MC202 - Estruturas de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Tópicos avançados em C

- Já sabemos que dados podem ser fornecidos pelo teclado e apresentados no console.

Tópicos avançados em C

- Já sabemos que dados podem ser fornecidos pelo teclado e apresentados no console.

- No entando, quando a quantidade de dados é muito grande, eles são lidos de ou escritos em arquivos.

- Já sabemos que dados podem ser fornecidos pelo teclado e apresentados no console.
- No entando, quando a quantidade de dados é muito grande, eles são lidos de ou escritos em arquivos.
- Esses arquivos podem ser codificados em formato texto (ASCII) ou binário.

Tópicos avançados em C

- Já sabemos que dados podem ser fornecidos pelo teclado e apresentados no console.
- No entando, quando a quantidade de dados é muito grande, eles são lidos de ou escritos em arquivos.
- Esses arquivos podem ser codificados em formato texto (ASCII) ou binário.
- Funções podem ter um número variável de argumentos (e.g., printf, scanf) e serem acessíveis por ponteiros.

Tópicos avançados em C

- Já sabemos que dados podem ser fornecidos pelo teclado e apresentados no console.
- No entando, quando a quantidade de dados é muito grande, eles são lidos de ou escritos em arquivos.
- Esses arquivos podem ser codificados em formato texto (ASCII) ou binário.
- Funções podem ter um número variável de argumentos (e.g., printf, scanf) e serem acessíveis por ponteiros.
- Por fim, programas podem ser organizados em sistemas de programas.

Agenda

- Funções com número variável de argumentos.
- Ponteiros para funções.
- Arquivo texto.
- Arquivo binário.
- Sistemas de programas.

Funções com número variável de argumentos

A função abaixo pode multiplicar uma quantidade variável de números – e.g., multiplica(2,3) e multiplica(2,3.5,20,50).

```
double multiplica(int num_args, ...) {
  double result = 1.0;
   va_list args; /* lista de argumentos */
   int i;

   va_start(args, num_args); /* prepara a lista para recuperar
                                argumentos após num_args */
   for(i = 0; i < num_args; i++) {
     result *= va_arg(args, double); /* recupera o próximo na lista */
   }
   va_end(args); /* finaliza o processo de recuperação */

   return(result);
}
```

Para usá-la é necessário incluir stdarg.h. Os argumentos são copiados para uma lista e acessados um a um.

Ponteiro de função

Podemos criar um ponteiro para a posição de memória onde o compilador gerou o código de uma função.

```
typedef float (*Operacao)(float v1, float v2); // cria o tipo apenas

// cria funções do tipo desejado

float Soma(float v1, float v2)
{
    return(v1 + v2);
}

float Produto(float v1, float v2)
{
    return(v1 * v2);
}
```

Ponteiro de função

```
int main()
{
    float v1, v2, res;
    char op;

    scanf("%f %f %c",&v1,&v2,&op);
    switch(op) {
      case '+':
        res = ExecutaOperacao(v1,v2,Soma);
        break;
      case '*':
        res = ExecutaOperacao(v1,v2,Produto);
        break;
        ...
    }     printf("%f %c %f = %f",v1,op,v2,res);
...}
```

Ponteiro de função

Soma e Produto guardam o endereço de memória onde está o código dessas funções, respectivamente. Este endereço pode ser armazenado em um ponteiro de função com os mesmos argumentos e que retorna float (i.e., do tipo Operacao).

```
float ExecutaOperacao(float v1, float v2, Operacao operacao)
{
    return(operacao(v1, v2));
}
```

Arquivo texto

Seja uma agenda de telefones definida pelos registros abaixo.

```
typedef struct _pessoa {
  char nome[100];    /* nome */
  char telefone[20]; /* telefone 0XX-XX-XXXXXXXX */
} Pessoa;

typedef struct _agenda{
  Pessoa *pessoa;      /* lista de pessoas */
  int     tamanho; /* tamanho da agenda */
} Agenda;
```

Arquivo texto

Seja uma agenda de telefones definida pelos registros abaixo.

```
typedef struct _pessoa {
  char nome[100];    /* nome */
  char telefone[20]; /* telefone 0XX-XX-XXXXXXXX */
} Pessoa;

typedef struct _agenda{
  Pessoa *pessoa;      /* lista de pessoas */
  int     tamanho; /* tamanho da agenda */
} Agenda;
```

Um arquivo texto pode ser criado no disco com o seguinte conteúdo.

5
José Maria;019-999451234
Roberta Oliveira;019-999152222
Carla da Silva;011-991432341
Carlos da Silveira;019-991785324
Raquel da Paixão;019-992715529

Arquivo texto

O arquivo pode ser acessado e carregado na memória primária em um vetor de registros a partir de um ponteiro para o seu endereço no disco.

```
Agenda *LeAgendaTexto(char *nomearq)
{
  FILE *fp=fopen(nomearq,"r"); /* gera apontador para o arquivo no
                                  disco. */
  Agenda *agenda = NULL;

  if (fp != NULL) {
    int tamanho;
    fscanf(fp,"%d\n",&tamanho);
    agenda = CriaAgenda(tamanho);
    for (int i=0; i < tamanho; i++){
      char linha[200];
      fscanf(fp,"%[^\n]s",linha); fscanf(fp,"\n");
      char **partes = QuebraCadeia(linha,";");
      sprintf(agenda->pessoa[i].nome,"%s",partes[1]);
      sprintf(agenda->pessoa[i].telefone,"%s",partes[2]);
      for (int j=0; j <= 2; j++)
        free(partes[j]);
      free(partes);
    }
    fclose(fp);
  } else {
    Erro("Arquivo %s inexistente","LeAgendaTexto",nomearq);
  }

  return(agenda);
}
```

