Programação com a Linguagem C

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Fevereiro/2011

Objetivo

- Apresentar a linguagem de programação C, destacando as boas práticas de programação e os principais problemas encontrados com o uso não disciplinado desta linguagem.

Programa – Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Referências

- C: A Software Engineering Approach, 3rd edition
  - Peter A. Darnell
- C Primer Plus, 5th Edition
  - Stephen Prata
- The C Programming Language, 2nd edition
  - Brian W. Kernigham, Dennis M. Ritchie
- Livros e tutoriais sobre a linguagem C em geral

Linguagem de Programação C

- **Introdução**
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Introdução 1/9

- Linguagem C criada em 1972
- Dennis M. Ritchie
- AT&T Bell Labs
- Linguagem para programação de sistemas
  - Escrever kernel e programa de sistemas
- Linguagem C tem alto nível
  - Legibilidade
  - Portabilidade
- Linguagem C tem baixo nível
  - Facilmente mapeável para assembly

Introdução 2/9

- Em 1973 Dennis Ritchie e Ken Thompson
  Reescrevem maior parte do Unix em C
- Mostram seu valor como linguagem para a programação de sistemas
- Cada porte de Unix para uma nova arquitetura exigia
  um compilador C para aquela arquitetura
  - O sucesso ou fracasso de C e Unix estavam amarrados

Introdução 3/9

- Especificação inicial:
  Documento escrito por Ritchie "The C Reference Manual"
- Em 1977 Ritchie & Brian Kernighan escrevem o livro

  "The C Programming Language"
- Conhecido como o "padrão K&R"

Introdução 4/9

- No inicio C era usada apenas em Unix
- Existiam diferentes variações de Unix
- Mas C era razoavelmente uniforme

Introdução 5/9

- Surgem os Personal Computers (PCs)
- Popularização do C (MS-DOS)
- C em diferentes máquinas e sistemas operacionais
- Desenvolvedores de compiladores adicionam suas features favoritas
  (porém divergentes)
  - 1988: cerca de 10 compiladores C para MS-DOS disputam o mercado
- Passa a existir diversas variantes de C
- Cada uma divergindo em pequenos aspectos
- Mas o suficiente para perder a portabilidade

Introdução 6/9

- ANSI – American National Standards Institute
  - Organização que define padrões
- Em 1983 comitê X3J11
  - Representantes dos principais vendedores de compiladores C
  - Diversas empresas
- Final: X3-159-1089 C Standard
- Ratificado em 1989: **ANSI-C**

Introdução 7/9

- **ISO-C**
- ISO adotou o mesmo padrão em 1990
  - Idêntico ao ANSI-C
- ISO/IEC 9899:1990
- **Clean C**
- C++ é quase um super-set de ANSI-C
  - Não suporta algumas coisas do C K&R
- Clean C é o sub-set do ANSI C aceito pelo compilador C++

Introdução 8/9

- Reputação de C inclui:
  - É misteriosa
  - É confusa
  - Promove hábitos ruins de programação
- Parte do problema: Caracteres de pontuação são usados com semânticas importantes, por exemplo "{", ":", "]", etc
  - Assusta no início
- C possui leis liberais
  - Conversão de tipos, uso de pointers, etc
  - Lema da linguagem C: "Confie no programador"
  - Ainda é melhor que muitas linguagens script populares

Introdução 9/9

- Liberdade exige responsabilidade e disciplina
- Programador deve evitar truques desnecessários
- **Programa bom versus programa que funciona**
- **Programa bom:**
  - Funciona
  - É fácil de ler
  - É fácil de manter

Linguagem de Programação C

- Introdução
- **Primeiro programa e informações essenciais**
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Anatomia de uma função em C

```
/*
 * Author: P. Margolis
 * Initial coding: 3/87
 * Returns the square of num
 */

int square( int num )
{
  int answer;

  answer = num * num;          /* Does not check for overflow */
  return answer;
}
```


Comentários

- Inclua um comentário para cada função
  - Descreve o que a função faz em um alto nível de abstração
- Se tiver uma boa razão para violar um bom estilo de programação em dado momento: documente a razão
- Use comentários para marcar seções de código incompletas
- Use comentários para anotar futuras possíveis mudanças
- Se o nome da variável não basta, use comentários para explica-la
  - Mas o nome deveria ser auto-explicativo
- Não use comentários para sempre descrever como um trecho de código atinge seu objetivo
- Não repita o código no comentário
  - Inclua informações adicionais

Função main

```
#include <stdlib.h>

int main( void )
{
  extern int square(int);
  int solution;

  solution = square (5);
  exit(0);
}
```

Comentários aninhados

```
/* Isto é um comentário
*
/* Isto é um comentário interno errado */
*
* Esta linha será considerada código
*/
```


printf

```
#include <stdio.h>      /* Header file of printf() */
#include <stdlib.h>

int main( void )
{
  extern int square();
  int solution;

  solution = square( 27 );
  printf( "The square of 27 is %d\n", solution );
  exit( 0 );
}
```


printf

- **%c** caracter
- **%s** string (array de caracter terminado com \0)
- **%x** inteiro hexadecimal
- **%f** número com ponto flutuante
- **%o** octal (muito importante nos anos 60-70)

```
printf("Esses três valores: %d %d %d\n", num1, num2, num3);

printf("O texto pode passar "
       "por várias linhas "
       " sucessivas\n" );
```


scanf

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

int main( void )
{
    extern int square();
    int solution;
    int input_val;

    printf( "Enter an integer value: " );
    scanf( "%d", &input_val );
    solution = square( input_val );
    printf ( "The square of %d is %d\n", input_val, solution );
    exit( 0 );
}
```


Pré-processador

- **Diretivas iniciam com #**
- #include <filename>
- #include "filename"
- #define TAMANHO_TABELA 32
  - Substituição textual
  - Usar no lugar de números mágicos

Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- **Tipos de dados elementares**
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Hierarquia dos tipos de dados

- Tipos de dados:
  - Void
  - Agregados
  - Escalares
- Tipos de dados escalares
  - Pointers
  - Enumeração
  - Tipos aritméticos
    - Inteiros
    - Ponto flutuante

Tipos de dados aritméticos

- Tipos básicos
  - char
  - int
  - float
  - double
- Modificadores
  - short
  - long
  - signed
  - unsigned
- Exemplo: unsigned long int x;
  unsigned long x;
- Default é signed

Caracteres como inteiros

```
/* Print the numeric code value of a character */

#include <stdio.h>

int main( void )
{
   char ch;

   printf( "Enter a character:" );
   scanf( "%c", &ch );
   printf( "Its numeric code value is: %d\n", ch );
   exit( 0 );
}
```


Printf em diferentes bases

```
/* Print the decimal and octal equivalents of a
 * hexadecimal constant.
 */

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

int main( void )
{
   int num;

   printf( "Enter a hexadecimal constant: ");
   scanf( "%x", &num);
   printf( "The decimal equivalent of %x is: %d\n", num, num);
   printf( "The octal equivalent of %x is: %o\n", num, num );
   exit( 0 );
}
```


Constantes inteiras

- Octal
  - Começa com “0”
- Hexadecimal
  - Começa com “0x”
- Decimal sem sufixo
  - Int, long int, unsigned long int
- Octal ou hexadecimal sem sufixo
  - Int, unsigned int, long int, unsigned long int
- Sufixo “u” ou “U”
  - Unsigned int, unsigned long int
- Sufixo “l” ou “L”
  - Long int, unsigned long int

Sequências de escape

- \141 a
- \x61 a
- \a bell
- \b backspace
- \f form feed
- \n new line
- \r carriage return
- \t horizontal tab
- \v vertical tab
- \\ contra-barra
- \’ apóstrofe
- \” aspas
- \? ?

Constantes de ponto flutuante

- Tipos básicos
  - float
  - double
- Modificador
  - long (long double)
- Ver "**limits.h**"
- Constantes:
  - 3.5 .5 3e2 3.7e12
  - 3.5f .5f 3e2f 3.7e12f
  - 3.5L .5L 3e2L 3.7e12L

Necessidade de inicialização

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

int main( void )
{
   int x;
   printf( "The value of x is: %d\n", x );
   exit( 0 );
}
```


Misturando tipos

- Compilador faz conversões implícitas
- Conversões implícitas são perigosas
  - 3.0 + 1/2
- Conversões implícitas acontecem:
  - Em atribuições int j = 2.6;
  - Quando char ou short int aparece em expressão (int)
  - Quando unsigned char ou unsigned short (int se der, senão unsigned int)
  - Para realizar uma operação
  - Parâmetros de funções em alguns casos (mais adiante)

Conversões implícitas em expressões

- Operando "menor" é convertido para tipo do operando "maior"
- Operação é feita no tipo "maior"

- Long double
- Double
- Float
- Unsigned long int
- Long int
- Unsigned int
- Int

Misturando tipos de inteiros

```
#include <stdlib.h>

int main( void )
{
   char c = 5;
   short j = 6;
   int k = 7;

   k = c+j+8;
   exit( 0 );
}
```


Exemplo com double

```
/* Convert a float value from Fahrenheit to Celsius
 */

double fahrenheit_to_celsius ( double temp_fahrenheit )
{
   double temp_celsius;
   temp_celsius = (temp_fahrenheit - 32.0) * 100.0 / (212.0 - 32.0);
   return temp_celsius;
}
```

Exemplo com float

```
/* Given the radius, find the area of a circle. */

#define                 PI      3.14159         // Double, f float, L long
   double

float area_of_circle( float radius )
{
   float area;
   area = PI * radius * radius;
   return area;
}
```


Misturando inteiro com ponto flutuante

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

int main( void )
{
   long int j = 2147483600;
   float x;

   x = j;
   printf( "j is %d\nx is %10f\n", j, x );
   exit( 0 );
}
```


Misturando tipos com e sem sinal

- Signed e unsigned diz respeito à interpretação do número
  - Não ao tamanho da variável
- Na maioria das vezes funciona bem
- Existem alguns casos complicados
- 10u - 15 == 4294967291 (inteiro de 4 bytes)
- unsigned jj;
- int k;
- if(jj - k < 0)
- nunca_vai_executar();
- if((int) jj - k < 0)
- pode_executar_ou_nao();

enum

```
enum { red, blue, green, yellow } color;

enum { bright, medium, dark} intensity;


enum{ APPLE, ORANGE = 10, LEMON, GRAPE = -5, MELONS };

enum{ APPLE = 0, ORANGE = 10, LEMON = 11,
        GRAPE = -5, MELONS = -4 };
```


typedef e define

```
#define USHORT unsigned int

typedef unsigned int USHORT

USHORT          i;              // Neste caso dá na mesma

#define PTR2INT int *

typedef int *PTR2INT

PTR2INT         x1, x2;         // Agora é diferente

int *x1, x2;

int *x1, *x2;
```


Endereços de variáveis

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

int main( void )
{
   int j=1;

   printf( "The value of j is: %d\n", j );
   printf( "The address of j is: %p\n", &j );
   exit( 0 );
}
```

