Ejercitación – Serie K+

Memoria dinámica

Además de poder desarrollar nuestras funciones y aplicaciones, en muchos casos tendremos que intervenir en códigos de terceros, e implementar nuestras modificaciones.

En los siguientes ejercicios/códigos se encontrarán con algunos inconvenientes que deberán corregir, sean estos de memoria dinámica u otro tema.

Los códigos acá presentados pueden copiarse o descargarse del aula virtual (serie K+ codes)

Ejercicio 1:

El siguiente código presenta varias fallas, además de imprimir valores incorrectos y no finalizar en forma correcta. Se solicita identificar estos errores y corregirlos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
  int code, cant;
  char *desc;
} data_t;

int setData(data_t *pd, int cod, int qtt, char *str);
void prtData(const data_t *pd);
void freeData(const data_t *pd);

int main() {
  data_t da, db, dc;

  setData(&da, 2456, 365, "Colector de base abstracta");
  setData(&db, 0056, 126, "Bromuro de sustrato metalico A o B (56:A / 57:B)");

  if (!da.desc || !db.desc) {
    fprintf(stdout, "Error grave de memoria\n");
    fprintf(stdout, "La aplicación finaliza\n");
    return 1;
  }

  dc = db;
  dc.code++;
  dc.cant++;

  fprintf(stdout, "----------------------------\n");

  prtData(&da);
  prtData(&db);
  prtData(&dc);
```

```
  fprintf(stdout, "----------------------------\n");

  freeData(&da);
  freeData(&db);
  freeData(&dc);
  return 0;
}

int setData(data_t *pd, int cod, int qtt, char *str) {
  int aux = strlen(str) + 1;
  pd->code = cod;
  pd->cant = qtt;

  if ((pd->desc = (char *)malloc(aux * sizeof(char)))) {
    strcpy(pd->desc, str);
    return 1;
  }
  return 0;
}

void prtData(const data_t *pd) {
  fprintf(stdout, "code: %5d | cant: %4d ", pd->code, pd->cant);
  fprintf(stdout, "| %s\n", pd->desc);
}

void freeData(const data_t *pd) { free(pd->desc); }
```

Ejercicio 2:

Por favor, revisar el siguiente código que parecería tener algunos inconvenientes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
  int len, size;
  char *str;
} data_t;

int setData(data_t *pd, char *str);
int addData(data_t *pd, char *str);
void freeData(const data_t *pd);

int main() {
  data_t db = {0, 0, NULL};
  data_t da;
  int aux = 0;

  aux = setData(&da, "Buenos días,\n");
  aux += addData(&da, "Este es un codigo que presenta algunos problemas.");
  aux += addData(&da, "Te animas a encontrarlos y corregirlos.\n");
  aux += addData(&da, "Quizas debas replantear parte de la solución.\n");
  if (aux != 4) {
    fprintf(stdout, "Error de memoria. Chau!!\n");
    return 1;
  }

  fprintf(stdout, "----------------------------\n");
```

```
  fprintf(stdout, ":--> %s\n", da.str);
  fprintf(stdout, "----------------------------\n");

  aux = 0;
  aux = addData( &db, "Bien, en este caso estamos iniciando con un addData en form
directa\n");
  aux += addData(&db, "Podemos hacer esto?.\n");
  aux += addData(&db, "Parecería que si, pues esto no se cuelga, por lo que debería
estar bien.\n");
  aux += addData(&db, "Si. parece que esta perfecto!!!.\n");
  if (aux != 4) {
    fprintf(stdout, "Error de memoria. Chau!!\n");
          free(da.str);
    return 1;
  }
  fprintf(stdout, ":--> %s\n", db.str);
  fprintf(stdout, "----------------------------\n");
  aux = setData(&da, "Estamos llegando al final de codigo,\n");
  aux += addData(&da, "Quien dijo que este codigo presenta algunos problemas.");
  aux += addData(&da, "Por lo que veo anda perfecto.\n");
  aux += addData(&da, "o quizás no? tendra eso que le llaman \"vicios ocultos\".\n");
  if (aux != 4) {
    fprintf(stdout, "Error de memoria. Chau!!\n");
    return 1;
  }
  fprintf(stdout, ":--> %s\n", da.str);
  fprintf(stdout, "----------------------------\n");

  freeData(&da);
  freeData(&db);
  return 0;
}

#define SIZE_BLOCK 64
int setData(data_t *pd, char *str) {
  int aux = strlen(str) + 1;
  aux = SIZE_BLOCK * ((aux / SIZE_BLOCK) + 1);

  if ((pd->str = (char *)malloc(aux * sizeof(char)))) {
    strcpy(pd->str, str);
    pd->len = strlen(str);
    pd->size = aux;
    return 1;
  }
  return 0;
}

int addData(data_t *pd, char *str) {
  int lx = strlen(str);
  int sz;
  if (pd->size > pd->len + lx) {
    strcat(pd->str, str);
    pd->len += lx;
    return 1;
  }

  sz = pd->len + lx;
  sz = SIZE_BLOCK * ((sz / SIZE_BLOCK) + 1);
```

```
  if ((pd->str = (char *)realloc(pd->str, sz * sizeof(char)))) {
    strcat(pd->str, str);
    pd->len += lx;
    pd->size = sz;
    return 1;
  }
  return 0;
}

void freeData(const data_t *pd) { free(pd->str); }
```

