1. Operações com Strings
Leitura e Impressão de Strings

```
void readString(char *str, int maxLen) {
  fgets(str, maxLen, stdin);
  str[strcspn(str, "\n")] = 0; // Remove o '\n'
}

void printString(char *str) {
  printf("%s\n", str);
}
```

Comparação e Cópia de Strings

```
int compareStrings(const char *str1, const char *str2) {
  return strcmp(str1, str2);
}

void copyString(char *dest, const char *src) {
  strcpy(dest, src);
}

#include <stdio.h>
#include <string.h>

// Lê uma string do usuário, com tamanho máximo definido
void readString(char *str, int maxLen) {
  fgets(str, maxLen, stdin);          // Lê uma linha de texto
  str[strcspn(str, "\n")] = 0;        // Remove o '\n' no final, se existir
}

// Imprime uma string
void printString(char *str) {
  printf("%s\n", str);              // Exibe a string seguida por uma nova linha
}
```

Reversão de String

```
#include <string.h>

// Reverte uma string
void reverseString(char *str) {
  int len = strlen(str);            // Obtém o comprimento da string
  for (int i = 0; i < len / 2; i++) {    // Itera pela metade da string
```

```
    char temp = str[i];             // Troca os caracteres simetricamente
    str[i] = str[len - i - 1];
    str[len - i - 1] = temp;
  }
}

void reverseString(char *str) {
  int len = strlen(str);
  for (int i = 0; i < len / 2; i++) {
    char temp = str[i];
    str[i] = str[len - i - 1];
    str[len - i - 1] = temp;
  }
}
```

Contagem de Caracteres

```
// Conta quantas vezes um caractere aparece em uma string
int countChar(char *str, char ch) {
  int count = 0;                  // Inicializa o contador
  for (int i = 0; str[i] != '\0'; i++) {  // Percorre cada caractere da string
    if (str[i] == ch) count++;        // Incrementa o contador se encontrar o caractere
  }
  return count;                   // Retorna o número total de ocorrências
}


int countChar(char *str, char ch) {
  int count = 0;
  for (int i = 0; str[i] != '\0'; i++) {
    if (str[i] == ch) count++;
  }
  return count;
}
```

Comparação e Cópia de Strings

```
#include <string.h>

// Compara duas strings; retorna 0 se forem iguais
int compareStrings(const char *str1, const char *str2) {
  return strcmp(str1, str2);          // Usa a função strcmp para comparação
}
```

```
// Copia o conteúdo de uma string para outra
void copyString(char *dest, const char *src) {
  strcpy(dest, src);                // Copia 'src' para 'dest'
}
```

2. Manipulação de Arquivos
Leitura de Arquivo

```
#include <stdio.h>

// Lê um arquivo e imprime seu conteúdo no console
void readFile(const char *filename) {
  FILE *file = fopen(filename, "r");      // Abre o arquivo em modo de leitura
  if (file == NULL) {                // Verifica se o arquivo foi aberto com sucesso
    printf("Erro ao abrir o arquivo.\n");
    return;
  }

  char line[256];                    // Buffer para armazenar cada linha do arquivo
  while (fgets(line, sizeof(line), file)) { // Lê o arquivo linha por linha
    printf("%s", line);              // Imprime cada linha no console
  }

  fclose(file);                      // Fecha o arquivo
}


void readFile(const char *filename) {
  FILE *file = fopen(filename, "r");
  if (file == NULL) {
    printf("Erro ao abrir o arquivo.\n");
    return;
  }

  char line[256];
  while (fgets(line, sizeof(line), file)) {
    printf("%s", line);
  }

  fclose(file);
}
```

Escrita em Arquivo

```
#include <stdio.h>

// Escreve uma string em um arquivo
void writeFile(const char *filename, const char *content) {
   FILE *file = fopen(filename, "w");      // Abre o arquivo em modo de escrita
   if (file == NULL) {                  // Verifica se o arquivo foi aberto com sucesso
      printf("Erro ao abrir o arquivo.\n");
      return;
   }

   fprintf(file, "%s\n", content);        // Escreve o conteúdo no arquivo
   fclose(file);                        // Fecha o arquivo
}


void writeFile(const char *filename, const char *content) {
   FILE *file = fopen(filename, "w");
   if (file == NULL) {
      printf("Erro ao abrir o arquivo.\n");
      return;
   }

   fprintf(file, "%s\n", content);
   fclose(file);
}
```

