Linguagem C ANSI:
Aula 2 – Estrutura do Programa
Entrada e Saída Padronizada

Software Básico, turma A

(Baseado no Curso de Linguagem C da UFMG)

Prof. Marcelo Ladeira – CIC/UnB

Sumário

- **Estrutura do Programa**
  - O Comando **return**
  - Protótipos de Funções
  - O Tipo **void**
  - Arquivos Cabeçalhos
  - Regras de Escopo
  - Variáveis **register**
  - Passagem de Parâmetros por Valor e Por Referência
  - Vetores como Argumentos de Funções
  - Recursividade
  - Outras Questões
- **Entrada e Saída Padronizada**

Funções

- **Permitem modularizar o programa**
  - **Aumenta a legibilidade do código**
    - **Esconde detalhes operacionais do código.**
  - **Aumenta a manutenibilidade do código.**
  - **Aumenta a produtividade do programador**
    - **Favorece o re-uso de código.**
    - **Favorece a prototipação**
      - **Facilita testar por partes e não todo o programa.**
  - **Abstração funcional mas sem encapsulamento**
    - **Facilita entendimento da funcionalidade implementada.**

Para testar uma função que dependa de outra ainda não programada, pode-se substituir a não programada por uma função vazia tipo strub() {}.

Funções
Modularização

- Programação de grandes programas exige modularização (dividir para conquistar).
  - cada módulo com funcionalidade específica
    - executa um conjunto limitado de operações sobre uma quantia limitada de dados.
    - facilita o entendimento do projeto.
- Enfoques para modularização:
  - decomposição funcional
    - módulo: subprograma, função ou procedimento.
  - decomposição de dados:
    - módulo: objeto

Modularização
Informação Escondida

- Na decomposição funcional o programador deve conhecer detalhes dos OD e das operações.
  - Métodos como refinamento passo a passo, programação modular, programação topdown são todos referentes a abstração.
- Na decomposição de dados é necessário saber apenas a especificação dos tipos de dados e as operações disponíveis.

Encapsulamento

- **Com informação encapsulada em abstração:**
  - **a) o usuário não necessita conhecer informações ocultas para usar a abstração.**
  - **b) o usuário não pode usar a informação oculta de forma direta, mesmo se desejar.**
- **Agrupa componentes do programa em unidades ⇒ facilita o uso pelo usuário.**
- **É uma construção lingüística que suporta módulos com informações escondidas.**

Facilita a manutenção: alterar apenas estruturas encapsuladas e não todo o programa.

Abstração

- Descreve módulos em 2 partes: interface e implementação.
  - Interface:
    - quais serviços são providos pelo módulo.
    - como eles podem ser acessados.
    - conjunto de entidades exportadas pelo módulo
      - os módulos clientes importam essas entidades.
  - Implementação:
    - descreve os detalhes internos do módulo.
    - codifica os algoritmos que implementam as operações.

Funções

- Subprograma é um mecanismo básico de abstração, existente em muitas LP.
  - permite fácil modificação do programa.
- Informação escondida
  - uma questão de projeto do programa.
- Encapsulamento
  - uma questão de projeto de LP, pois cabe a LP impedir o acesso as informações escondidas.

Funções

- Facilitam desenvolver via refinamentos sucessivos
- C foi projetada para uso fácil e eficiente de funções
  - Em geral, programas C possuem muitas funções ao invés de apenas algumas funções grandes.
    - Programa é dividido em um ou mais arquivos cabeçalhos com definições ou arquivos fontes
      - Arquivos fontes podem ser compilados separadamente gerando módulos objetos
        - Esses módulos devem ser ligados junto para resolver chamadas entre eles.
        - A ligação pode usar módulos objetos de bibliotecas do sistema ou do usuário.
  - Seus protótipos viabilizam a checagem de tipos estática.