Conceito de pointer

```
#include <stdio.h>

int main( void )
{
    int j=1;
    int *pj;

    pj = &j;                /* Assign the address of j to pj */
    printf( "The value of j is: %d\n", j );
    printf( "The address of j is: %p\n", pj );
    exit( 0 );
}
```


Usando pointers

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

int main( void )
{
    char *p_ch;
    char ch1 = 'A', ch2;

    printf( "The address of p_ch is %p\n", &p_ch );

    p_ch = &ch1;
    printf( "The value stored at p_ch is %p\n", p_ch );
    printf( "The dereferenced value of p_ch is %c\n", *p_ch );
    ch2 = *p_ch;

    exit( 0 );
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- **Controle de fluxo**
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Comando if

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> /* Include file for sqrt() */

int main( void )
{
    double num;

    printf( "Enter a non negative number: " );

    /* The %lf conversion specifier indicates data of type double.   */
    scanf( "%lf", &num);

    if (num < 0)
        printf( "Input Error: Number is negative.\n" );
    else
        printf( "The square root is: %f\n", sqrt( num ));
    exit( 0 );
}
```


Uso do else

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

int main( void )
{
    double num;

    printf( "Enter a non negative number: " );
    scanf( "%lf", &num );
    if (num < 0)
        printf( "Input Error: Number is negative.\n" );

    /* Next statement is always executed. */
    printf( "The square root is: %f\n", sqrt( num) );

    exit( 0 );
}
```


Identação

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

int main( void )
{ double num;
  printf("Enter a non negative number:"
  ) ;scanf("%lf", &num);
  if (num <
  0) printf ("input Error: Number is negative.\n");
  else printf ("The square root is: %f\n",
  sqrt(num)); exit(0);
}
```

Ponto e virgula mal colocado

```
if ( j == 1 );
    j = 0;


if ( j == 1 )
    ;       /* null statement */
    j = 0;
```


Operadores relacionais

- <
- >
- <=
- >=
- ==
- !=

- =

- Falso é representado pelo zero
- Verdadeiro é representado por diferente de zero

Função como expressão condicional

```
#include <stdio.h>
#include <ctype.h>          /* included for isalpha() */
#include <stdlib.h>

int main( void )
{
    char ch;

    printf( "Enter a character: ");
    scanf( "%c", &ch );
    if (isalpha ( ch ))
        printf( "%c", ch );
    else
        printf( "%c is not an alphabetic character.\n", ch );
    exit( 0 );
}
```


Comandos compostos

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

int main( void )
{
    double num;

    printf( "Enter a non negative number: " );
    scanf( "%lf", &num);
    if (num < 0)
        printf( "That's not a non negative number !\n" );
    else
    {
        printf( "%f squared is: %f\n", num, num*num );
        printf( "%f cubed is: %f\n", num, num*num*num );
    }
     exit( 0 );
}
```


If aninhado

```
int min( int a, int b, int c)
{
if( a < b )
    if( a < c )
        return a;
    else
        return c;
else    if( b < c )
        return b;
    else
        return c;
}
```

- Cuidado com os elses pendurados

switch

```
int switch_example( char input_arg )
{
  switch ( input_arg )
  {
    case 'A': return 1;
    case 'B': return 2;
    case 'C': return 3;
    case 'D': return 4;
    default : return  1;
  }
}
```

Equivalente ao switch

```
int switch_example( char input_arg )
{
  if (input_arg == 'A')
    return 1;
  else if (input_arg == 'B')
    return 2;
  else if (input_arg == 'C')
    return 3;
  else if (input_arg == 'D')
    return 4;
  else
    return  1;
}
```


Switch com vários comandos

```
/* Print error message based on error_code.  */
#define ERR_INPUT_VAL 1
#define ERR_OPERAND 2
#define ERR_OPERATOR 3
#define ERR_TYPE 4

void print_error( int error_code )
{
  switch (error_code)
  {
      case ERR_INPUT_VAL:
              printf("Error: Illegal input value.\n");          break;
      case ERR_OPERAND:
              printf("Error: Illegal operand.\n");              break;
      case ERR_OPERATOR:
              printf("Error: Unknown operator.\n");             break;
      case ERR_TYPE:
              printf("Error: Incompatible data.\n");            break;
      default:
              printf("Error: Unknown error code %d\n", error_code);          break;
  }
}
```


Switch: um comando para vários labels

```
/* Return 1 if argument is a punctuation character.
 * Otherwise, return zero.
 */

int is_punc( char arg )
{
    switch (arg)
    {
            case '.':
            case ',':
            case ':':
            case ';':
            case '!': return 1;
            default : return 0;
  }
}
```


Switch: outro exemplo

```
/* This function evaluates an expression, given
 * the two operands and the operator.
 */
#include <stdlib.h>
#include "err.h"                    /* contains the typedef declaration of ERR_CODE. */

double evaluate( double op1, double op2, char operator )
{
    switch (operator)
    {
            case '+':   return op1 + op2;
            case '-':   return op1 - op2;
            case '*':   return op1 * op2;
            case '/':   return op1 / op2;
            default :   /* Illegal operator */
                        print_error( ERR_OPERATOR );
   exit( 1 );
  }
}
```


while

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

int main( void )
{
  int ch, num_of_spaces = 0;

  printf( "Enter a sentence:\n" );
  ch = getchar();
  while ( ch != '\n' )
  {
    if (ch == ' ')
            num_of_spaces++;
    ch = getchar();
  }
  printf( "The number of spaces is %d.\n", num_of_spaces );
  exit( 0 );
}
```


do while

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

int main( void )
{
  int ch, num_of_spaces = 0;

  printf( "Enter a sentence:\n" );
  do
  {
    ch = getchar();
    if (ch == ' ')
            num_of_spaces++;
  } while (ch != '\n');

  printf( "The number of spaces is %d.\n", num_of_spaces );
  exit( 0 );
}
```

for 1/2

```
long int factorial( long val )
{
  int j, fact = 1;

  for ( j=2; j <= val; j++)
          fact = fact * j;
  return fact;
}
```


for 2/2

```
/* This function reads a string of digits from the
 * terminal and produces the string's integer value.
 */

#include <stdio.h>
#include <ctype.h>

int make_int(void )
{
  int num=0, digit;

  digit = getchar();
  for ( ; isdigit( digit ); digit = getchar() )
  {
     num = num * 10;
     num = num + (digit - '0');
  }
  return num;
}
```


Equivalente com while

```
#include <stdio.h>
#include <ctype.h>

int make_int(void)
{
  int num=0, digit;

  digit = getchar();
  while (isdigit( digit ))
  {
     num = num * 10;
     num = num + digit - '0';
     digit = getchar();
  }
  return num;
}
```


Atribuição no teste

```
#include <stdio.h>
#include <ctype.h>

int make_int(void)
{
  int num=0, digit;

  while (isdigit( digit = getchar() ))
  {
    num = num * 10;
    num = num + (digit - '0');
  }
  return num;
}
```


for sem inicialização

```
#include <stdio.h>

void pr_newline( int newline_num )
{
  for (; newline_num > 0; newline_num  )
    printf( "\n" );
}
```


for sem comando

```
#include <stdio.h>
#include <ctype.h> /* Header file for isspace(). */

void skip_spaces(void)
{
  int c;
  for (c = getchar(); isspace( c ); c = getchar())
          ; /* Null Statement */
  ungetc( c, stdin ); /* Put the nonspace character back in the buffer */
}
```

for sem comando – colocação do ponto e virgula

```
#include <stdio.h>
#include <ctype.h>

void skip_spaces(void)
{
    int c;

    for (c = getchar(); isspace( c ); c = getchar());
    ungetc( c, stdin );
}
```


Equivalente com o while

```
#include <stdio.h>
#include <ctype.h>

void skip_spaces (void)
{
  char c;

  while (isspace( c = getchar() ))
      ; /* Null Statement */
  ungetc( c, stdin );
}
```


Laços aninhados

```
/* print a multiplication table using nested loops */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
  int j, k;
  printf("    1  2  3  4  5  6  7  8  9  10\n");
  printf("                                \n");
  for (j = 1; j <= 10; j++)       /* outer loop */
  {
     printf( "%5dl", j );
     for (k=1; k <= 10; k++)    /* inner loop */
           printf( "%5%d", j*k );
    printf( "\n" );
  }
  exit( 0 );
}
```


Tudo junto

```
#include <stdio.h>
#include <ctype.h>
#define DECIMAL_POINT '.'

double parse_num()
{
  int c, j, digit_count = 0;
  double value = 0, fractional_digit;

  while (isdigit( c = getchar()) )
  {
    value = value * 10;
    value = value + (c - '0');
  }
  /* If c is not digit, see if there's decimal point
     */
  if (c == DECIMAL_POINT) /* get fraction */
    while (isdigit( c = getchar() ))
    {
      digit_count++;
      fractional_digit = c - '0';
      for (j=0; j < digit_count; j++)
            fractional_digit = fractional_digit/10;
      value = value + fractional_digit;
    }
  ungetc( c, stdin );
  return value;
}
```


Exemplo: uma calculadora simples

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

int main( void )
{
  extern double parse_num(), evaluate();
  extern void skip_spaces();
  double op1, op2, answer;
  int operator;

  printf( "Enter <number> <op> <number><newline>: ");
  skip_spaces();
  op1 = parse_num();
  skip_spaces();
  operator = getchar();
  skip_spaces();
  op2 = parse_num();
  answer = evaluate( op1, operator, op2 );
  printf( "%f\n", answer );
  exit( 0 );
}
```


Calculadora simples usando scanf

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

int main( void )
{
  double op1, op2, answer, evaluate();
  char operator;

  printf( "Enter <number> <op> <number><newline>: ");
  scanf( "%lf %c %lf", &op1, &operator, &op2 );
  answer = evaluate( op1, operator, op2 );
  printf( "%f\n", answer );
  exit( 0 );
}
```

Calculadora simples usando scanf e printf

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

int main( void )
{
 double op1, op2, evaluate();
 char operator;

 printf( "Enter <number> <op> <number><newline>: ");
 scanf( "%lf %c %lf", &op1, &operator, &op2 );
 printf( "%f\n", evaluate( op1, operator, op2 ) );
 exit( 0 );
}
```


break e continue

```
#include <stdio.h>
#include <ctype.h>

int mod_make_int()
{
 int num = 0, digit;
 while ((digit = getchar()) != '\n')
 {
   if (isdigit( digit ) == 0)
     continue;
   num = num * 10;
   num = num + (digit - '0');
 }
 return num;
}
```


goto

```
#include <stdio.h>
#include <math.h> /* for sqrt() function def */
#include <stdlib.h>

int main( void )
{
 int num;

 scanf( "%d", &num );
 if (num < 0)
     goto bad_val;
 else
 {
     printf( "The square root of num is %f", sqrt( num ) );
     goto end;
 }

bad_val:
     printf( "Error: Negative Value.\n" );
     exit( 1 );
end:
     exit( 0 );
}
```


Laço infinito

