

v něm bude řetězec `"abcXYd"`.

Řetězec `st` je alokován v dynamické paměti na hromadě a je nutné mu paměť rozšířit pomocí knihovní funkce `realloc`.

Formální parametr `st` je zde ukazatel na řetězec, protože je volán odkazem. Funkce `realloc` ho totiž může změnit.

# Vložení řetězce

```
int insertStr (char *si, char **st, int k) {
                // vkládaný, cílový, pozice

    /*  DOPLŇTE  */

  return 0;  // funkce vrací jen chybový kód
}

int main (void) {
  char *st = (char *) malloc(5);
  if (st == NULL) return 1;
  strcpy(st, "abcd");

  printf("%s\n", st);       // abcd
  insertStr("XY", &st, 3);
  printf("%s\n", st);       // abcXYd

  free(st);
  return 0;
}
```

# Vložení řetězce s vynecháním úseku v řetězci

Napište funkci

```
int replaceStr (char *si, char **st, int k, int p)
```

která se chová podobně jako `insertStr`, ale kromě vložení `si` na pozici `k` se v původním řetězci vynechají znaky na pozicích `k` až $p-1$.

Musí platit $0 \leqslant k \leqslant$ `strlen(st)`, $k \leqslant p \leqslant$ `strlen(st)`.

Například, je-li v `st` řetězec `"abcdefgh"`, pak po volání

```
replaceStr ("XY", st, 2, 5)
```

v něm bude řetězec `"abXYfgh"`.

Rozdíl od funkce `insertStr` je v tom, že teď se v první fázi posunuje kratší úsek (o $p-k$ znaků), ale může se posunovat

- jak doprava (je-li `strlen(si)` $> p-k$),
- tak doleva (je-li `strlen(si)` $> p-k$),
- případně vůbec ne (při rovnosti délek).

# Náhrada řetězce

Napište funkci

```
substitute (char *s, char *t, char *u)
```

která v řetězci `u` nalezne první výskyt řetězce `t` a nahradí ho řetězcem `s`.

- Pro nalezení pozice použijte funkci `match` ze 4. cvičení.
- Použijte funkci `replacetStr` z tohoto cvičení.
- Funkce `substitute` mění jen řetězec `u`, ostatní nemění.
- Funkce vrací záporné kódy chyb, číslo 1 při úspěšném nahrazení, číslo 0 při nenalezení vzorku.

Modifikujte funkci `substitute` tak, aby nahrazovala *všechny* výskyty vzorku `t`. Jako svůj výsledek funkce vrací *počet provedených náhrad*.

# Ladění

Pro detekci a analýzu chyb v programu použijte:

- kontrolní výpisy (viz přednášku).

```
#ifdef DEBUG
#define pmsg(s,...) fprintf(stderr,__FILE__":%u: " s "\n",__LINE__,__VA_ARGS__)
#else
#define pmsg(...) {}
#endif
...
pmsg("indexing array: i=%d, ar[i]=%f", i, ar[i])
```

- nástroj *GDB*
  Program překládejte s volbou `-g` pro gcc.