Funções

- **Comunicação entre Funções**
  - **Via lista de parâmetros**
    - **Parâmetros formais**
      - **Argumentos na declaração da função**
    - **Parâmetros atuais**
      - **Argumentos na chamada da função**
    - **Uso de parâmetros facilita a abstração funcional**
      - **Torna a definição das funcionalidades mais auto contida**
  - **Via variáveis externas**
    - **Permitem efeitos colaterais**
  - **Via registradores**
  - **Outros métodos (pilha, arquivo, memória compartilhada..)**

C opta por passagem de parâmetros. É uma boa opção? Por quê ? Compare com a opção adotada na linguagem Assembly.

Funções

- **Declaração de Funções**
  - **Sintaxe Geral**

    tipo_de_retorno nome_da_função (declaração_de_parâmetros)
    {
    corpo_da_função
    }
  - **O tipo** int **é o tipo da função assumido por falta**
  - **Sintaxe da Declaração dos Parametros**
    - **tipo nome1, tipo nome2, ... , tipo nomeN;**
  - **O aninhamento de funções não é permitido.**
    - **Todas as funções possuem o mesmo nível lexicográfico.**

O Comando return

- **Retorna um valor a quem a chamou.**
- **Sintaxe Geral**

  return *expressão*;

  return;
- **Comentários**
  - Valor retornado pode ser ignorado.
  - **void** não retorna valor.
  - Nenhum valor é retornado ao se encontrar o "}" que fecha o corpo da função.
  - Pode haver conversão de tipo do valor retornado.

```
#include <ctype.h>
double atof(char s[]) {
double val, power;
int i, sign;
for (i = 0; isspace(s[i]); i++)  ;
sign = (s[i] == '-') ? -1 : 1;
if (s[i] == '+' || s[i] == '-') i++;
for (val = 0.0; isdigit(s[i]); i++)
     val = 10.0 * val + (s[i] - '0');
if (s[i] == '.') i++;
for (power = 1.0; isdigit(s[i]); i++) {
     val = 10.0 * val + (s[i] - '0');
     power *= 10;
}
return sign * val / power;
}
```

Função tipo atof() para converter uma string s em double. KR, pág. 62.

Protótipo de Funções

- **Definem a assinatura da função**
  - **Número, ordem e tipo dos argumentos.**
  - **Tipo retornado pela função.**
- **Sintaxe Geral**

  tipo_de_retorno nome_da_função (declaração_de_parâmetros);
- **Comentários**
  - **São necessários mesmo se as funções são declaradas antes da main().**

Relembrem que permitem a checagem de tipos estática evitandos erros de tipos na chamada da função.

Protótipos de Funções

- **Nomes dos argumentos não precisam ser os mesmos dos parâmetros formais**
  - **São opcionais**

    int power (**int** base, **int** n);

    int power (**int** , **int**);
  - **Para documentação devem sempre ser fornecidos com nomes mnemônicos**

```
#include <stdio.h>
float Square (float a);      /* sem esse
     protótipo haveria erro. Por quê? */
int main () {
float num;
printf ("Entre com um numero: ");
scanf ("%f",&num);
num=Square(num);
printf ("\n\nO seu quadrado vale: %f\n",
     num);
return 0;
}
float Square (float a)    /* declaração da
     função após ser chamada */ {
return (a*a); }
```


UFMG, pág. 67.

O Tipo void

- **Permite declarar subrotinas**

  void nome_da_função (declaração_de_parâmetros);

  - Nesse caso o **return** não é necessário.

- **Permite declarar funções sem argumentos**

  tipo_de_retorno nome_da_função (**void**);

- **Permite declarar subrotinas sem argumentos**

  void nome_da_função (**void**);

  - Também nesse caso o **return** não é necessário.

Arquivos Cabeçalhos

- **Centralizam definições e declarações compartilhadas entre arquivos do programa**
  - **Contêm protótipos de funções, declarações de variáveis e constantes simbólicas.**
    - **Arquivo cabeçalho possui extensão ponto h.**
    - **As funções são previamente compiladas.**
      - **O código objeto está em bibliotecas que serão ligadas ao programa para resolver as referências externas.**
  - **Essas informações são usadas para a checagem estática de tipos.**
  - **Centralização permite manter apenas uma cópia dessas informações.**
    - **Facilita a manutenção do programa quando ele evolui.**

O programador pode definir seus próprios arquivos cabeçalhos. Quantos arquivos cabeçalhos devemos projetar por programa?