```
#include <ctype.h>
#include <stdio.h>

int main( void )
{
 double op1, op2, answer, evaluate();
 char operator;

 while (1)                                   /* for(;;) */
 {
   printf( "Enter <number> <op> <number>"
     "<newline>: ");
   scanf( "%lf %c %lf", &op1, &operator, &op2 );
   answer = evaluate( op1, operator, op2 );
   printf( "%f\n", answer );
 }
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- **Operadores e expressões**
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Precedência entre operadores

() [] -> .	esquerda-para-direita
operadores unários - ++ -- ! & * ~ (type) sizeof	direita-para-esquerda
* / %	esquerda-para-direita
+ -	esquerda-para-direita
<< >>	esquerda-para-direita
< <= >= >	esquerda-para-direita
== !=	esquerda-para-direita
&	esquerda-para-direita
^	esquerda-para-direita
|	esquerda-para-direita
&&	esquerda-para-direita
||	esquerda-para-direita
? :	direita-para-esquerda
= op=	direita-para-esquerda
,	esquerda-para-direita

Resto da divisão (módulo)

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

int main( void )
{
  int c, j = 0;

  printf( "Enter string to be squished: ");
  while ((c = getchar()) != '\n')
  {
     if ( j % 5 == 0)             /* j divisivel por 5 ? */
           printf( "\n" );
     putchar( c );
     j++;
  }
  exit( 0 );
}
```


Versão melhorada

```
#include <stdio.h>

void break_line ( int interval )
{
  int c, j = 1;

  while ((c = getchar()) != '\n')
  {
     putchar( c );
     if ( j % interval == 0 )
           printf( "\n" );
     j++;
  }
}
```


Pós-incremento

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

int main( void )
{
  int j = 5, k = 5;
  printf( "j: %d\t k: %d\n", j++, k );
  printf( "j: %d\t k: %d\n", j, k );
  exit( 0 );
}
```


Pré-incremento

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

int main( void )
{
  int j = 5, k = 5;
  printf( "j: %d\t k: %d\n", ++j, k );
  printf( "j: %d\t k: %d\n", j, k );
  exit( 0 );
}
```


Efeitos colaterais

- x = j * j++;

- x = j * j;
- ++j

- f(a, a++);

- f(a, a);
- ++a;

Diferença entre pré e pós-incremento

```
#include <stdio.h>

void break_line( int interval )
{
  int c, j=0;

  while ((c = getchar()) != '\n')
  {
    if ((j++ % interval) == 0)
          printf( "\n" );
    putchar( c );
  }
}
```

Operador virgula

```
#include <stdio.h>

void break_line( int interval)
{
  int c, j;

  for (c=getchar(), j = 0; c != EOF; j++, c = getchar())
  {
    putchar(c);
    if ( j % interval == 0)
          putchar( '\n' );
  }
}
```


Operadores bit-a-bit 1/4

```
/* Return bitmap of correct answers, limited to 32 total answers
 */
#include <stdio.h>

long get_answers()
{
     long answers = 0;
     int j;
     char c;

     for ( j=0; j <= 31; j++)
     {
          scanf( "%c", &c );
          if (c == 'y' || c == 'Y')
                answers |= 1L << j;
     }
     printf( "Answers entered = (%1x)", answers );
     return answers;
}
```


Operadores bit-a-bit 2/4

```
/* correct answers are:
 * nnyy ynyn nyyy yynn nnyn yyyy ynyy nyny
 * 0011 1010 0111 1100 0010 1111 1011 0101
 */

#define CORRECT_ANSWERS 0x3A7C2FB5

double grade_test( long int answers )
{
  extern int count_bits();
  long wrong_bits;
  double grade;

  wrong_bits = answers ^ CORRECT_ANSWERS;
  grade = 100 * ((32 - count_bits( wrong_bits )) / 32.0);
  return grade;
}
```


Operadores bit-a-bit 3/4

```
int count_bits( long bitCnt )
{
    int j, count = 0;

    for (j = 0; j <= 31; j++)
          if (bitCnt & (1L << j))
                ++count;
    return count;
}
```


Operadores bit-a-bit 4/4

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

int main( void )
{
  extern double grade_test();
  extern long int get_answers();
  double grade;

  printf( "Enter the answers: \n" );
  grade = grade_test( get_answers() );
  printf( "The grade is %3.0f\n", grade );
  exit( 0 );
}
```


sizeof

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

int main( void )
{
  printf( "TYPE\t\tSIZE\n\n" );
  printf( "char\t\t%d\n", sizeof (char) );
  printf( "short\t\t%d\n", sizeof (short) );
  printf( "int\t\t%d\n", sizeof (int) );
  printf( "float\t\t%d\n", sizeof (float) );
  printf( "double\t\t%d\n", sizeof (double) );
  exit( 0 );
}
```

Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- **Arrays**
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Arrays

```
#include <stdio.h>
#include <stdlib.h>
#define DAYS_IN_YEAR 365

int main( void )
{
  int j, sum=0;
  int daily_temp[DAYS_IN_YEAR];

  /* Assign values to daily_temp[] here. */

  for (j=0; j < DAYS_IN_YEAR; ++j)
     sum += daily_temp[j];
  printf( "The average temperature for the year is %d.\n",
                    sum/DAYS_IN_YEAR );
  exit( 0 );
}
```


Inicialização de array

```
/* Return a coded value for a character
*/
#define ILLEGAL_VAL  -1
char encode ( char ch)
{
static unsigned char encoder [128] = {  121, 124, 121,
118, 115, 112, 109, 106, 103, 100, 97, 94, 91, 88, 85,
82, 79, 76, 73, 70, 67, 64, 61, 58, 55, 52, 49, 46,
43, 40, 37, 34, 31, 28, 25, 22, 19, 16, 13, 10, 7, 4,
1, 126, 123, 120, 117, 114, 111, 108, 105, 102, 99,
96, 93, 90, 87, 84, 81, 78, 75, 72, 69, 66, 63, 60,
57, 54, 51, 48, 45, 42, 39, 36, 33, 30, 27, 24, 21,
18, 15, 12, 9, 6, 3, 125, 122, 119, 116, 113, 110,
107, 104, 101, 98, 95, 92, 89, 86, 83, 80, 77, 74, 71,
 68, 65, 62, 59, 56, 53, 50, 47, 44, 41, 38, 35, 32,
29, 26, 23, 20, 17, 14, 11, 8, 5, 2, 0
};

/* Test for illegal character. */
if (ch > 127)
     return ILLEGAL_VAL;
else
     return encoder[ch];              /* Return coded character.*/
}
```


Uso de array

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

int main( void )
{
  char c[5];
  int i;
  c[0] = encode ('W');
  c[1] = encode('h');
  c[2] = encode ('a');
  c[3] = encode ('t');
  c[4] = encode ('?');
  for (i=0; i< 5; ++i)
          printf( "%d\t", c[i] );
  exit( 0 );
}
```


Pointer para elemento de array 1/2

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

void clr( long *p )
{
    *p = 0;          /* Store a zero at location p. */
}

int main( void )
{
    static short s[3] = {1, 2, 3};
    clr( &s[1] );    /* Clear element 1 of s[] */
    printf( "s[0]=%d\ns[1]=%d\ns[2]=%d\n", s[0], s[1], s[2] );
    exit( 0 );
}
```


Pointer para elemento de array 2/2

```
short ar[4];
short *p;

p = &ar[0];

p = ar;
```

- Expressões possíveis
- *p ar[0]
- *(p+3) ar[3]
- *(p+i) ar[i]

- **Aritmética com ponteiros**

Array como parâmetro de função

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

int main( void )
{
  void print_size();
  float f_array[10];

  printf( "The size of f_array is: %d\n", sizeof(f_array) );
  print_size( f_array );
  exit( 0 );
}

void print_size( float arg [] )          /*      (float *arg)      */
{
    printf("The size of arg is: %d\n", sizeof(arg) );
}
```


Array multidimencional 1/6

- int x[3][5];
- x é um array de 3 elementos, cada um deles é um array de 5 inteiros.
- char x[3][4][5];
- x é um array de 3 elementos, cada um deles é um array de 4 elementos, sendo que cada um destes é um array de 5 caracteres.

Array multidimencional 2/6

```
int ar[2][3] = {
        { 0, 1, 2 },
        { 3, 4, 5 }
        };
```

ar [0][0]	0
ar [0][1]	1
ar [0][2]	2
ar [1][0]	3
ar [1][1]	4
ar [1][2]	5

Array multidimencional 3/6

ar[1][2]

*(ar[1] + 2)

*(*(ar+1) + 2)

ar[1] &ar[1][0]

Array multidimencional 4/6

```
int exemplo[] [3] [2] = {
                {   { 000, 001},
                    { 010, 011},
                    { 020, 021} },

                {   { 100, 101},
                    { 110, 111},
                    { 120, 121} }
                }
```


Array multidimencional 5/6

```
void f1( void )
{
   int ar[5][6][7];

   f2(ar);
}

void f2( int argumento[][6][7])
{
...
}
```

Array multidimencional 6/6

```
typedef enum {T_SPECIAL = -2, T_ILLEGAL, T_INT, T_FLOAT,
                T_DOUBLE, T_POINTER, T_LAST} TYPE;

TYPE type_needed( TYPE type1, TYPE type2)
{
static TYPE result_type[T_LAST][T_LAST] = {
/*        int      float   double   pointer */
/* int */  T_INT,    T_FLOAT,  T_DOUBLE, T_POINTER,
/*float */ T_FLOAT,  T_DOUBLE, T_DOUBLE, T_ILLEGAL,
/*double */T_DOUBLE, T_DOUBLE, T_DOUBLE, T_ILLEGAL,
/*pointer*/T_POINTER,T_ILLEGAL,T_ILLEGAL,T_SPECIAL
};

TYPE result = result_type[type1][type2];

if (result == T_ILLEGAL)
     printf( "Illegal pointer operation.\n" );
return result;
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- **Strings**
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Strings 1/2

- char str[] = "some text";
- char str[10] = "yes";
- char str[4] = "four";
- char *ptr = "more text";

Strings 2/2