3. Matemática e Operações Numéricas
Fatorial

```
long long factorialIterative(int n) {
   long long result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```

```
// Calcula o fatorial de um número de forma iterativa
long long factorialIterative(int n) {
  long long result = 1;              // Inicializa o resultado como 1
  for (int i = 1; i <= n; i++) {     // Itera de 1 até n
    result *= i;                     // Multiplica o resultado pelo número atual
  }
  return result;                     // Retorna o resultado final
}
```

Potência Rápida (Exponenciação Modular)

```
// Calcula (base^exp) % mod de forma eficiente
long long powerMod(long long base, long long exp, long long mod) {
  long long result = 1;              // Inicializa o resultado como 1
  base = base % mod;                 // Garante que base esteja no intervalo [0, mod-1]

  while (exp > 0) {                  // Enquanto houver expoente a calcular
    if (exp % 2 == 1) {              // Se o expoente for ímpar
      result = (result * base) % mod;  // Multiplica o resultado pela base
    }
    exp = exp >> 1;                  // Divide o expoente por 2
    base = (base * base) % mod;      // Eleva a base ao quadrado
  }
  return result;                     // Retorna o resultado
}
```

```
long long powerMod(long long base, long long exp, long long mod) {
  long long result = 1;
  base = base % mod;

  while (exp > 0) {
    if (exp % 2 == 1)
      result = (result * base) % mod;
    exp = exp >> 1;
    base = (base * base) % mod;
  }
  return result;
}
```

Raiz Quadrada Inteira

```
// Calcula a parte inteira da raiz quadrada de um número
int intSqrt(int x) {
  int res = 0;                    // Inicializa o resultado como 0
  int bit = 1 << 30;              // Começa com o maior bit possível

  while (bit > x)                 // Ajusta o bit inicial se for muito grande
    bit >>= 2;

  while (bit != 0) {              // Processa cada bit relevante
    if (x >= res + bit) {         // Se o valor ajustado for menor ou igual a x
      x -= res + bit;             // Reduz x
      res = (res >> 1) + bit;     // Atualiza o resultado
    } else {
      res >>= 1;                  // Caso contrário, apenas reduz o resultado
    }
    bit >>= 2;                    // Move para o próximo bit
  }
  return res;                     // Retorna a parte inteira da raiz quadrada
}
```

```
int intSqrt(int x) {
  int res = 0;
  int bit = 1 << 30;

  while (bit > x)
    bit >>= 2;

  while (bit != 0) {
    if (x >= res + bit) {
      x -= res + bit;
      res = (res >> 1) + bit;
    } else {
      res >>= 1;
    }
    bit >>= 2;
  }
  return res;
}
```

4. Algoritmos de Busca e Ordenação

Busca Linear

```
// Busca um elemento em um array de forma linear
int linearSearch(int arr[], int n, int x) {
  for (int i = 0; i < n; i++) {         // Itera por todos os elementos do array
    if (arr[i] == x) return i;         // Retorna o índice se encontrar o elemento
  }
  return -1;                          // Retorna -1 se o elemento não for encontrado
}
```

```
int linearSearch(int arr[], int n, int x) {
  for (int i = 0; i < n; i++) {
    if (arr[i] == x) return i;
  }
  return -1;
}
```

Merge Sort

```
#include <stdio.h>

// Junta duas metades ordenadas de um array
void merge(int arr[], int l, int m, int r) {
  int n1 = m - l + 1;                 // Tamanho da primeira metade
  int n2 = r - m;                     // Tamanho da segunda metade

  int L[n1], R[n2];                   // Arrays temporários para as metades
  for (int i = 0; i < n1; i++)        // Copia os elementos para o array L
    L[i] = arr[l + i];
  for (int j = 0; j < n2; j++)        // Copia os elementos para o array R
    R[j] = arr[m + 1 + j];

  int i = 0, j = 0, k = l;            // Índices iniciais de L, R e arr
  while (i < n1 && j < n2) {          // Combina os arrays em ordem
    if (L[i] <= R[j]) arr[k++] = L[i++];
    else arr[k++] = R[j++];
  }

  while (i < n1) arr[k++] = L[i++];   // Copia os elementos restantes de L
  while (j < n2) arr[k++] = R[j++];   // Copia os elementos restantes de R
}
```