Regras de Escopo

- **Determinam o uso e a validade dos nomes nas diversas partes do programa**
  - **O escopo é a parte do programa dentro da qual o nome pode ser usado.**
  - **Regras de escopo**
    - **Variáveis locais**
      - **O escopo é o bloco no qual é declarada.**
      - **As variáveis locais e parametros formais de funções do mesmo nome em funções diferentes não são relacionadas.**
    - **Variável externa ou nome de função**
      - **O escopo vai do ponto em que são declaradas até o final do arquivo sendo compilado.**

Variáveis externas também são denominadas variáveis globais.

Regras de Escopo
Variáveis Locais

- **Locais ao bloco onde declaradas.**
- **Alocadas na ativação e desalocadas na saída.**
  - **Por isso são chamadas automáticas.**
    - **Registro de ativação as suporta em funções.**
    - **Não retêm valores entre ativações.**
      - **Seus valores iniciais são indefinidos.**
- Uso do modificador **auto** na declaração é opcional.

```
func1 (...) {
int abc,x;        /* abc e x locais a func1 */
...
}
func (...) {
auto int abc;  /* abc local a func */
...
}
void main () {
int a,x,y;        /* a,x e y locais a main */
for (...)
{
float a,b,c;      /* a, b e c locais ao for */
...
}
...
}
```

Regras de Escopo
Parâmetros Formais

- São os argumentos de definição de uma função.
- São variáveis automáticas.
  - Os valores dos argumentos atuais são utilizados para iniciar as variáveis locais correspondentes
    - Alterações em seus valores não repercutem fora da função.
      - Isso é verdade mesmo para os ponteiros passados como argumentos de função.
        - Alterar um ponteiro não repercute no valor do parâmetro atual.
        - Alterar o conteúdo do objeto apontado repercute externamente.

Regras de Escopo
Variáveis Externas

- São declaradas fora de qualquer função.
  - Funções são externas.
    - Aninhamento de funções não é permitido em C ANSI.
- Possuem a propriedade de ligação externa
  - Referência ao mesmo nome são ao mesmo objeto.
    - Mesmo se feitas de funções compiladas em separado.
    - Variáveis locais de mesmo nome escondem as externas.
  - São globalmente visíveis
    - Constituem um alternativa ao método de parâmetros
      - Use se a lista de parâmetros for longa.
  - Permanentes, retêm valores entre ativações de funções
    - Provocam efeitos colaterais reduzindo a manutenibilidade
    - Ocupam memória todo o tempo.

São análogas ao bloco COMMON do Fortran ou à variáveis do bloco mais externo do Pascal.

Regras de Escopo
Variáveis Externas

- **Declaração anuncia as propriedade da variável**
  - **Requerida se a variável for referenciada antes de ser definida.**
    - **extern**
  - **A variável pode ter mais de uma declaração**
- **Definição aloca espaço**
  - **Variável deve ser definida em apenas um local.**
  - **Admite iniciação.**

- **Declaração**

```
extern int sp;
extern double val[];
```

- **Definição**

```
main() { ... }
int sp = 0;
double val[MAXVAL];
void push (double f) { ... }
double pop (void) { ... }
```

Regras de Escopo
Variáveis Estáticas

- **Globais**
  - **Limita o escopo ao resto do arquivo fonte**
    - **Isola partes do programa escondendo nomes**
      - **evita mudanças acidentais nas variáveis globais.**
  - **Aplica-se a variáveis e funções.**
    - **Funções estáticas são visíveis somente no resto do arquivo fonte onde são declaradas.**
  - **Exemplo**

```
static char buf[BUFSIZE]; /* buffer para ungetch */
static int bufp = 0;          /* próxima posição livre no buffer */
int getch (void) { ... }
void ungetch (int c) { ... }
```