```
#include <stdlib.h>

int main( void )
{
  char array[10];
  char *ptr1 = "10 spaces";
  char *ptr2;
  array = "not OK";          /* cannot assign to an address */
  array[5] = 'A';            /* OK */
  ptr1[5] = 'B';             /* OK */
  ptr1 = "OK";
  ptr1[5] = 'C';             /* "legal" but runtime error due to prior */
  /* assignment */
  *ptr2 = "not OK";          /* type mismatch */
  ptr2 = "OK";
  exit( 0 );
}
```


Ler e escrever string

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

#define MAX_CHAR 80

int main( void )
{
  char str [MAX_CHAR];
  int i;
  printf( " Enter a string: ");
  scanf( "%s", str );
  for (i = 0; i < 10; ++i)
     printf( "%s\n", str );
  exit( 0 );
}
```


strlen 1/3

```
int strlen( char str[] )
{
  int i=0;
  while (str[i])
          ++i;
  return i;
}
```

strlen 2/3

```
int strlen( char str[] )
{
  int i;
  for (i=0; str[i]; ++i)
          ;          /* null statement in for body */
  return i;
}
```


strlen 3/3

```
int strlen( char *str )
{
  int i;
  for (i = 0; *str++; i++)
          ; /* null statement */
  return i;
}
```


strcpy 1/3

```
/* Copy s2 to s1 */
void strcpy( char s1[], char s2[])
{
  int i;

  for (i=0; s2[i]; ++i)
          s1[i] = s2[i];
  s1[++i] = 0;
}
```


strcpy 2/3

```
void strcpy( char *s1, char *s2)
{
  int i;

  for (i=0; *(s2+i); ++i)
          *(s1+i) = *(s2+i);
  s1[++i] = 0;
}
```


strcpy 3/3

```
void strcpy( char *s1, char *s2)
{
  while (*s1++ = *s2++)
          ; /* null statement */
}
```


Pesquisa por padrões 1/4

```
/* Return the position of str2 in str1; -1 if not found. */

int pat_match( char str1[], char str2[])
{
  int j, k;
  /* Test str1[j] with each character in str2[]
   * If equal, get next char in str1[]
   * Exit loop if we get to end of str1[], or if chars are equal
   */
  for (j=0; j < strlen(str1); ++j)
  {
      for (k=0; k < strlen(str2) && (str2[k] == str1[k+j]); k++);
     /* Check to see if loop ended because we arrived at
      * end of str2. If so, strings must be equal.
      */
     if (k == strlen( str2 ))
              return j;
  }
  return -1;
}
```

Pesquisa por padrões 2/4

```
/* Return the position of str2 in str1; -1 if not found.
 */
pat_match( char str1[], char str2[])
{
  int j, k;
  int length1 = strlen( str1 );
  int length2 = strlen( str2 );

  for (j=0; j < length1; ++j)
  {
     for (k=0; k < length2; k++)
           if (str2[k] != str1[k+j])
                 break;
     if (k == length2)
           return j;
  }
  return -1;
}
```


Pesquisa por padrões 3/4

```
/* Return the first occurrence of str2 in str1
 * using pointers instead of arrays; return -1
 * if no match is found.
 */
pat_match( char *str1, char *str2)
{
  char *p, *q, *substr;

  /* Iterate for each character position in str1 */
  for (substr = str1; *substr; substr++)
  {
       p = substr;
       q = str2;
       /* See if str2 matches at this char position */
       while (*q)
             if (*q++ != *p++)
                   goto no_match;
    /* Only arrive here if every char in str2
     * matched. Return the number of characters
     * between the original start of str1 and the
     * current character position by using pointer
     * subtraction.
     */
    return substr - str1;

/* Arrive here if while loop couldn't match str2.
     * Since this is the end of the for loop, the
     * increment part of the for will be executed
     * (substr++), followed by the check for
     * termination (*substr), followed by this loop
     * body. We have to use goto to get here because
     * we want to break out of the while loop and
     * continue the for loop at the same time. Note
     * that the semicolon is required after the label
     * so that the label prefixes a statement (albeit
     * a null one).
     */
no_match: ;
  }
  /* We arrive here if we have gone through every
   * character of str1 and did not find a match.
   */
  return -1;
}
```


Pesquisa por padrões 4/4

```
extern int pat_match( char[], char[]);

int main( void )
{
  char first_string[100] , pattern[100];
  int pos;

  printf( "Enter main string:" );
  gets( first_string );
  printf( "Enter substring to find: " );
  gets( pattern );
  pos = pat_match( first_string, pattern );
  if (pos == -1)
        printf( "The substring was not matched.\n" );
  else
     printf( "Substring found at position %d\n", pos );
  exit( 0 );
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- **Pointers**
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Array de pointers 1/2

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

void print_month( int m )
{
  static char *month[13] = { "Badmonth", "January",
                             "February","March", "April", "May", "June",
                             "July", "August", "September","October",
                             "November", "December" };

  if (m > 12)
  {
     printf( "Illegal month value. \n" );
     exit( 1 );
  }
  printf( "%s\n", month [m] );
}
```


Array de pointers 2/2

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

char *month_text( int m )
{
  static char *month[13] = {"Badmonth", "January",
                            "February","March", "April", "May", "June",
                            "July", "August", "September", "October",
                            "November", "December" };

  if (m > 12)
  {
     printf( "Illegal month value.\n" );
     exit( 1 );
  }
  return month [m];
}
```

Pointer para pointer 1/3

```
int **p;
int j;
j = **p;

int r = 5;
int *q = &r;
int **p = &q;

r = 10;
*q = 10;
**p = 10;
```


Pointer para pointer 2/3

```
#include <string.h>
#include "spell.h"

#define MAX_WORDS 50

/* Dictionary in alphabetic order
 * with NULL as last entry.  */
static char *dict[LANG_NUM][MAX_WORDS] =
      {
  { "aardvark", "abacus", "abash", "abbot",
    "abhor", "able", "abort", "about", NULL
  },
  { "abeille", "absence", "absurde", "accepter",
    "accident", "accord", "achat", "acheter",
    NULL}
  };
```

```
/* Return NULL pointer if str is found in dictionary.
 * Otherwise, return a pointer to the closest match
 */
char *check_spell( char *str, LANGUAGE_T language)
{
  int j, diff;
  /* Iterate over the words in the dictionary */
  for (j=0; dict[language] [j] != NULL; ++j)
  {
    diff = strcmp( str, dict[language][j] );
    /* Keep going if str is greater than dict entry */
    if (diff > 0)
      continue;
    if (diff == 0)
      return NULL; /* Match! */
    return dict[language][j]; /* No match, return closest */
  }
/* Return last word if str comes after last
* dictionary entry */
  return dict[language][j - 1];
}
```


Pointer para pointer 3/3

```
#include <string.h>
#include "spell.h"

#define MAX_WORDS 50

/* Dictionary in alphabetic order
* with NULL as last entry */
static char *dict[LANG_NUM] [MAX_WORDS] =
      {
  {"aardvark", "abacus", "abash", "abbot", "abhor",
    "able", "abort", "about", NULL
  },
  {"abeille", "absence", "absurde", "accepter",
    "accident" "accord", "achat", "acheter",
    NULL}
  };
```

```
/* Return NULL pointer if str is found in dictionary.
 * Otherwise, return a pointer to the closest match.
 * This time use pointers instead of array references.
 */
char *check_spell( char *str, LANGUAGE_T language)
{
  int diff;
  char **curWord;

  /* Iterate over dictionary entries */
  for (curWord = dict[language]; *curWord; curWord++)
  {
    diff = strcmp( str, *curWord);
    /* Keep going if str is greater than dict entry */
    if (diff > 0)              continue;
    if (diff == 0)             return NULL;    /* Match! */
    /* No match, return closest spelling */
    return *curWord;
  }
/* Return last word if str comes after last entry */
  return curWord[-1];
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- **Classes de armazenamento**
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Variáveis estáticas e automáticas 1/2

```
#include <stdio.h>

void increment(void)
{
  int j=1;
  static int k=1;

  j++;
  k++;
  printf( "j: %d\tk: %d\n", j, k );
}

int main( void )
{
  increment();
  increment();
  increment();
}
```


Variáveis estáticas e automáticas 2/2

```
#include <stdio.h>

void increment(void)
{
  int j;
  static int k;

  j++;
  k++;
  printf( "j: %d\tk: %d\n", j, k );
}

int main( void )
{
  increment();
  increment();
  increment();
}
```

Uso de variável estática

```
#define IMPAR 0
#define PAR 1

print_header( char *chap_title )
{
 static char page_type = IMPAR;

 if (page_type == IMPAR)
 {
    printf( "\t\t\t\t%s\n\n", chap_title );
    page_type = PAR;
 }
 else
 {
    printf( "%s\n\n", chap_title );
    page_type = IMPAR;
 }
}
```


Escopo das variáveis 1/2

- Válida em todo o programa
- Válida somente em um arquivo
- Válida somente em uma função
- Válida somente em um bloco

```
int i;
static int j;
void func(void)
{
      int k;
      while(1)
      {
            int x;
```


Escopo das variáveis 2/2

```
#include <stdio.h>

int j=10;          /* Program scope */

int main( void )
{
 int j;  /* Block scope hides j at program scope */
 for (j=0; j < 5; ++j)
    printf( "j: %d", j );
}
```


register

```
int strlen( register char *p)
{
 register int len = 0;
 while (*p++)
        len++;
 return len;
}
```


const 1/2

- Não permite que a variável seja modificada
  - Mas alguns compiladores permitem acessar via pointers

- const char str[10] = “Constant”;
- str[0] = ‘a’; Ilegal !!!

- char *p = &str[0];
- *p = ‘m’; Ilegal ???

const 2/2

```
const long double pi = 3.1415926535897932385;

int *const const_ptr;          /* O pointer é constante */

int const *ptr_to_const;       /* O apontado é constante */


char *strcpy( char *p, const char *q)
{
...
}
```

Volatile – tentativa inicial

- Informa ao compilador que uma variável pode ser alterada por meios fora do programa
  - Variável mapeada para registrador de controlador

```
void get_two_kbd_chars (void)
{
  extern char KEYBOARD;
  char c0, c1;

  c0 = KEYBOARD;
  c1 = KEYBOARD;
}
```


Volatile – como o compilador entendeu

```
void get_two_kbd_chars(void)
{
  extern char KEYBOARD;
  char c0, c1;
  register char temp;

  temp = KEYBOARD;
  c0 = temp;
  c1 = temp;
}
```


Volatile usado

```
void get_two_kbd_chars (void)
{
  extern volatile char KEYBOARD;
  char c0, c1;

  c0 = KEYBOARD;
  c1 = KEYBOARD;
}
```


Volatile – outro exemplo

```
void read_ten_chars(void)
{
  extern char KEYBOARD;
  extern void copy(int);
  int x;
  char c;

  for (x=0; x < 10; x++)
  {
    c = KEYBOARD;
    copy( c );
  }
}
```


Volatile – como o compilador entendeu

```
void read_ten_chars( void )
{
  extern char KEYBOARD;
  extern void copy(int);
  int x;
  char c;

  c = KEYBOARD; /* The invariant expression is
                   removed from the loop. */
  for (x=0; x < 10; x++)
        copy(c);
}
```


Volatile - atribuição

- Permite impedir a otimização do código no caso de atribuições
- c = (* **(volatile char *)** 0x20);

Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- **Alocação de memória**
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Alocação dinâmica de memória 1/3

