MC102 - Algoritmos e Progração de Computador

Prof. Alexandre Xavier Falcão

25° Aula: Sistemas e Programas

1 Sistema

Um sistema é um conjunto de programas que compartilham uma ou mais bibliotecas de funções. Quando projetamos um sistema, cada estrutura abstrata e as funções que realizam operações com esta estrutura devem ser declaradas em arquivos com extensões .h e .c separados dos arquivos de programa. Por exemplo, nos arquivos contorno.h e matriz.h, nós declaramos estruturas e protótipos das funções.

1. Arquivo contorno.h

```
#ifndef CONTORNO_H
#define CONTORNO_H 1

#include <malloc.h>

typedef struct _ponto
{
  float x,y;
} Ponto;

typedef struct _contorno
{
  Ponto *p; /* vetor de pontos */
  int n;    /* numero de pontos */
} Contorno;

Contorno CriaContorno(int n); /* Aloca espaco para contorno com n pontos */
void     DestroiContorno(Contorno c); /* Desaloca espaco do contorno */

/* Aqui voce pode acrescentar os prototipos de outras
   funcoes que manipulam Contorno */

#endif
```

2. Arquivo matriz.h

```
#ifndef MATRIZ_H
#define MATRIZ_H 1

#include <malloc.h>

typedef struct _matriz {
  float **elem; /* elementos da matriz */
  int nlin,ncol; /* numero de linhas e colunas */
} Matriz;

Matriz CriaMatriz(int nlin, int ncol); /* Aloca espaco para matriz nlin x ncol */
void   DestroiMatriz(Matriz m); /* Desaloca espaco */

/* Aqui voce pode acrescentar os prototipos de outras
   funcoes que manipulam Matriz */


#endif
```

Nos arquivos contorno.c e matriz.c, nós incluímos os respectivos arquivos .h e declaramos os escopos dessas funções.

1. Arquivo contorno.c

```
#include "contorno.h"

Contorno CriaContorno(int n)
{
  Contorno c;

  c.n = n;
  c.p = (Ponto *)calloc(n,sizeof(Ponto));
  return(c);
}

void     DestroiContorno(Contorno c)
{
  if (c.p != NULL) {
    free(c.p);
    c.p = NULL;
  }
}

/* Aqui voce acrescenta os escopos das outras funcoes
   que manipulam Contorno */
```

2. Arquivo matriz.c

```
#include "matriz.h"

Matriz CriaMatriz(int nlin, int ncol)
{
  Matriz m;
  int i;

  m.nlin = nlin;
  m.ncol = ncol;
  m.elem = (float **)calloc(nlin,sizeof(float *));
  if (m.elem != NULL)
  for (i=0; i < nlin; i++)
    m.elem[i] = (float *)calloc(ncol,sizeof(float));
  return(m);
}

void DestroiMatriz(Matriz m)
{
  int i;

  if (m.elem != NULL){
    for (i=0; i < m.nlin; i++)
      free(m.elem[i]);
    free(m.elem);
    m.elem = NULL;
  }
}

/* Aqui voce acrescenta os escopos das outras funcoes
   que manipulam Matriz */
```

Para organizar o sistema, nós podemos criar uma estrutura de diretórios com raiz mc102, por exemplo, colocando os arquivos com extensão .h em mc102/include e os arquivos com extensão .c em mc102/src. Como um arquivo com extensão .c não possue a função main, ele não é considerado programa. Seu código após compilação tem extensão .o e é denominado **código objeto** (não executável). Os respectivos arquivos com extensão .o podem ser colocados em mc102/obj. Para que as estruturas e funções desses arquivos sejam disponibilizadas para uso por programas, os arquivos objeto devem ser "ligados" em um arquivo com extensão .a, i.e. arquivo biblioteca de funções. O *script* **Makefile** abaixo realiza esta tarefa colocando a biblioteca libmc102.a no diretório mc102/lib. O Makefile fica no diretório mc102.

```
LIB=./lib
INCLUDE=./include
BIN=./
SRC=./src
OBJ=./obj
```

```
#FLAGS= -g -Wall
FLAGS= -O3 -Wall

libmc102: $(LIB)/libmc102.a
echo "libmc102.a construida com sucesso..."

$(LIB)/libmc102.a: \
$(OBJ)/contorno.o \
$(OBJ)/matriz.o

ar csr $(LIB)/libmc102.a \
$(OBJ)/contorno.o \
$(OBJ)/matriz.o

$(OBJ)/contorno.o: $(SRC)/contorno.c
gcc $(FLAGS) -c $(SRC)/contorno.c -I$(INCLUDE) \
-o $(OBJ)/contorno.o

$(OBJ)/matriz.o: $(SRC)/matriz.c
gcc $(FLAGS) -c $(SRC)/matriz.c -I$(INCLUDE) \
-o $(OBJ)/matriz.o

apaga:
rm $(OBJ)/*.o;

apagatudo:
rm $(OBJ)/*.o; rm $(LIB)/*.a
```

2 Programas

Todo programa que quiser usar as estruturas e funções da biblioteca libmc102.a devem incluir os arquivos .h da biblioteca e devem ser compilados com o comando abaixo. Suponha, por exemplo, que o programa prog.c abaixo está no mesmo nível do diretório mc102.

1. Programa prog.c

```
#include "contorno.h"
#include "matriz.h"

int main()
{
  Contorno c;
  Matriz m;

  c = CriaContorno(100);
  m = CriaMatriz(5,9);
```

```
  DestroiContorno(c);
  DestroiMatriz(m);
  return(0);
}
```

2. Comando de compilação

```
gcc prog.c -I./mc102/include -L./mc102/lib -o prog -lmc102
```

2.1 Argumentos de programa

Em uma das aulas anteriores assumimos um nome fixo para os arquivos lidos e gravados em disco.

```
int main()
{
  Agenda a;

  a = LeAgenda("arq.txt");
  OrdenaAgenda(a);
  GravaAgenda(a,"arq-ord.txt");
  DestroiAgenda(a);
  return(0);
}
```

Porém, seria mais conveniente se esses nomes fossem passados como argumentos do programa na linha de execução (e.g. **programa arq.txt arq-ord.txt**). Isto é possível, pois a função main pode ser declarada com dois argumentos: int main(int argc, char **argv). O valor de **argc** é o número de argumentos em argv e **argv** é um vetor de *strings*, onde cada *string* contém um argumento do programa (e.g. pode ser um número real, inteiro, um caracter, ou uma cadeia de caracteres). No caso do programa acima, teríamos:

```
int main(int argc, char **argv)
{
  Agenda a;

  if (argc != 3) {
    printf("uso: %s <entrada> <saida>\n",argv[0]);
    exit(-1);
  }
  a = LeAgenda(argv[1]);
  OrdenaAgenda(a);
  GravaAgenda(a,argv[2]);
  DestroiAgenda(a);
  return(0);
}
```

Nos casos de argumentos inteiros e reais, as funções **atoi** e **atof** podem ser usadas para converter *string* nesses valores, respectivamente.

```
#include "contorno.h"
#include "matriz.h"

int main(int argc, char **argv)
{
  Contorno c;
  Matriz m;

  if (argc != 4){
    printf("uso: %s <numero de pontos> <numero de linhas> <numero de colunas>\n",argv[0]);
    exit(-1);
  }

  c = CriaContorno(atoi(argv[1]);
  m = CriaMatriz(atoi(argv[2]),atoi(argv[3]));

  DestroiContorno(c);
  DestroiMatriz(m);
  return(0);
}
```