```
// Função principal do Merge Sort
void mergeSort(int arr[], int l, int r) {
  if (l < r) {                    // Divide o array até que tenha tamanho 1
    int m = l + (r - l) / 2;          // Calcula o ponto médio

    mergeSort(arr, l, m);             // Ordena a primeira metade
    mergeSort(arr, m + 1, r);         // Ordena a segunda metade
    merge(arr, l, m, r);              // Combina as duas metades
  }
}



void merge(int arr[], int l, int m, int r) {
  int n1 = m - l + 1;
  int n2 = r - m;

  int L[n1], R[n2];
  for (int i = 0; i < n1; i++)
    L[i] = arr[l + i];
  for (int j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];

  int i = 0, j = 0, k = l;
  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) arr[k++] = L[i++];
    else arr[k++] = R[j++];
  }

  while (i < n1) arr[k++] = L[i++];
  while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);
    merge(arr, l, m, r);
  }
}
```

5. Estruturas de Dados Avançadas

Pilha

```
#define MAX 100
typedef struct {
   int data[MAX];
   int top;
} Stack;

void initStack(Stack *s) {
   s->top = -1;
}

int isEmpty(Stack *s) {
   return s->top == -1;
}

int isFull(Stack *s) {
   return s->top == MAX - 1;
}

void push(Stack *s, int value) {
   if (isFull(s)) return;
   s->data[++s->top] = value;
}

int pop(Stack *s) {
   if (isEmpty(s)) return -1;
   return s->data[s->top--];
}
```

Pilha (Stack)
A pilha segue o princípio LIFO (Last In, First Out), ou seja, o último elemento inserido é o primeiro a ser removido.

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

// Definição de uma estrutura de Pilha
typedef struct {
   int *arr;      // Array para armazenar os elementos da pilha
   int top;       // Índice do topo da pilha
   int capacity;   // Capacidade máxima da pilha
} Stack;
```

```
// Função para criar uma pilha com capacidade inicial
Stack* createStack(int capacity) {
  Stack *stack = (Stack *)malloc(sizeof(Stack));  // Aloca memória para a pilha
  stack->capacity = capacity;                // Define a capacidade máxima
  stack->top = -1;                           // Inicializa o topo como -1 (indica que a pilha está vazia)
  stack->arr = (int *)malloc(capacity * sizeof(int)); // Aloca o array para os elementos
  return stack;                              // Retorna o ponteiro para a pilha
}

// Verifica se a pilha está cheia
int isFull(Stack *stack) {
  return stack->top == stack->capacity - 1;  // Retorna 1 se a pilha estiver cheia, 0 caso
contrário
}

// Verifica se a pilha está vazia
int isEmpty(Stack *stack) {
  return stack->top == -1;                   // Retorna 1 se a pilha estiver vazia, 0 caso contrário
}

// Adiciona um elemento à pilha
void push(Stack *stack, int value) {
  if (isFull(stack)) {                       // Verifica se a pilha está cheia
    printf("Pilha cheia!\n");
    return;
  }
  stack->arr[++stack->top] = value;          // Adiciona o valor no topo da pilha e incrementa
o índice do topo
}

// Remove o elemento do topo da pilha
int pop(Stack *stack) {
  if (isEmpty(stack)) {                      // Verifica se a pilha está vazia
    printf("Pilha vazia!\n");
    return -1;                               // Retorna -1 se a pilha estiver vazia
  }
  return stack->arr[stack->top--];           // Retorna o elemento do topo e decrementa o
índice do topo
}

// Retorna o elemento no topo da pilha sem removê-lo
int peek(Stack *stack) {
  if (isEmpty(stack)) {                      // Verifica se a pilha está vazia
```

```
    printf("Pilha vazia!\n");
    return -1;                    // Retorna -1 se a pilha estiver vazia
  }
  return stack->arr[stack->top];          // Retorna o valor do topo da pilha
}

// Libera a memória alocada para a pilha
void freeStack(Stack *stack) {
  free(stack->arr);                 // Libera o array da pilha
  free(stack);                      // Libera a estrutura da pilha
}
```

Fila (Queue)

A fila segue o princípio FIFO (First In, First Out), ou seja, o primeiro elemento inserido é o primeiro a ser removido.