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

int main( void )
{
 extern void bubble_sort();
 int *list, sort_num, j;

 printf("How many numbers are you going to enter?");
 scanf("%d", &sort_num);
 list = (int *) malloc (sort_num * sizeof(int) );
 for( j=0; j < sort_num; j++)
        scanf( "%d", list + j );
 bubble_sort( list, sort_num );
 exit( 0 );
}
```


Alocação dinâmica de memória 2/3

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

int main( void )
{
 extern void bubble_sort();
 int *list, sort_num, j;

 printf("How many numbers are you going to enter?");
 scanf("%d", &sort_num);
 list = (int *) calloc (sort_num, sizeof(int) );
 for( j=0; j < sort_num; j++)
        scanf( "%d", list + j );
 bubble_sort( list, sort_num );
 exit( 0 );
}
```


Alocação dinâmica de memória 3/3

- malloc
- calloc

- Realloc – Muda o tamanho de uma área previamente alocada
- Free – Libera uma área previamente alocada

Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- **Estruturas e uniões**
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

structure

```
struct vitalstat
{
      char vs_name[19], vs_ssnum[11];
      short vs_month, vs_day, vs_year;
      };

struct vitalstat vs;

struct vitalstat vsa[1000], *pvs;

pvs = &vsa[10];
```

structure com typedef

```
struct vitalstat
{
    char vs_name[19], vs_ssnum[11];
    short vs_month, vs_day, vs_year;
    } vs, *pvs, vsa[10];

typedef struct
{
    char vs_name[19], vs_ssnum[11];
    short vs_month, vs_day, vs_year;
    } VITALSTAT;

VITALSTAT vs;
```


Acessando structures

```
VITALSTAT vs = { “George Smith”, “002340671”, 3, 5, 1946 };

vs.vs_month = 3;
vs.vs_day = 15;
vs.vs_year = 1987;

if( vs.vs_month > 12 || vs.vs_day > 31 )
    printf(“Illegal date.\n”);
```


Acessando via pointer para structure

```
VITALSTAT vs = { “George Smith”, “002340671”, 3, 5, 1946 };
VITALSTAT *pvs;
pvs = &vs;

pvs->vs_month = 3;
pvs->vs_day = 15;
pvs->vs_year = 1987;

if( pvs->vs_month > 12 || pvs->vs_day > 31 )
    Printf(“Illegal date.\n”);

(*pvs).vs_day
vs2 = vs;
```


Array de structures 1/3

```
#include "v_stat.h"          /* Contains declaration of VITALSTAT */

int agecount ( VITALSTAT vsa[], int size, int low_age,
                    int high_age, int current_year)
{
 int i, age, count = 0;


 for (i = 0; i < size; ++i)
 {
  age = current_year - vsa[i].vs_year;
  if (age >= low_age && age <= high_age)
        count++;
 }
 return count;
}
```


Array de structures 2/3

```
#include "v_stat.h"          /* Contains declaration of VITALSTAT */

int agecount ( VITALSTAT vsa[], int size, int low_age,
                    int high_age, int current_year)
{
 int i, age, count = 0;


 for (i = 0; i < size; ++vsa, ++i)
 {
  age = current_year - vsa->vs_year;
  if (age >= low_age && age <= high_age)
        count++;
 }
 return count;
}
```


Array de structures 3/3

```
#include "v_stat.h"          /* Contains declaration of VITALSTAT */

int agecount ( VITALSTAT vsa[], int size, int low_age,
                    int high_age, int current_year)
{
 int age, count = 0;
 VITALSTAT *p, *p_last = &vsa[size];

 for ( p = vsa; p < p_last; ++p)
 {
  age = current_year - p->vs_year;
  if (age >= low_age && age <= high_age)
        count++;
 }
 return count;
}
```

Structures aninhadas 1/2

```
typedef struct
{
    char vs_name[19], vs_ssnum[11];
    struct
    {
        char vs_month;
        char vs_day;
        short vs_year;
    } vs_birth_date;
} VITALSTAT;

vs.vs_birth_date.vs_year = 2000;
```


Structures aninhadas 2/2

```
typedef struct
{
    char month;
    char day;
    short year;
} DATE;


typedef struct
{
    char vs_name[19], vs_ssnum[11];
    DATE vs_birth_date;
} VITALSTAT;
```


Aninhamento dos membros

```
struct align_example
{
    char mem1;
    short mem2;
    char mem3;
    } s1;
```

- 1 byte + 2 bytes + 1 byte
- 1 byte + 1 gap + 2 bytes + 1 byte + 1 gap

offsetof

```
#include <stddef.h>

typedef struct
{
    char widgetName[MAX_NAME];
    int widgetCount;
    enum WIDGET_TYPE widgetType;
} WIDGET_INFO;

size_t typeOffset = offsetof( WIDGET_INFO, widgetType);
```


Bit fields

```
struct
{
    int a:3;
    int b:7;
    int c:2;
} s;
```

- Tamanho máximo é o de um inteiro normal
- Arranjo interno dos bits pode variar
- Um campo não pode passar fronteira de int (10 + 10 = 32)

Structure como parâmetro de função

- VITALSTAT st;
- ...
- func1(vs);
- ...
- func2(&vs);
- ...
- void func1(VITALSTAT v) {
- ...
- void func2(VITALSTAT *pv) {
- ...

- **Structures são diferentes de arrays (nome)**

Função que retorna structure

```
#include <stdio.h>
#include <math.h>      /* include file for trig functions */
#define TOO_LARGE 1e6          /* Differs from one machine to another. */
typedef struct {
  double sine, cosine, tangent;
} TRIG;

TRIG *get_trigvals( double radian_val )
{
  static TRIG result;
  /* If radian_val is too large, the sine, cosine and tangent values will be meaningless. */
  if (radian_val > TOO_LARGE)
  {
            printf( "Input value too large -- cannot return meaningful results\n" );
            return NULL;
  }
  result.sine = sin( radian_val );
  result.cosine = cos( radian_val );
  result.tangent = tan( radian_val );
  return &result;
}
```


union 1/4

```
typedef union
{
    struct
    {
        char c1, c2;
    } s;
    long j;
    float x;
} U;

U example;
example.s.c1 = 'a';
example.s.c2 = 'b';
example.j = 5;
```


union 2/4

```
union doub
{
  char c[8];
  double val;
};

double get_double(void)
{
  extern char get_byte();
  int j;
  union doub d;

  for (j=0; j < 8; j++)
     d.c[j] = get_byte();
  return d.val;
}
```


union 3/4

```
#include <stdio.h>
int main( void )
{
    union {
            long long_element;
            float float_element;
    } u;

    long lng_var;
    float flt_var;

    lng_var = u.long_element = 10;
    printf( "The value of lng_var cast to a float is: %f\n", (float) lng_var );
    printf( "The value of float element after\n"
                    "assignment to long_element is: %f\n\n", u.float_element );

    flt_var = u.float_element = 3.555;

    printf( "The value of flt_var cast to a long is: %d\n", (long) flt_var );
    printf( "The value of long_element after an \n"
              "assignment to float_element is: %d\n", u.long_element );
}
```


union 4/4

```
struct vitalstat
{
    struct vitalstat *next;
    char name[19], ssnum[11];
    unsigned int vs_day : 5,
    vs_month : 4,
    vs_year : 11;
    unsigned UScitizen : 1;
    union {
        char nationality[20];
        char city_of_birth[20];
        } location;
};
```


Union – exemplo 1/3

```
#include <stdio.h>
#include <string.h>
#include "v_stat2.h" /* includes location union */
#define TRUE 1
#define FALSE 0

static int is_yes(void)
{
 char answer[64];
 while (1)
 {
    fgets( answer, sizeof (answer), stdin);
    switch (answer[0])
    {
            case 'y':
            case 'Y': return TRUE;
            case 'n':
            case 'N': return FALSE;
            default : printf( "Please answer Y or N\n");
    }
 }
}
```

Union – exemplo 2/3

```
/* Remove trailing newline (if any), and see if user typed the right entry.
 */
static int double_check( char *s )
{
  int last_char = strlen( s ) - 1;
  if (s[last_char] == '\n')
      s[last_char] = '\0';
  printf( "Is '%s' correct? (Y or N) ", s);
  return is_yes();
}
```


Union – exemplo 3/3

```
void get_city_info( VITALSTAT *pvs)
{
  int answered = FALSE;

  printf("Are you a U.S. citizen? ");
  pvs->UScitizen = is_yes();
  while (!answered)
      if (!pvs->UScitizen)
      {
              printf("What is your nationality?");
              fgets(pvs->location.nationality, sizeof(pvs->location.nationality), stdin);
              answered = double_check(pvs->location.nationality);
      }
      else /* UScitizen */
      {
              printf("Enter city of birth: ");
              fgets(pvs->location.city_of_birth, sizeof(pvs->location.city_of_birth), stdin);
              answered = double_check(pvs->location.city_of_birth);
      }
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- **Funções**
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- C99

Passagem de parâmetros 1/3

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

int main( void )
{
  extern void f(int);
  int a = 2;

  f( a );                    /* pass a copy of "a" to "f()" */
  printf( "%d\n", a);
  exit(0);
}

void f( int received_arg )
{
  received_arg = 3;          /* Assign 3 to argument copy */
}
```


Passagem de parâmetros 2/3

```
/* Swap the values of two int variables */

void swap( int *x, int *y)
{
  register int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
```


Passagem de parâmetros 3/3

```
int main( void )
{
  int a = 2, b = 3;
  swap ( &a, &b );
  printf( "a = %d\t b = %d\n", a, b );
}
```

Retorno da função

```
float f(void)
{
  float f2;
  int a;
  char c;

  f2 = a;         /* OK, quietly converts a to float */
  return a;       /* OK, quietly converts a to float */
  f2 = c;         /* OK, quietly converts c to float */
  return c;       /* OK, quietly converts c to float */
}
```


Função que retorna pointer

```
char *f(void)
{
  char **cpp, *cp1, *cp2, ca[10];
  int *ip1;

  cp1 = cp2;          /* OK, types match */
  return cp2;         /* OK, types match */
  cp1 = *cpp;         /* OK, types match */
  return *cpp;        /* OK, types match */

  /* An array name without a subscript gets converted
   * to a pointer to the first element */

  cp1 = ca;           /* OK, types match */
  return ca;          /* OK, types match */
  cp1 = *cp2;         /* Error, mismatched types (pointer to char vs. char) */
  return *cp2;        /* Error, mismatched types (pointer to char vs. char) */
  cp1 = ip1;          /* Error, mismatched pointer types */
  return ip1;         /* Error, mismatched pointer types */
  return;             /* Produces undefined behavior, should return (char *) */
}
```


Protótipos

- extern int prototyped_func(int, float);
- extern int prototyped_func(int tamanho, float media);
  - Mais fácil de entender
- Void usado quando não recebe ou retorna nada
- Extern tem escopo de bloco ou de arquivo
- Extern int printf(const char *format, ...);

Pointers para função 1/5

```
int (*pf) (void);

int *pi (void);

extern int f1(void);

pf = f1;
```

