Resumo de Programação em C para Iniciantes

Este resumo cobre os tópicos principais da linguagem C com base no manual de iniciante, estruturado para caber em folhas A4 (frente e verso, use fonte 10-12pt, margens padrão). Inclui explicações detalhadas de cada parte dos códigos, de todos os tópicos, e foco em format specifiers como %u (para unsigned int). Estude os exemplos para conseguir criar códigos semelhantes. Compile com gcc (ex: gcc arquivo.c -o exec).

1. Introdução e História do C
- C é uma linguagem procedural de baixo nível, criada por Dennis Ritchie em 1972 nos Bell Labs para o UNIX.
- Usada em sistemas operacionais, jogos e hardware. É compilada (código fonte → executável).
- Vantagens: rápida, portátil, controle de memória.

2. Instalação e Configuração
- Instale um compilador: GCC (gratuito, via MinGW no Windows, apt no Linux, Xcode no Mac).
- Editor: VS Code, Code::Blocks ou Notepad++.
- Compile: `gcc hello.c -o hello` e execute `./hello`.

3. Primeiro Programa (Hello World)
Exemplo:
```c
#include <stdio.h>  // Inclui biblioteca padrão para input/output (stdio.h define printf).
int main() {        // Função principal; todo programa C inicia aqui. Retorna int (0 para sucesso).
    printf("Hello, World!\n");  // Imprime string na tela; \n é nova linha.
    return 0;       // Encerra o programa com código de saída 0 (sucesso).
}
```
- `#include`: Pré-processador para importar headers.
- `main()`: Ponto de entrada; chaves {} delimitam bloco.
- `printf`: Função de saída; aspas "" para strings.

4. Variáveis e Tipos de Dados
- Variáveis armazenam dados; declaradas com tipo (ex: int x;).
- Tipos principais:
  - int: Inteiro assinado (-2^31 a 2^31-1).
  - unsigned int: Inteiro não assinado (0 a 2^32-1).
  - float: Ponto flutuante (decimais).
  - double: Float com mais precisão.
  - char: Caractere (1 byte, ex: 'a').
  - short, long: Modificadores de tamanho.

Exemplo:
```c
```

```
int idade = 25;     // Declara 'idade' como int e inicializa com 25 (armazena em memória).
float altura = 1.75; // Declara float e atribui valor decimal.
char letra = 'A';   // Armazena caractere ASCII.
```
- Atribuição (=): Coloca valor na variável.

5. Constantes
- Valores imutáveis; use #define ou const.
Exemplo:
```c
#define PI 3.14     // Pré-processador define constante global (substitui texto em compile-time).
const int MAX = 100; // Constante local à função; não pode ser alterada após declaração.
```
- #define: Não usa memória; substituição textual.
- const: Usa memória, mas protege contra mudanças.

6. Input/Output (Entrada/Saída)
- Use <stdio.h> para printf (saída) e scanf (entrada).
- Format specifiers: Placeholders em strings para inserir valores.
  - %d ou %i: Inteiro assinado (ex: -5).
  - %u: Unsigned int (inteiro positivo, ex: 5).
  - %f: Float (decimal, ex: 3.14).
  - %lf: Double.
  - %c: Char (caractere).
  - %s: String (sequência de chars).
  - %p: Ponteiro (endereço de memória).
Exemplo:
```c
int num;
printf("Digite um número: ");  // Imprime prompt sem format.
scanf("%d", &num);             // Lê int do teclado; &num passa endereço para armazenar valor.
printf("Número: %d\n", num);   // Imprime valor; %d formata int.
unsigned int positivo = 10;
printf("Unsigned: %u\n", positivo);  // %u para unsigned, evita erros com negativos.
```
- & (endereço): Necessário em scanf para modificar variável.
- printf/scanf: Retornam número de itens processados.

7. Operadores
- Aritméticos: + (soma), - (subtração), * (multiplicação), / (divisão), % (módulo, resto).
- Relacionais: == (igual), != (diferente), >, <, >=, <=.
- Lógicos: && (E), || (OU), ! (NÃO).
- Atribuição: =, +=, -=, etc.
- Bitwise: & (AND), | (OR), ^ (XOR), << (shift left), >> (shift right).