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

// Definição de uma estrutura de Fila
typedef struct {
  int *arr;     // Array para armazenar os elementos da fila
  int front;    // Índice do primeiro elemento
  int rear;     // Índice do último elemento
  int capacity;  // Capacidade máxima da fila
} Queue;

// Função para criar uma fila com capacidade inicial
Queue* createQueue(int capacity) {
  Queue *queue = (Queue *)malloc(sizeof(Queue)); // Aloca memória para a fila
  queue->capacity = capacity;             // Define a capacidade da fila
  queue->front = 0;                       // Inicializa o início da fila
  queue->rear = -1;                       // Inicializa o fim da fila
  queue->arr = (int *)malloc(capacity * sizeof(int)); // Aloca o array para os elementos
  return queue;                           // Retorna o ponteiro para a fila
}

// Verifica se a fila está cheia
int isFullQueue(Queue *queue) {
  return queue->rear == queue->capacity - 1;   // Retorna 1 se a fila estiver cheia, 0 caso
contrário
}
```

```
// Verifica se a fila está vazia
int isEmptyQueue(Queue *queue) {
  return queue->front > queue->rear;          // Retorna 1 se a fila estiver vazia, 0 caso
contrário
}

// Adiciona um elemento à fila
void enqueue(Queue *queue, int value) {
  if (isFullQueue(queue)) {                // Verifica se a fila está cheia
    printf("Fila cheia!\n");
    return;
  }
  queue->arr[++queue->rear] = value;          // Adiciona o valor ao final da fila e incrementa
o índice do fim
}

// Remove o elemento da frente da fila
int dequeue(Queue *queue) {
  if (isEmptyQueue(queue)) {                // Verifica se a fila está vazia
    printf("Fila vazia!\n");
    return -1;                        // Retorna -1 se a fila estiver vazia
  }
  return queue->arr[queue->front++];          // Retorna o valor da frente e incrementa o
índice do início
}

// Retorna o elemento da frente da fila sem removê-lo
int front(Queue *queue) {
  if (isEmptyQueue(queue)) {                // Verifica se a fila está vazia
    printf("Fila vazia!\n");
    return -1;                        // Retorna -1 se a fila estiver vazia
  }
  return queue->arr[queue->front];            // Retorna o valor da frente da fila
}

// Libera a memória alocada para a fila
void freeQueue(Queue *queue) {
  free(queue->arr);                     // Libera o array da fila
  free(queue);                        // Libera a estrutura da fila
}
```

6. Funções de Utilidade Geral

Troca de Variáveis (Swap)

```
// Troca os valores de duas variáveis
void swap(int *a, int *b) {
   int temp = *a;  // Armazena o valor de a em uma variável temporária
   *a = *b;       // Atribui o valor de b a a
   *b = temp;     // Atribui o valor da variável temporária (valor antigo de a) a b
}
```

Verificação de Número Primo

```
#include <math.h>

// Verifica se um número é primo
int isPrime(int n) {
   if (n <= 1) return 0;                // Números menores ou iguais a 1 não são primos
   for (int i = 2; i <= sqrt(n); i++) {     // Checa divisores até a raiz quadrada de n
      if (n % i == 0) return 0;           // Se encontrar um divisor, retorna 0 (não primo)
   }
   return 1;                           // Se não encontrar divisores, retorna 1 (primo)
}
```

Leitura de Inteiros com Validação

```
// Lê um número inteiro do usuário, com validação
int readInteger() {
   int num;
   while (scanf("%d", &num) != 1) {        // Tenta ler um número inteiro
      printf("Entrada inválida. Tente novamente: ");
      while (getchar() != '\n');           // Limpa o buffer de entrada
   }
   return num;                         // Retorna o número lido
}
```

Conversão de Inteiro para String

```
void intToStr(int num, char *str) {
   sprintf(str, "%d", num);
}
```

Conversão de String para Inteiro

```
int strToInt(char *str) {
```

```
    return atoi(str);
}
```

Geração de Números Aleatórios

```
void generateRandomNumbers(int arr[], int n, int min, int max) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = min + rand() % (max - min + 1);
    }
}
```