ERRADOS:

```
    pf = f1();
    pf = &f1();
    pf = &f1;

int x = (*pf)();
```


Pointers para função 2/5

```
extern int if1(), if2(), (*pif)();
extern float ff1(), (*pff)();
extern char cf1(), (*pcf)();

int main( void )
{
  pif = if1;      /* Legal    types match */
  pif = cf1;      /* ILEGAL  type mismatch */
  pff = if2;      /* ILEGAL  type mismatch */
  pcf = cf1;      /* Legal    types match */
  if1 = if2;      /* ILEGAL  Assign to a constant */
}
```


Pointers para função 3/5

```
/* Compare two integers and return 1 if a is
 * greater than b use for ascending sorts.
 */
int compare_ascend( int a, int b )
{
    return a > b;
}

/* Compare two integers and return 1 if a is less
 * than b use for descending sorts.
*/
int compare_descend( int a, int b)
{
    return a < b;
}
```

Pointers para função 4/5

```
#define FALSE 0
#define TRUE 1

void bubble_sort( int list [], int list_size, int (*compare)())
{
    int j, temp, sorted = FALSE;

    while (!sorted)
    {
        sorted = TRUE;          /* assume list is sorted */
        for (j = 0; j < list_size - 1; j++)
            if ((*compare)(list[j], list[j+1]))
            {
                temp = list[j];
                list[j] = list[j+1];
                list [j+1] = temp;
                sorted = FALSE;
            }
    } /* end of while loop */
}
```


Pointers para função 5/5

```
#include <stdlib.h>

#define ASCEND compare_ascend
#define DESCEND compare_descend

extern void bubble_sort(int[], int, int(int,int));
extern int compare_ascend(int,int), compare_descend(int,int);

int main( void )
{
 static int list[] = {1, 0, 5, 444,  332, 76 };

 #define LIST_SIZE (sizeof(list)/sizeof(list[0]))

 bubble_sort( list, LIST_SIZE, DESCEND );
 exit( 0 );
}
```


Retornando pointer para função 1/4

```
char (*f(int a, int b))(float  x, float y)
{
    ....
```

F é uma função que retorna um pointer para função

Retornando pointer para função 2/4

```
void (*best_sort(float list[])) ( float list[] )
{
    extern void quick_sort(), merge_sort(),

    heap_sort();
    /* Analyze data */
    /* If quick sort is best */
        return quick_sort;
    /* Else if merge sort is best */
        return merge_sort;
    /* Else if heap sort is best */
        return heap_sort;
}
```


Retornando pointer para função 3/4

```
void sort_array( float list[])
{
    extern void (* best_sort())();

    (best_sort( list )) ( list );
}
```


Retornando pointer para função 4/4

```
int (*f(void)) (void)          /* f is a function that returns a pointer to
                                * a function that returns an int.
                                */
{
    extern int f1(void);       /* f1 is a function that returns an int. */

    extern int (* f2(void))(void);      /* f2 is a function that returns a
                                         * pointer to a function that
                                         * returns an int. */

    int (*pf)(void);           /* pf is a pointer to a function
                                *  that returns an int      */

    pf = f1;        /* OK, types match. */
    return f1;      /* OK types match. */
    pf = f2;        /* Error, mismatched pointer types. */
    return f2;      /* Error, mismatched pointer btypes */
}
```

Recursão 1/3

```
#include <stdio.h>

void recurse(void)
{
    static count = 1;
    printf("%d\n", count);
    count++;
    recurse();
}

int main( void )
{
    extern void recurse();
    recurse();
}
```


Recursão 2/3

```
void recurse()
{
    static count = 1;
    if (count > 3)
        return;
    else
    {
        printf( "%d\n", count );
        count ++;
        recurse();
    }
}

int main( void )
{
    extern void recurse();
    recurse();
}
```


Recursão 3/3

```
int sum( int n )
{
    if (n <= 1)
        return n;
    else
        return n + sum(n - 1);
}
```


Função main 1/2

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

/* echo command line arguments */

int main( int argc, char *argv[])
{
  while(--argc > 0)
        printf( "%s ", *++argv);
  printf( "\n" );
  exit( 0 );
}
```


Função main 2/2

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

int main( int argc, char *argv[])
{
  float x, y;

  if (argc < 3)
  {
     printf( "Usage: power <number>\n" );
     printf( "Yields arg1 to arg2 power\n" );
     return;
  }

  x = atof( *++argv );
  y = atof( *++argv );
  printf( "%f\n", pow( x, y ) );
}
```


Declarações complexas 1/3

- **char *x[];**

- x[] é um array
- *x[] é um array de pointers
- char *x[] é um array de pointers para caracteres

Declarações complexas 2/3

- **int (*x[])(void);**

- x[] é um array
- (*x[]) é um array de pointers
- (*x[])(void) é um array de pointers para funções
- int (*x[])(void) é um array de pointers para funções que retornam int

- int *x[](void);
- É um array de funções que retornam pointer para int

Declarações complexas 3/3

- **(*x)**
  - x é um pointer
- **(*x)[]**
  - x é um pointer para um array
- **(*(*x)[])**
  - x é um pointer para um array de pointers
- **(*(*x)[])(void)**
  - x é um pointer para um array de pointers para funções
- **(*(*(*x)[])(void))**
  - x é um pointer para um array de pointers para funções que retornam pointers
- **(*(*(*x)[])(void))[]**
  - x é um pointer para um array de pointers para funções que retornam pointers para arrays
- **struct s (*(*(*x)[])(void))[]**
  - x é um pointer para um array de pointers para funções que retornam pointers para arrays de structures s

Linguagem de Programação C

macros 1/7

```
#define BUFFER_LENGTH       256
char buffer[BUFFER_LENGTH];

#define DOUBLE(X)           X+X
int j = 3 * DOUBLE(4);      /*      3 * 4 + 4       */

#define DOUBLE(X)           (X+X)
int j = 3 * DOUBLE(4);      /*      3 * (4 + 4)     */

#define SQUARE(x)           (x*x)
int j = SQUARE(2+4);        /*      (2+4*2+4)       */

#define SQUARE(x)           ( (x) * (x) )
int j = SQUARE(2+4);        /*      ((2+4)*(2+4))   */
```


macros 2/7

- Verificação de tipos

```
#define SQUARE(x)           ( (x) * (x) )


int square( int x )
{
  return x * x;
}
```


macros 3/7

- #define até o final do arquivo
- Ou até encontrar um #undef

- Múltiplos #defines sem #undef são aceitos se forem idênticos

macros 4/7

- #define **MIN(a,b)** ((a) < (b) ? (a) : (b))
- x = **MIN(y++, z)**;
- x = ((y++) < (z) ? (y++) : (z));

macros 5/7

- __LINE__
- __FILE__
- __TIME__
- __DATE__
- __STDC__

```
void print_version()
{
  printf( "This utility compiled on %s at %s\n",
          __DATE__, __TIME__);
}
```


macros 6/7

```
#define CHECK (a, b) \
if ((a) != (b)) \
fail (a, b, __FILE__, __LINE__)

void fail( int a, int b, char *file, int line )
{
  printf( "Check failed in file %s at line %d:\
          got %d, expected %d\n", file, line, a, b );
}
```


macros 7/7

- #define str(s) #s
- printf(str(This is a string));
- printf(“This is a string”);
- #define ASSERT(b) if(!(b)) \
  - { \
  - Printf(“The following condition failed: %s\n”, #b); \
  - Exit(1); \
  - }
- ASSERT(array_ptr < array_start + array_size);

Compilação condicional 1/4

```
#if      x==1
    #undef x
    #define x 0
#elif    x== 2
    #undef x
    #define x 3
#else
    #define y 4
#endif
```

- Expressão condicional precisa ser uma constante
- Outras macros na expressão são expandidas antes
- Constante não definida é tratada como zero

Compilação condicional 2/4

```
#if      DEBUG
if (exp_debug)
  {
    printf( "1hs = " );
    print_value( result );
    printf( " rhs = " );
    print_value( &rvalue );
    printf( "\n" );
  }
#endif
```

Compilação condicional 3/4

```
#if (__STDC__)
  extern int foo( char a, float b );
  extern char *goo( char *string );
#else
  extern int foo();
  extern char *goo();
#endif
```


Compilação condicional 4/4

```
#ifdef TEST
    Printf("This is a test.\n");
#else
    Printf("This is not a test.\n");
#endif

#ifdef  TEST
#if defined     TEST
#if defined     (TEST)

#if !defined    TEST
#ifndef         TEST
```


Include

- Uso típico:
- modulo.h:

```
extern int page_num;
```

- modulo.c:

```
int page_num = 1;
```

- outro.c:

```
#include "modulo.h"
if( page_num > 0 )
...
```


line

```
/* Useful for programs that produce C code */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
 printf( "Current line: %d\nFilename: %s\n\n",
        __LINE__, __FILE__);
#line 100
 printf( "Current line: %d\nFilename: %s\n\n",
        __LINE__ , __FILE__);
#line 200 "new name"
 printf( "Current line: %d\nFilename: %s\n\n",
        __LINE__, __FILE__);
 exit (0);
}
```


error

```
#if      INTSIZE < 16
#error  INTSIZE too small
#endif
```

- Testa aspectos da compilação

```
cc  –DINTSIZE=8   test.c
```


pragma

- Definido pelo compilador

Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- **Entrada e saída**
- Biblioteca padrão
- C99

Acesso sequencial 1/7

```
#include <stddef.h>
#include <stdio.h>

/* Returns a pointer to opened FILE */
/* r, w, a, r+, w+, a+ */

FILE *open_test()
{
 FILE *fp;

 fp = fopen ( "test", "r" );
 if (fp == NULL)
    fprintf( stderr, "Error opening file test\n" );
 return fp;
}
```


Acesso sequencial 2/7

```
/* Return stream status flags.
 * Two flags are possible: EOF and ERROR
 */

#include <stdio.h>
#define EOF_FLAG 1
#define ERR_FLAG 2

char stream_stat( FILE *fp )
{
    char stat = 0;
    if (ferror( fp ))
        stat |= ERR_FLAG;
    if (feof( fp ))
        stat |= EOF_FLAG;
    clearerr(fp);
    return stat;
}
```


Acesso sequencial 3/7

```
#include <stddef.h>
#include <stdio.h>

FILE *open_file( char *file_name, char *access_mode)
{
 FILE *fp;

 if ((fp = fopen( file_name, access_mode )) == NULL)
        fprintf( stderr, "Error opening file %s with access"
                "mode %s\n", file_name, access_mode );
 return fp;
}
```


Acesso sequencial 4/7

```
#include <stddef.h>
#include <stdio.h>
#define FAIL 0
#define SUCCESS 1

int copyfile( char *infile, char *outfile)
{
 FILE *fp1, *fp2;

 if ((fp1 = fopen( infile, "rb" )) == NULL)
     return FAIL;
 if ((fp2=fopen ( outfile, "wb" )) == NULL)
 {
     fclose( fp1 );
     return FAIL;
 }
 while (!feof( fp1))
     putc( getc( fp1), fp2 );
 fclose( fp1 );
 fclose( fp2 );
 return SUCCESS;
}
```