Arquivo texto

As informações nos registros podem ser alteradas e o arquivo pode ser gravado de volta na memória secundária.

```
void  GravaAgendaTexto(Agenda *agenda, char *nomearq)
{
  FILE *fp=fopen(nomearq,"w"); /* gera apontador para arquivo no
                                  disco. */
  if (fp != NULL) {
    fprintf(fp,"%d\n",agenda->tamanho);
    for (int i=0; i < agenda->tamanho; i++){
      fprintf(fp,"%s;%s\n",agenda->pessoa[i].nome,agenda->pessoa[i].telefone);
    }
    fclose(fp);
  } else {
    Erro("Gravação do arquivo %s não permitida","GravaAgendaTexto",nomearq);
  }

}
```

Arquivo binário

Similarmente, o ponteiro para um arquivo binário no disco permite carregar suas informações em um vetor de registros na memória primária.

```
Agenda *LeAgendaBinario(char *nomearq)
{
  FILE *fp=fopen(nomearq,"rb"); /* gera apontador para o arquivo no
                                   disco. */
  Agenda *agenda = NULL;

  if (fp != NULL) {
    int tamanho;
    fread(&tamanho,sizeof(int),1,fp);
    agenda = CriaAgenda(tamanho);
    for (int i=0; i < tamanho; i++){
      fread(&agenda->pessoa[i],sizeof(Pessoa),1,fp);
    }
    fclose(fp);
  } else {
    Erro("Arquivo %s inexistente","LeAgendaBinario",nomearq);
  }

  return(agenda);
}
```

Arquivo binário

Novamente, após manipulação, as informações nos registros podem ser gravadas de volta no disco.

```
void GravaAgendaBinario(Agenda *agenda, char *nomearq)
{
  FILE *fp=fopen(nomearq,"wb"); /* gera apontador para o arquivo no
                                   disco. */
  if (fp != NULL) {
    fwrite(&agenda->tamanho,sizeof(int),1,fp);
    for (int i=0; i < agenda->tamanho; i++){
      fwrite(&agenda->pessoa[i],sizeof(Pessoa),1,fp);
    }
    fclose(fp);
  } else {
    Erro("Gravação do arquivo %s não permitida","GravaAgendaBinario",nomearq);
  }
}
```

Arquivos texto x binário

- Arquivos texto permitem o uso de editores de texto para visualizar e alterar o conteúdo deles no disco.

Arquivos texto x binário

- Arquivos texto permitem o uso de editores de texto para visualizar e alterar o conteúdo deles no disco.

- Arquivos binários constumam ocupar muito menos espaço em disco, o que se torna vital quando o volume de informações é muito grande.

Arquivos texto x binário

- Arquivos texto permitem o uso de editores de texto para visualizar e alterar o conteúdo deles no disco.
- Arquivos binários constumam ocupar muito menos espaço em disco, o que se torna vital quando o volume de informações é muito grande.
- Arquivos binários também permitem acessar e carregar um único registro em memória principal para alteração.

Arquivos texto x binário

```
bool BuscaPessoaAtualizaTelefoneDisco(char *nomearq, char *nome, char *telefone)
{
  FILE *fp = fopen(nomearq,"r+");

  if (fp != NULL) {
    int tamanho;
    fread(&tamanho,sizeof(int),1,fp);
    Pessoa pessoa;
    for (int i=0; i < tamanho; i++){
      fread(&pessoa,sizeof(Pessoa),1,fp);
      if (CadeiasIguais(pessoa.nome,nome)){
        sprintf(pessoa.telefone,"%s",telefone);
        fseek(fp,-sizeof(Pessoa),SEEK_CUR); /* retorna o apontador para o início do
                                               registro lido */
        fwrite(&pessoa,sizeof(Pessoa),1,fp); /* atualiza o registro */
        fclose(fp);
        return(true);
      }
    }
  } else {
    Erro("Arquivo %s inexistente","BuscaPessoaAtualizaTelefoneDisco",nomearq);
  }

  return(false);
}
```

A busca, no entanto, pode ser otimizada através de índices – estruturas de dados que armazenam uma chave de busca e o deslocamento em bytes (*offset*) para acesso direto ao registro.

Arquivos texto x binário

- No exemplo dado, usamos registros de 120 bytes e para poucas informações, o arquivo binário é maior do que o arquivo texto.

Arquivos texto x binário

- No exemplo dado, usamos registros de 120 bytes e para poucas informações, o arquivo binário é maior do que o arquivo texto.

- Poderíamos, no entanto, usar registros de tamanhos variáveis, gravando o índice no cabeçário do arquivo.

Arquivos texto x binário

- No exemplo dado, usamos registros de 120 bytes e para poucas informações, o arquivo binário é maior do que o arquivo texto.
- Poderíamos, no entanto, usar registros de tamanhos variáveis, gravando o índice no cabeçário do arquivo.
- Ao carregar o índice em memória, a busca pelo deslocamento em bytes fica bem mais rápida e a função fseek pode ser usada para acessar diretamente o registro em disco.

Arquivos texto x binário

- No exemplo dado, usamos registros de 120 bytes e para poucas informações, o arquivo binário é maior do que o arquivo texto.
- Poderíamos, no entanto, usar registros de tamanhos variáveis, gravando o índice no cabeçário do arquivo.
- Ao carregar o índice em memória, a busca pelo deslocamento em bytes fica bem mais rápida e a função fseek pode ser usada para acessar diretamente o registro em disco.

O código agenda.c apresenta um exemplo de manipulação de arquivos texto e binário com registros de tamanho fixo.

Sistemas de programas

Para concluir esta parte introdutória do curso, vamos ver como definir bibliotecas de funções e como integrar elas quando compilamos o código de um programa (código de sistemas-programas).