Regras de Escopo
Variáveis Estáticas

- **Locais**
  - **Provêm armazenamento permanente privado dentro de uma função.**
    - **Valores são mantidos entre ativações da função.**
    - **A definição pode conter iniciação da variável**
      - **Mas é executada apenas na chamada inicial da função.**
  - **Exemplo**

```
int count (void) {
static int num=0;
num++;
return num;
}
```

Variáveis register

- **Informa que a variável vai ser muito usada**
  - **Aplica-se às variáveis automáticas e parâmetros formais.**
  - **Solicita ao compilador alocá-la em registrador**
    - **O compilador pode ignorar o pedido.**
      - **A idéia é alocar em memória mais rápida, se disponível.**
    - **Pode haver restrições de número e tipo da variável**
      - Em geral os tipos **char** e **int** são aceitos**.**

- Exemplo típico

```
main (void)
{
register int count;
for (count=0;count<10;count++)
{
...
}
return 0;
}
```

Passagem de Parâmetros por Valor

- Para cada parâmetro é criada uma variável local
  - iniciada com o valor do parâmetro atual.
    - Alterações em seus valores não repercutem fora da função.
  - os parâmetros atuais não são atualizados.
    - mesmo se são ponteiros.
- Método evita efeito colaterais
  - Maior manutenibilidade
    - Legibilidade e segurança
- Traz transtornos se se deseja que as alterações permaneçam

```
#include <stdio.h>
float sqr (float num);
void main () {
float num,sq;
printf ("Entre com um numero: ");
scanf ("%f",&num);
sq=sqr(num);
printf ("\n\nO numero original eh:
    %f\n",num);
printf ("O seu quadrado vale: %f\n",sq);
}
float sqr (float num)
{
num=num*num;
return num;
}
```


UFMG, pág. 74. Alterações em num não refletem fora da função sqr.

Passagem de Valores por Referência

- **Para cada parâmetro é criada variável local**
  - **iniciada com o endereço no parâmetro atual.**
    - **Alterações repercutem fora da função.**
  - **Parâmetros formais são declarados como ponteiros.**
  - **Parâmetros atuais são endereços**
- **Facilita a ocorrência de efeitos colaterais**
  - **Reduz a legibilidade.**
    - **Mas aumenta a eficiência.**

```
#include <stdio.h>
void Swap (int *a,int *b);
void main (void) {
int num1,num2;
num1=100;
num2=200;
Swap (&num1,&num2);
printf ("\n\nEles agora valem %d
     %d\n",num1,num2);
}

void Swap (int *a, int *b) {
int temp;
temp=*a;
*a=*b;
*b=temp;
}
```

Vetores como Argumentos de Funções

- **O vetor é passado por referência**
  - **O nome do vetor é o endereço passado.**
- **Exemplo**
  - Passagem de int matrx [50] como argumento de func()
    - Formas de definição para func()

```
void func (int matrx[50]);
void func (int matrx[]);
void func (int *matrx);
```

Iniciação

- **Na ausência de iniciação explícita**
  - **Zero para variáveis externas ou estáticas**
  - **Lixo para variáveis automáticas ou** register
- **Iniciação de escalar**
  - **Automáticas e register.**
    - **Expressão**

```
int binsearch (int x, int v[],
    int n)
{
int low = 0;
int high = n - 1;
int mid;
...
}
```

- **Iniciação de escalar**
  - **Externas e estática**
    - **Expressão constante iniciada uma única vez.**

```
int x = 1;
char contrabarra = '\\';
long day = 1000L * 60L * 60L *
    24L;  /* milisegundos/dia */
```

- **Iniciação de vetores**
  - **Lista de valores separados por vírgula e entre chaves.**
    - **Lista menor que tamanho do vetor**
      - **Zero para estática, externa e automática**
    - **Lista maior que tamanho**
      - **Erro**

Recursividade

- Função pode chamar a si mesmo direta ou indiretamente.
- Funções em C podem ser recursivas
  - Requer critério de parada
  - Recursividade é suportada pelo registro de ativação
    - Tende a utilizar muita memória e é lenta
      - Deve ser evitada sempre que possível.
    - Código é mais compacto e fácil de entender
      - Recomendada em procedimentos com árvores.