Acesso sequencial 5/7

```
#include <stddef.h>
#include <stdio.h>
#define FAIL 0
#define SUCCESS 1
#define LINESIZE 100

int copyfile( char *infile, char *outfile )
{
 FILE *fp1, *fp2;
 char line[LINESIZE];

 if ((fp1 = fopen( infile, "r" )) == NULL)      return FAIL;
 if ((fp2 = fopen( outfile, "w" )) == NULL)
 {   fclose( fp1 );
     return FAIL;
 }
 while ( fgets( line, LINESIZE - 1, fp1) != NULL )
     fputs( line, fp2);
 fclose( fp1 );
 fclose( fp2 );
 return SUCCESS;
}
```

Acesso sequencial 6/7

```
#include <stddef.h>
#include <stdio.h>

#define FAIL 0
#define SUCCESS 1
#define BLOCKSIZE 512

typedef char DATA;

int copyfile( char *infile, char *outfile)
{
 FILE *fp1, *fp2;
 DATA block[BLOCKSIZE];
 int num_read;

 if ((fp1 = fopen( infile, "rb")) == NULL)
 {
  printf( "Error opening file %s for input.\n",
          infile );
  return FAIL;
 }

 if ((fp2 = fopen( outfile, "wb" )) == NULL)
 {
  printf( "Error opening file %s for output.\n",
          outfile );
  fclose( fp1 );
  return FAIL;
 }

 while ( (num_read =
  fread( block, sizeof(DATA),BLOCKSIZE, fp1 )) > 0 )

        fwrite( block, sizeof(DATA), num_read, fp2 );

 fclose( fp1 );
 fclose( fp2 );
 if (ferror( fp1 ))
 {
    printf( "Error reading file %s\n", infile );
    return FAIL;
 }
 return SUCCESS;
}
```


Acesso sequencial 7/7

```
#include <stdio.h>
#include <stddef.h>
#define MAX_LINE_SIZE 120

int lines_in_file( FILE *fp )
{
 char buf[MAX_LINE_SIZE];
 int line_num = 0;

 rewind(fp);      /* Moves the file position indicator
                   * to the beginning of the file.
                   */
 while ( fgets( buf, MAX_LINE_SIZE, fp ) != NULL )
    line_num++;

 return line_num;
}
```


Acesso aleatório 1/5

```
#ifndef NULL
#define NULL 0
#endif

#define MAX_REC_NUM      100
#define NAME_LEN         32

typedef struct {
        char city_of_birth[23];
        char location[32];
        char nationality[32];
} LOCATION;


typedef struct element {
        int vs_year;
        char vs_name[NAME_LEN];
        struct element *next;
        char UScitizen;
        LOCATION location;
} ELEMENT;
```

```
typedef struct {
        int day;
        int month;
        int year;
} BDATE;


typedef struct {
        int vs_year;
        BDATE bdate;
        long ssnum;
        char name[NAME_LEN];
        struct element *next;
        char UScitizen;
        LOCATION location;
} VITALSTAT;

typedef struct {
        int index;
        char key[NAME_LEN];
        char name[NAME_LEN];
        ELEMENT *element;
} INDEX;
```


Acesso aleatório 2/5

```
/* Reads up to max_rec_num records from a file and stores the key field of each record in an index array
 * Returns the number of key fields stored.
 */

#include "stdio.h"
#include "recs.h"

int get_records( FILE *data_file, INDEX names_index[], int max_rec_num)
{
       int k, offset = 0, counter = 0;

      for (k = 0; !feof( data_file ) && counter < max_rec_num; k++)
      {
              fgets(names_index[k].key, NAME_LEN, data_file);
              offset += sizeof(VITALSTAT);
              if (fseek( data_file, offset, SEEK_SET ) && (!feof( data_file )))
                      exit( 1 );
              counter++;
  }
      return counter;
}
```


Acesso aleatório 3/5

```
/* Sort an array of NAMES_INDEX structures by the name field.
 * There are index_count elements to be sorted. Returns a pointer to the sorted array.
 */
#include <stdlib.h>     /* Header file for qsort() */
#include "recs.h"

INDEX* sort_index( INDEX names_index[], int index_count)
{
 int j;
 static int compare_func();          /* Defined later */

 /* Assign values to the index field of each structure. */
 for (j = 0; j < index_count; j++)
     names_index[j].index = j;
 qsort( names_index, index_count, sizeof(INDEX), compare_func );
 return names_index;
}

static int compare_func( INDEX *p, INDEX *q)
{
      return strcmp( p->name, q->name );
}
```


Acesso aleatório 4/5

```
/* Print the records in a file in the order indicated by the index array.
 */
#include <stdio.h>
#include "recs.h"

void print_indexed_records( FILE *data_file,INDEX index[], int index_count)
{
 VITALSTAT vs;
 int j;

 for (j = 0; j <= index_count; j++)
 {
  if (fseek( data_file, sizeof(VITALSTAT) * index[j].index, SEEK_SET ))
          exit( 1 );
  fread( &vs, 1, sizeof(VITALSTAT), data_file );
  printf("%20s, %hd, %hd, %hd, %12s",
                                  vs.name,vs.bdate.day,vs.bdate.month,vs.bdate.year,vs.ssnum);
 }
}
```

Acesso aleatório 5/5

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

int main( int argc, char *argv[] )
{
 extern int get_records();
 extern void sort_index();
 extern int print_indexed_records();

 FILE *data_file;
 static INDEX index[MAX_REC_NUM];
 char filename[32];
 int num_recs_read;

 if (argc != 2)
 {
    printf( "Error: must enter filename\n" );
    printf( "Filename: ");
    scanf( "%s", filename );
 }
 else       strcpy( filename, argv[1]);

