RAFAEL VIEIRA COELHO

ALOCAÇÃO DINÂMICA/ ERROS

PROGRAMAÇÃO 2

CAPÍTULO 27: ERROS

https://books.goalkicker.com/CBook/

ERRNO

Quando uma função falha, é definido um código de erro apropriado.

Em linguagem C, precisamos definir 3 valores:

Value	Meaning
EDOM	Domain error
ERANGE	Range error
EILSEQ	Illegal multi-byte character sequence

STRERROR

Trata-se de uma função da biblioteca **string.h** que proporciona uma representação legível do erro.

Precisamos das bibliotecas:

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

```c
int main(int argc, char *argv[])
{
    FILE *fout;
    int last_error = 0;

    if ((fout = fopen(argv[1], "w")) == NULL) {
        last_error = errno;
        /* reset errno and continue */
        errno = 0;
    }

    /* do some processing and try opening the file differently, then */


    if (last_error) {
        fprintf(stderr, "fopen: Could not open %s for writing: %s",
                argv[1], strerror(last_error));
        fputs("Cross fingers and continue", stderr);
    }

    /* do some other processing */

    return EXIT_SUCCESS;
}
```

PERROR

Para mostrar na tela o erro, precisamos da função perror() da biblioteca stdio.h

```c
int main(int argc, char *argv[])
{
    FILE *fout;

    if ((fout = fopen(argv[1], "w")) == NULL) {
        perror("fopen: Could not open file for writing");
        return EXIT_FAILURE;
    }
return EXIT_SUCCESS;
}
```

CAPÍTULO 46: ALOCAÇÃO DE MEMÓRIA

https://books.goalkicker.com/CBook/

ALOCAÇÃO DINÂMICA DE MEMÓRIA

Precisamos seguir sempre as seguintes ações:

(1) Alocar o espaço necessário:

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

```
person * myperson = (person *) malloc(sizeof(person));
```

ALOCAÇÃO DINÂMICA DE MEMÓRIA

(2) Usar o espaço reservado:

```
myperson->name = "John";
myperson->age = 27;
```

(3) Liberar o espaço reservado:

```
free(myperson);
```

ALOCAÇÃO DINÂMICA DE MEMÓRIA

As funções que podem ser usadas para alocação dinâmica (biblioteca **stdlib.h**) são: **malloc, calloc, realloc** e **free.**

No exemplo abaixo, está sendo alocado **10 espaços** de memória nos quais podem ser armazenados valores **inteiros.**

```
int *p = malloc(10 * sizeof *p);
if (p == NULL)
{
    perror("malloc() failed");
    return -1;
}
```

Caso não exista espaço ou ocorra algum erro, retorna **NULL.**

A função **realloc()** pode ser usada para realocar memória caso necessário.

ALOCAÇÃO DINÂMICA DE MEMÓRIA

A função **calloc()** retorna o bloco de memória inicializado em zero.

```
int *p = calloc(10, sizeof *p);
if (p == NULL)
{
    perror("calloc() failed");
    return -1;
}
```

LIBERANDO MEMÓRIA

Ao término do uso, deve-se liberar a memória alocada: **free()**

```c
int *p = malloc(10 * sizeof *p); /* allocation of memory */
if (p == NULL)
{
    perror("malloc failed");
    return -1;
}

free(p); /* release of memory */
/* note that after free(p), even using the *value* of the pointer p
   has undefined behavior, until a new value is stored into it. */

/* reusing/re-purposing the pointer itself */
int i = 42;
p = &i; /* This is valid, has defined behaviour */
```

REALOCANDO MEMÓRIA

Podemos precisar aumentar o espaço alocado: **realloc()**

```c
int main(void)
{
    int *p = malloc(10 * sizeof *p);
    if (NULL == p)
    {
        perror("malloc() failed");
        return EXIT_FAILURE;
    }

    p[0] = 42;
    p[9] = 15;

    /* Reallocate array to a larger size, storing the result into a
     * temporary pointer in case realloc() fails. */
    {
        int *temporary = realloc(p, 1000000 * sizeof *temporary);

        /* realloc() failed, the original allocation was not free'd yet. */
        if (NULL == temporary)
        {
            perror("realloc() failed");
            free(p); /* Clean up. */
            return EXIT_FAILURE;
        }

        p = temporary;
    }

    /* From here on, array can be used with the new size it was
     * realloc'ed to, until it is free'd. */

    /* The values of p[0] to p[9] are preserved, so this will print:
       42 15
    */
    printf("%d %d\n", p[0], p[9]);
```

```c
double sumAll(size_t n, size_t m, double A[n][m]) {
    double ret = 0.0;
    for (size_t i = 0; i < n; ++i)
        for (size_t j = 0; j < m; ++j)
            ret += A[i][j]
    return ret;
}

int main(int argc, char *argv[argc+1]) {
    size_t n = argc*10;
    size_t m = argc*8;
    double (*matrix)[m] = malloc(sizeof(double[n][m]));
    // initialize matrix somehow
    double res = sumAll(n, m, matrix);
    printf("result is %g\n", res);
    free(matrix);
}
```

ALOCAÇÃO DINÂMICA: MATRIZ

OBRIGADO!

RAFAELVC2@GMAIL.COM

https://books.goalkicker.com