Outras Questões

- **Funções devem ser implementadas da forma mais geral possível**
  - **Facilita o reuso**
- **Evite o uso de variáveis externas**
- **Chamada a função consome tempo e recurso**
  - **Se rapidez for necessária, implemente o código sem usar funções!**

Porquê?

E/S Padrão ANSI

Nome	Função
fopen	Abre um arquivo
fclose	Fecha um arquivo
fputc()	Grava um caracter em um arquivo
fgetc()	Lê um caracter de um arquivo
fseek()	Posiciona ponteiro no arquivo
fprintf()	Gravação formatado em arquivo
fscanf()	Leitura formatada de arquivo
feof()	Testa se atingiu fim de arquivo
ferror()	Testa se houve erro
rewind()	reposiciona o ponteiro arq no inicio
remove()	apaga um arquivo
fflush()	grava buffer

Esses funções serão detalhadas a seguir ....

Abrindo Arquivos: Bufferizados

FILE *fopen (const char* nome_externo, const char* modo)

Modo	Significado
r	abre arquio texto para leitura
w	cria arquivo texto para escrita
a	estende (append) arquivo texto
rb	abre arquivo binário para leitura
wb	abre arquivo binário para escrita
ab	estende (append) arquivo binário,
r+	abre arquivo texto para leitura e escrita
w+	cria um arquivo texto para leitura e escrita
a+	estende ou cria arquivo texto para leitura e escrita
r+b, w+b ou a+b	similar, para arquivo binário

```
FILE*   arq;
char    cadeia [101] ;
printf("\n informe o nome do arquivo >"); fgets (cadeia, 101, stdin);
if ( (arq = fopen(cadeia, "a+")) == NULL) {printf ("Erro abrindo  %s",
cadeia); exit(1); }
```


arquivos abertos pelo S.O.: stdin, stdout, stderr, stdprn, stdaux

Checando Fim de Arquivo

- **Macro int EOF = -1**
  - **Na biblioteca GNU, muitas funções retorna o valor dessa macro para indicar fim de arquivo ou alguma condição de erro.**
- **Function int feof (FILE *stream)**
  - **Esta função retorna um valor diferente de 0 (zero) se, e só se, um fim-de-arquivo é encontrado para o fluxo stream.**
    - **Ela é declarada em `stdio.h`.**

Lendo Um Caracter

int fgetc (FILE *stream)

- **Lê o próximo char como um unsigned char do arquivo stream**
- **Retorna o valor de char convertido para int**
- **Retorna EOF se chegou ao fim-de-arquivo ou houver algum erro**

int getc (FILE *stream)

- **Mesmo que fgetc, exceto que é implementada via macro e otimizada para desempenho.**

int getchar (void)

- **Similar a getc com stdin como valor para stream**

Gravando Um Caracter

int fputc (int c, FILE *stream)

- Converte c para o tipo unsigned char e o grava em stream
  - Retorna o caracter c se a operação foi bem sucedida.
  - Retorna EOF se ocorreu um erro.

int putc (int c, FILE *stream)

- Mesmo que fputc em termos de funcionalidade
  - Implementado como macro é mais eficiente do que fputc.

int putchar (int c)

- Similar a putc com stdout como valor para stream

Lendo Strings

char * fgets (char *s, int count, FILE *stream)

- **Grava caracteres de stream em s até encontrar um ‘\n’, armazenando-o junto com um ‘\0’ no final da string em s.**
  - **Se count-1 for encontrado antes de achar um ‘\n’ então só count-1 caracteres + ‘\0’ serão incluidos em s.**
  - **Em geral, s deve ter tamanho count caracteres.**
  - **fgets não avança sobre outras áreas como o faz gets, se count for do tamanho de s.**
  - **stream pode ser stdin**

O que acontece se stream contém o caracter nulo? Use getline !

Lendo Strings

char *gets (char *s)

- **Lê caracteres de stdin até encontrar um ‘\n’, e os armazena na string s.**
  - **Descarta o ‘\n’ e introduz um ‘\0’ no final da string em s.**
  - **Se for encontrado um erro ou um fim-de-arquivo então retorna um ponteiro nulo, senão retorna s.**
  - **Se até encontrar ‘\n’ houver mais caracteres que o espaço em s, então há um avanço sobre outras áreas, gerando erros imprevistos.**

DESACONSELHÁVEL O USO !!!!