Exemplo:
```c
int a = 5, b = 3;
int soma = a + b;   // Soma valores e atribui (8).
if (a > b) {        // Verifica relacional; true aqui.
    a += 2;         // Incrementa a por 2 (a vira 7).
}
```

8. Condicionais (If/Else, Switch)
- Controle de fluxo baseado em condições.
Exemplo If/Else:
```c
int nota = 7;
if (nota >= 7) {    // Verifica condição; executa bloco se true.
    printf("Aprovado\n");
} else {            // Executa se if false.
    printf("Reprovado\n");
}
```
Exemplo Switch:
```c
char op = '+';
switch (op) {       // Avalia 'op' e pula para case correspondente.
    case '+': printf("Soma\n"); break;  // Executa e sai com break.
    case '-': printf("Subtração\n"); break;
    default: printf("Inválido\n");  // Se nenhum case.
}
```
- break: Evita fall-through (executar próximos cases).

9. Loops (For, While, Do-While)
- Repetição de código.
Exemplo For:
```c
for (int i = 0; i < 5; i++) {  // Inicializa i=0; verifica i<5; incrementa i++ após cada iteração.
    printf("%d\n", i);         // Imprime 0 a 4.
}
```
Exemplo While:
```c
int i = 0;
while (i < 5) {     // Verifica condição antes; loop infinito se não alterar i.
    printf("%d\n", i);
```

```
    i++;            // Incrementa para sair.
}
```
Exemplo Do-While:
```c
int i = 0;
do {                // Executa pelo menos uma vez; verifica após.
    printf("%d\n", i);
    i++;
} while (i < 5);
```

10. Arrays
- Coleção de elementos do mesmo tipo, índice 0-based.
Exemplo:
```c
int nums[5] = {1, 2, 3, 4, 5};  // Declara array de 5 ints, inicializa.
nums[0] = 10;                   // Altera primeiro elemento.
for (int i = 0; i < 5; i++) {   // Acessa com índice.
    printf("%d ", nums[i]);
}
```
- Tamanho fixo; não redimensionável.

11. Strings
- Arrays de chars terminados por '\0' (null terminator).
Exemplo:
```c
char nome[10] = "João";     // Armazena string; espaço para '\0'.
printf("%s\n", nome);       // %s imprime até '\0'.
strcpy(nome, "Maria");      // Copia string (de <string.h>).
```
- Funções: strlen (tamanho), strcat (concatena), strcmp (compara).

12. Funções
- Blocos reutilizáveis; declaradas com tipo de retorno.
Exemplo:
```c
int soma(int a, int b) {    // Declara função; parâmetros a e b.
    return a + b;           // Retorna valor.
}
int main() {
    int res = soma(3, 4);   // Chama função; res=7.
    return 0;
```

```
}
```
- Protótipos: Declare antes de main se definida depois.

13. Ponteiros (Pointers)
- Variáveis que armazenam endereços de memória.
Exemplo:
```c
int x = 10;
int *p = &x;               // Declara ponteiro; &x pega endereço de x.
printf("%p\n", p);         // %p imprime endereço.
*p = 20;                   // * (dereference) altera valor em endereço (x vira 20).
```
- Útil para arrays, funções e alocação dinâmica (malloc).

14. Estruturas (Structs)
- Tipos compostos personalizados.
Exemplo:
```c
struct Pessoa {            // Define struct com campos.
    char nome[20];
    int idade;
};
struct Pessoa p1 = {"Ana", 30};  // Inicializa.
printf("%s, %d\n", p1.nome, p1.idade);  // Acessa com . (dot).
```

15. Manipulação de Arquivos (File Handling)
- Use <stdio.h> para fopen, fclose, fprintf, fscanf.
Exemplo:
```c
FILE *f = fopen("arquivo.txt", "w");  // Abre arquivo para escrita ("r" para leitura).
if (f != NULL) {                      // Verifica se abriu.
    fprintf(f, "Texto\n");            // Escreve como printf.
    fclose(f);                        // Fecha arquivo.
}
```
- Modos: "w" (write), "r" (read), "a" (append).

Dicas Finais para Criar Qualquer Código
- Entenda o fluxo: main → funções → loops/condicionais.
- Depure com printf para ver valores.
- Inclua headers necessários (ex: <string.h> para strings).
- Evite erros comuns: faltar ;, mismatched {}, overflow.

- Pratique: Escreva códigos simples variando exemplos acima.

Imprima em A4: Frente (1-8), Verso (9-15 + dicas). Estude códigos linha por linha para entender funcionamento. Boa sorte na matéria!