Ejercicio 3:

El siguiente código puede genera inconvenientes en el sistema, o bien, el sistema puede interrumpir su funcionamiento. Lo invito a que analice lo que hace la función getAllocNombre, y corrija los errores de la aplicación.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE_BUFF 1000
char *getAllocNombre() {
  char *nombresFemeninos[] = {
      "María",     "Ana",       "Laura",     "Luisa",    "Sofía",     "Lucía",
      "Elena",     "Isabel",    "Carmen",    "Rosa",     "Patricia",  "Teresa",
      "Raquel",    "Natalia",   "Julia",     "Andrea",   "Clara",     "Victoria",
      "Eva",       "Sara",      "Paula",     "Beatriz",  "Monica",    "Olga",
      "Marta",     "Carolina",  "Mónica",    "Silvia",   "Diana",     "Gloria",
      "Susana",    "Lorena",    "Yolanda",   "Adela",    "Lidia",     "Antonia",
      "Fabiola",   "Camila",    "Manuela",   "Eloisa",   "Rocio",     "Liliana",
      "Celia",     "Lourdes",   "Rosario",   "Elisa",    "Alicia",    "Clotilde",
      "Esperanza", "Milagros",  "Constanza", "Renata",   "Gabriela",  "Adriana",
      "Marina",    "Eugenia",   "Fernanda",  "Rita",     "Josefina"};
  char *nombresMaculinos[] = {
      "Néstor",    "Juan",      "Carlos",    "Luis",     "Pedro",     "José",
      "Miguel",    "Alejandro", "Javier",    "Fernando", "David",     "Manuel",
      "Antonio",   "Francisco", "Pablo",     "Jorge",    "Rafael",    "Diego",
      "Daniel",    "Roberto",   "Alberto",   "Santiago", "José Luis", "Andrés",
      "Ricardo",   "Lorenzo",   "Ángel",     "Víctor",   "Enrique",   "Héctor",
      "Guillermo", "Eduardo",   "Ignacio",   "Óscar",    "Emilio",    "Hugo",
      "Félix",     "Tomás",     "Marcos",    "Raúl",     "Felipe",    "Nicolás",
      "Armando",   "Rogelio",   "Alfredo",   "Gustavo",  "Samuel",    "Ernesto",
      "Rodrigo",   "Esteban",   "Sergio",    "Adrián",   "Abel",      "Benjamín",
      "Arturo",    "César",     "Leonardo",  "Mariano",  "Mauricio"};
  char **s;
  char *sRet;
  int sz, i,cant;
  char sBuff[SIZE_BUFF];

  if (rand() % 2) {
    s = nombresFemeninos;
    sz = sizeof(nombresFemeninos) / sizeof(*nombresFemeninos);
  } else {
    s = nombresMaculinos;
    sz = sizeof(nombresMaculinos) / sizeof(*nombresMaculinos);
  }
```

```
  sBuff[0] = 0;
  cant = 1 + rand() % 4;
  for (i = 0; i < cant; i++) {
    strcat(sBuff, s[rand() % sz]);
    strcat(sBuff, " ");
  }
  sBuff[strlen(sBuff) - 1] = 0;
  sRet = (char *)malloc(SIZE_BUFF * sizeof(char));
  strcpy(sRet, sBuff);
  return sRet;
}

#define SIZE_TEST 100000
int main() {
  char **s;
  int i;

  while (1) {
    s = (char **)malloc(SIZE_TEST * sizeof(char *));
    if (!s) {
      printf("Error de Memoria");
      return 1;
    }
    for (i = 0; i < SIZE_TEST; i++) {
      s[i] = getAllocNombre();
    }

    for (i = 0; i < 5; i++)
      printf("%s\n", s[i]);

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