 if ((data_file = fopen( filename, "r" )) == NULL)
 {
    printf( "Error opening file %s.\n", filename );
    exit( 1 );
 }
 num_recs_read = get_records( data_file, index,
    MAX_REC_NUM );
 sort_index( index, num_recs_read );
 print_indexed_records( data_file, index,
    num_recs_read );
 exit( 0 );
}
```


Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- **Biblioteca padrão**
- C99

Biblioteca padrão: assert.h ctype.h errno.h

- <assert.h>
- **void assert(int expression);**
- <ctype.h>
- int isalnum(int c);
- int isalpha(int c);
- int iscntrl(int c);
- int isdigit(int c);
- int isgraph(int c);
- **int islower(int c);**
- int isprint(int c);
- int ispunct(int c);
- int isspace(int c);
- int isupper(int c);
- int isxdigit(int c);
- int tolower(int c);
- int toupper(int c);
- <errno.h>
- **errno**
- EDOM
- ERANGE

Biblioteca padrão: float.h

- <float.h>
- FLT_RADIX
- FLT_ROUNDS
- FLT_DIG
- DBL_DIG
- LDBL_DIG
- FLT_EPSILON
- DBL_EPSILON
- LDBL_EPSILON
- FLT_MANT_DIG
- DBL_MANT_DIG
- LDBL_MANT_DIG
- FLT_MAX
- **DBL_MAX**
- LDBL_MAX
- FLT_MAX_EXP
- DBL_MAX_EXP
- LDBL_MAX_EXP
- FLT_MIN
- DBL_MIN
- LDBL_MIN
- FLT_MIN_EXP
- DBL_MIN_EXP
- LDBL_MIN_EXP

Biblioteca padrão: limits.h

- <limits.h>
- CHAR_BIT
- CHAR_MAX
- CHAR_MIN
- SCHAR_MAX
- SCHAR_MIN
- **UCHAR_MAX**
- SHRT_MAX
- SHRT_MIN
- USHRT_MAX
- INT_MAX
- INT_MIN
- UINT_MAX
- LONG_MAX
- LONG_MIN
- ULONG_MAX

Biblioteca padrão: locale.h

- <locale.h>
- struct lconv
  - char* decimal_point; char* grouping;
  - char* thousands_sep; **char* currency_symbol;**
  - char* int_curr_symbol; char* mon_decimal_point;
  - char* mon_grouping; char* mon_thousands_sep;
  - char* negative_sign; char* positive_sign;
  - char frac_digits; char int_frac_digits;
  - char n_cs_precedes; char n_sep_by_space;
  - char n_sign_posn; char p_cs_precedes;
  - char p_sep_by_space; char p_sign_posn;
- struct lconv* localeconv(void);
- char* setlocale(int category, const char* locale);
- LC_ALL
- LC_NUMERIC
- LC_MONETARY
- LC_COLLATE
- LC_CTYPE
- LC_TIME
- NULL

Biblioteca padrão: math.h

- <math.h>
- HUGE_VAL
- double exp(double x);
- double log(double x);
- double log10(double x);
- double pow(double x, double y);
- **double sqrt(double x);**
- double ceil(double x);
- double floor(double x);
- double fabs(double x);
- double ldexp(double x, int n);
- double frexp(double x, int* exp);
- double modf(double x, double* ip);
- double fmod(double x, double y);
- double sin(double x);
- double cos(double x);
- double tan(double x);
- double asin(double x);
- double acos(double x);
- double atan(double x);
- double atan2(double y, double x);
- double sinh(double x);
- double cosh(double x);
- double tanh(double x);

Biblioteca padrão: setjmp.h signal.h

- <setjmp.h>
- jmp_buf
- **int setjmp(jmp_buf env);**
- **void longjmp(jmp_buf env, int val);**
- <signal.h>
- SIGABRT
- SIGFPE
- SIGILL
- SIGINT
- SIGSEGV
- SIGTERM
- SIG_DFL
- SIG_ERR
- SIG_IGN
- **void (*signal(int sig, void (*handler)(int)))(int);**
- **int raise(int sig);**

Biblioteca padrão: stdarg.h stddef.h

- <stdarg.h>
- **va_list**
- void va_start(va_list ap, lastarg);
- type va_arg(va_list ap, type);
- void va_end(va_list ap);

- <stddef.h>
- NULL
- **offsetof(stype, m)**
- ptrdiff_t
- size_t

Biblioteca padrão: stdio.h 1/3

- <stdio.h>
- BUFSIZ
- EOF
- FILENAME_MAX
- FOPEN_MAX
- L_tmpnam
- NULL
- SEEK_CUR
- SEEK_END
- SEEK_SET
- TMP_MAX
- _IOFBF
- _IOLBF
- _IONBF
- stdin stdout stderr
- **FILE**
- fpos_t
- size_t
- FILE* fopen(const char* filename, const char* mode);
- FILE* freopen(const char* filename, const char* mode, FILE* stream);
- int fflush(FILE* stream);
- int fclose(FILE* stream);
- int remove(const char* filename);
- int rename(const char* oldname, const char* newname);

Biblioteca padrão: stdio.h 2/3

- <stdio.h>
- FILE* tmpfile();
- char* tmpnam(char s[L_tmpnam]);
- int setvbuf(FILE* stream, char* buf, int mode, size_t size);
- void setbuf(FILE* stream, char* buf);
- int fprintf(FILE* stream, const char* format, ...);
- **int printf(const char* format, ...);**
- int sprintf(char* s, const char* format, ...);
- int vfprintf(FILE* stream, const char* format, va_list arg);
- int vprintf(const char* format, va_list arg);
- int vsprintf(char* s, const char* format, va_list arg);
- int fscanf(FILE* stream, const char* format, ...);
- int scanf(const char* format, ...);
- int sscanf(char* s, const char* format, ...);

Biblioteca padrão: stdio.h 3/3

- <stdio.h>
- int fgetc(FILE* stream);
- char* fgets(char* s, int n, FILE* stream);
- int fputc(int c, FILE* stream);
- char* fputs(const char* s, FILE* stream);
- int getc(FILE* stream);
- **int getchar(void);**
- char* gets(char* s);
- int putc(int c, FILE* stream);
- int putchar(int c);
- int puts(const char* s);
- int ungetc(int c, FILE* stream);
- size_t fread(void* ptr, size_t size, size_t nobj, FILE* stream);
- size_t fwrite(const void* ptr, size_t size, size_t nobj, FILE* stream);
- int fseek(FILE* stream, long offset, int origin);
- long ftell(FILE* stream);
- void rewind(FILE* stream);
- int fgetpos(FILE* stream, fpos_t* ptr);
- int fsetpos(FILE* stream, const fpos_t* ptr);
- void clearerr(FILE* stream);
- int feof(FILE* stream);
- int ferror(FILE* stream);
- void perror(const char* s);

Biblioteca padrão: stdlib.h 1/2

- <stdlib.h>
- EXIT_FAILURE
- EXIT_SUCCESS
- RAND_MAX
- NULL
- div_t
- ldiv_t
- size_t
- int abs(int n);
- long labs(long n);
- div_t div(int num, int denom);
- ldiv_t ldiv(long num, long denom);
- **double atof(const char* s);**
- int atoi(const char* s);
- long atol(const char* s);
- double strtod(const char* s, char** endp);
- long strtol(const char* s, char** endp, int base);
- unsigned long strtoul(const char* s, char** endp, int base);

Biblioteca padrão: stdlib 2/2

- <stdlib.h>
- void* calloc(size_t nobj, size_t size);
- **void* malloc(size_t size);**
- void* realloc(void* p, size_t size);
- void free(void* p);
- void abort();
- void exit(int status);
- int atexit(void (*fcm)(void));
- int system(const char* s);
- char* getenv(const char* name);
- void* bsearch(const void* key, const void* base, size_t n, size_t size, int (*cmp)(const void* keyval, const void* datum));
- void qsort(void* base, size_t n, size_t size, int (*cmp)(const void*, const void*));
- int rand(void);
- void srand(unsigned int seed);

Biblioteca padrão: string.h

- <string.h>
- NULL
- size_t
- char* strcpy(char* s, const char* ct);
- char* strncpy(char* s, const char* ct, size_t n);
- char* strcat(char* s, const char* ct);
- char* strncat(char* s, const char* ct, size_t n);
- int strcmp(const char* cs, const char* ct);
- int strncmp(const char* cs, const char* ct, size_t n);
- int strcoll(const char* cs, const char* ct);
- char* strchr(const char* cs, int c);
- char* strrchr(const char* cs, int c);
- size_t strspn(const char* cs, const char* ct);
- size_t strcspn(const char* cs, const char* ct);
- char* strpbrk(const char* cs, const char* ct);
- char* strstr(const char* cs, const char* ct);
- **size_t strlen(const char* cs);**
- char* strerror(int n);
- char* strtok(char* s, const char* t);
- size_t strxfrm(char* s, const char* ct, size_t n);
- void* memcpy(void* s, const void* ct, size_t n);
- void* memmove(void* s, const void* ct, size_t n);
- int memcmp(const void* cs, const void* ct, size_t n);
- void* memchr(const void* cs, int c, size_t n);
- void* memset(void* s, int c, size_t n);

Biblioteca padrão: time.h

- <time.h>
- **CLOCKS_PER_SEC**
- NULL
- clock_t
- time_t
- struct tm
- **clock_t clock(void);**
- time_t time(time_t* tp);
- double difftime(time_t time2, time_t time1);
- time_t mktime(struct tm* tp);
- char* asctime(const struct tm* tp);
- char* ctime(const time_t* tp);
- struct tm* gmtime(const time_t* tp);
- struct tm* localtime(const time_t* tp);
- size_t strftime(char* s, size_t smax, const char* fmt, const struct tm* tp);

Linguagem de Programação C

- Introdução
- Primeiro programa e informações essenciais
- Tipos de dados elementares
- Controle de fluxo
- Operadores e expressões
- Arrays
- Strings
- Pointers
- Classes de armazenamento
- Alocação de memória
- Estruturas e uniões
- Funções
- Pré-processador
- Entrada e saída
- Biblioteca padrão
- **C99**

C99

- C89 - ANSI
- C90 - ISO

- 1994 Iniciou revisão do padrão
- Comitê C9x conjunto ANSI/ISO: Resultou no C99

- Algumas adições
- Mais importante:

// comentário

```
for( int i=0; i<10; ++i)
   printf("%d\n",i);
```

Booleanos 1/2

- stdbool.h
  - bool
  - true
  - false
- _Bool

Booleanos 2/2

```
// boolean.c -- using a _Bool variable
#include <stdio.h>
int main(void)
{
   long num;
   long sum = 0L;
   _Bool input_is_good;

   printf("Please enter an integer to be summed ");
   printf("(q to quit): ");
   input_is_good = (scanf("%ld", &num) == 1);
   while (input_is_good)
   {
      sum = sum + num;
      printf("Please enter next integer (q to quit): ");
      input_is_good = (scanf("%ld", &num) == 1);
   }
   printf("Those integers sum to %ld.\n", sum);

   return 0;
}
```


Tipos inteiros 1/2

- inttypes.h
  - usa typedef
- int16_t
- uint32_t
- int_least8_t
- int_fast8_t
- intmax_t
- uintmax_t
- ...

Tipos inteiros 2/2

```
/* altnames.c -- portable names for integer types */
#include <stdio.h>
#include <inttypes.h> // supports portable types
int main(void)
{
   int16_t   zz16;     // zz16 is a 16-bit signed variable

   zz16 = 4593;
   printf("First, assume int16_t is short: ");
   printf("zz16 = %hd\n", zz16);
   printf("Next, let's not make any assumptions.\n");
   printf("Instead, use a \"macro\" from inttypes.h: ");
   printf("zz16 = %" PRId16 "\n", zz16);                    // Print decimal 16 = = 'd'

   return 0;
}
```


Constante ponto flutuante

- Constante ponto flutuante

  0xa.1fp10
- Valor é A.1F x 2^{10}
- Expoente é 10 decimal

Operadores lógicos

- iso646.h
- **and** no lugar de **&&**
- **or** no lugar de ||
- **not** no lugar de **!**

Array 1/2

- Inicialização de array

int a[6] = { 0, 0, 0, 0, 0, 212};

int a[6] = { [5]=212};

Array 2/2

- Array de tamanho variável

int n=5;
int m=8;
float x[n];
float y[m];

int sum2d(int rows, int cols, **int a[rows][cols]**);

Literais compostos 1/2

5
81.3
'4'
"elefante"
int x[2] = { 10, 20};

(int x[2]){10,20}

(int []){50,20,90}

int *pt;
pt = **(int[2]){10,20}**;

Literais compostos 2/2

```
#include <stdio.h>
#define COLS 4
int sum2d(int ar[][COLS], int rows);
int sum(int ar[], int n);

int main(void)
{
    int total1, total2, total3;
    int * pt1;
    int (*pt2)[COLS];

    pt1 = (int [2]) {10, 20};
    pt2 = (int [2][COLS]) { {1,2,3,-9}, {4,5,6,-8} };

    total1 = sum(pt1, 2);
    total2 = sum2d(pt2, 2);
    total3 = sum((int []){4,4,4,5,5,5}, 6);
    printf("total1 = %d\n", total1);
    printf("total2 = %d\n", total2);
    printf("total3 = %d\n", total3);
    return 0;
}
```

```
int sum(int ar[], int n)
{
    int i;
    int total = 0;

    for( i = 0; i < n; i++)
        total += ar[i];

    return total;
}

int sum2d(int ar[][COLS], int rows)
{
    int r;
    int c;
    int tot = 0;

    for (r = 0; r < rows; r++)
        for (c = 0; c < COLS; c++)
            tot += ar[r][c];
return tot;
}
```


Const e restrict

- void xxx(int *const a1, int *restrict a2, int n);
  - a1 não muda
  - a2 única forma de acessar
- void yyy(int a1[const], int a2[restrict], int n);
  - Outra forma de escrever
- double zzz(double ar[static 20]);
  - Ao menos 20

Estruturas 1/3

```
struct book {
    char title[MAX_TITLE];
    char author[MAX_AUTHOR];
    float value;
};

struct book x = {
    "the book is on the table",
    "Fulano de Tal",
    1.99
};

struct book surpresa = { .value = 10.99};

(struct book){"The Idiot", "Fyoder Dostoyevsky", 6.99};
```

Estruturas 2/3

- Flexible Arrays Members

```
struct flex{
    int count;
    double average;
    double scores[];
};

struct flex *pf;

pf = malloc( sizeof(struct flex) + 5 * sizeof(double));
```


Estruturas 3/3

```
struct flex
{
    int count;
    double average;
    double scores[];   // flexible array member
};

int main(void)
{
    struct flex * pf1, *pf2;
    int n = 5;
    int i;
    int tot = 0;
    // allocate space for structure plus array
    pf1 = malloc(sizeof(struct flex) + n * sizeof(double));
    pf1->count = n;
    for (i = 0; i < n; i++)
    {
        pf1->scores[i] = 20.0 - i;
        tot += pf1->scores[i];
    }
    pf1->average = tot / n;
    n = 9;
    tot = 0;
    pf2 = malloc(sizeof(struct flex) + n * sizeof(double));
    pf2->count = n;
    for (i = 0; i < n; i++)
    {
        pf2->scores[i] = 20.0 - i/2.0;
        tot += pf2->scores[i];
    }
    pf2->average = tot / n;
    showFlex(pf2);
    free(pf1);
    free(pf2);

    return 0;
}
```


Inline functions

```
inline void eatline()
{
    while( getchar() != '\n' )
        continue;
}

int main()
{
    ...
    eatline();
    ...
}
```


Novas bibliotecas

- **complex.h** complex arithmetic
- **fenv.h** floating-point environment
- **inttypes.h** fixed size integer types
- **stdbool.h** boolean type and values
- **tgmath.h** type-generic macros
- **wchar.h** wide-character handling
- **wctype.h** wide-character classification and mapping utilities

Linguagem de Programação C - Sumário