Gravando Strings

int fputs (const char *s, FILE *stream)

- Grava a string s em stream.
  - O null caracter de s não é escrito.
  - Não adiciona o caracter newline (não pula de linha), exceto se stream for a saída padrão.
- Retorna um valor não negativo (caracteres gravados)
  - Em caso de erro, retorna EOF.

int puts (const char *s)

- Grava a string s em stdout, seguindo de um newline.
  - O caracter nulo de s não é escrito.
  - Adiciona o caracter newline ao final de s.
- É a mais conveniente para imprimir mensagens simples.
- Exemplo: puts (“isto é uma msg”)

RECOMENDADA !

Lendo ou Gravando Blocos

Ler ou gravar blocos de dados

- **Em arquivos binários**
- **Em arquivos de texto operando sobre bloco de dados**
  - **Mais eficiente que ler ou escrever caracteres ou linhas.**

size_t fread (void *data, size_t size, size_t count, FILE *stream)

- **Lê stream até count objetos de tamanho size no arranjo data.**
- **Retorna o número de objetos, completos, realmente lidos, o qual pode ser menor do que count se encontrar um EOF ou houve erro.**

size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)

- **Escreve até count objetos de tamanho size do arranjo data para o arquivo stream.**
- **Retorna count. Qualquer outro valor indica condição de erro.**

Reportando Erros de I/O

```
#include <errno.h>          /* erros de acesso a arquivos, para impressão */
#include <stdio.h>          /* biblioteca padrão de E/S */
#include <stdlib.h>         /* tipos comuns, funções de conversão, ... */
#include <string.h>         /* funções para manipulação de strings. */
FILE * open_sesame (char *name)
{
     FILE *stream;
     errno = 0;
     stream = fopen (name, "r");
     if (stream == NULL)         {
        fprintf (stderr, "%s: Couldn't open file %s; %s\n",
              program_invocation_short_name, name, strerror (errno));
        exit (EXIT_FAILURE);
     }
     else
              return stream;
}
```


The GNU C Library 2.2 - Chapter 2: Error Reporting, page 28-29.

Posicionando Ponteiro de arquivo

int fseek (FILE *stream, long int offset, int origem)

- **desloca o ponteiro de stream offset posições**
- **o valor de origem precisa ser uma das constantes:**
  - **SEEK_SET = desloca relativo ao início do arquivo**
  - **SEEK_CUR = desloca relativo a posição corrente**
  - **SEEK_END = desloca relativo ao final do arquivo**
- **retorna**
  - **zero para indicar sucesso ou não zero caso contrário**

esvazia buffers!

Posicionando Ponteiro de arquivo

long int ftell (FILE *stream)

- retorna a posição corrente no arquivo stream.
- retorna –1 se ocorreu alguma falha porque:
  - a stream não suporta posicionamento
  - a posição não pode ser representada por um long int
  - ou ainda por outras razões

void rewind (FILE *stream)

- posiciona ponteiro de stream no início.
- é equivalente ao fseek com os argumentos offset igual a 0L e origem igual a SEEK_SET :
  - exceto que o valor de retorno é descartado e o indicador de erro é zerado.
  - como se o arquivo tivesse sido aberto agora.

0L - sufixo para objeto de dado do tipo long int

Caracteres Especiais em Arquivos

- Caracter ^Z (ASCII 26) como eof
  - O MSDOS o usa como último caracter de um arquivo tipo texto.
  - O Windows também o adota.
- O par CR-LF (ASCII 13 e 10) como EOL
  - Alguns sistemas usam esse par para indicar fim de linha em arquivos do tipo texto.
- Assistência não requerida (problemas?)
  - Alguns sistemas adiciona o par CR-LF quando encontram CR sozinho .
  - Outros (VMS) remove CR e o substitui por um contador de caracteres na linha.