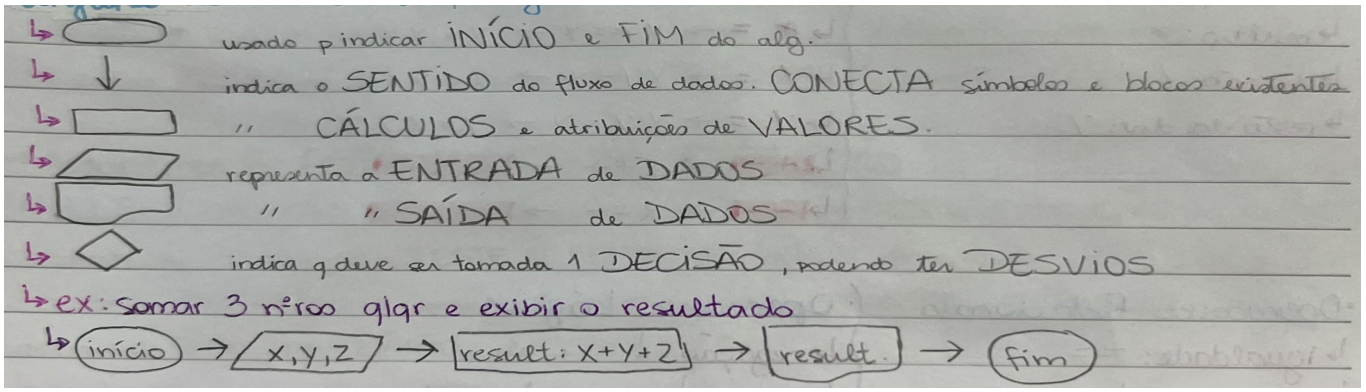

➢ TIPOS BÁSICOS EM C

→ **INT =** número inteiro, sem casa decimal
→ **FLOAT / DOUBLE (+ preciso) =** ponto flutuante, com casa decimal
→ **BOOL =** lógico (true/false)
→ **CHAR =** caracter

➢ SAÍDA FORMATADA

→ **INT**: %d
→ **FLOAT:** %f ou %.xf
→ **DOUBLE:** %lf ou %.xlf
→ **BOOL:** %b
→ **CHAR:** %c
→ **.x =** opcional de CASAS DECIMAIS

➢ OPERAÇÕES LÓGICOS

→ **AND:** && (se tem 1 proposição FALSA, acabou)
→ **OR:** || (se tem 1 proposição VERDADEIRA, ta ok, continua)

➢ ESTRUTURAS CONDICIONAIS

→ **IF:** o bloco de comandos 1 somente será executado SE a CONDIÇÃO for VERDADEIRA, caso contrário o bloco será IGNORADO.
→ **IF-ELSE:** o bloco de comandos 1 somente será executado SE a CONDIÇÃO for VERDADEIRA, caso contrário o bloco será IGNORADO e o bloco 2 será EXECUTADO
↳ Apenas 1 dos blocos pode ser executado.
→ **IF-ELSE-IF:** Os blocos são avaliados de CIMA p/ BAIXO desde que NENHUMA CONDIÇÃO ANTERIOR tenha sido VERDADEIRA. Nesse caso o bloco da condição verdadeira é EXECUTADO e os demais são IGNORADOS
↳ O bloco ELSE é executado AUTOMATICAMENTE caso todas as condições sejam FALSAS.
→ **SWITCH:** Ocorre 1 AVALIAÇÃO da correspondência de 1 EXPRESSÃO com as opções disponíveis.
↳ Utilizaremos o PADRÃO DE UMA CONSTANTE, ou seja, o conteúdo de 1 variável é COMPARADO às constantes disponíveis nas opções. Apenas UMA opção é executada ou NENHUMA delas.
↳ É OPCIONAL indicar a situação DEFAULT, que é executada no caso de NÃO OCORRER correspondência as demais opções.
↳ O uso do BREAK no final de cada caso é OBRIGATÓRIO

➢ ESTRUTURAS DE REPETIÇÃO

→ **WHILE (ENQUANTO)**: a AVALIAÇÃO da expressão condicional ocorre ANTES da EXECUÇÃO do BLOCO de comando, por isso se diz que o TESTE É FEITO NO INÍCIO da repetição
↳ Se ela for VERDADE, o bloco de comando é executado novamente e APÓS SEU TÉRMINO, avalia-se novamente a condição. CASO CONTRÁRIO, ele é IGNORADO
↳ Exemplo: Imprima os primeiros 1000 números a partir do 1:

```
int i=1;              |  printf("%d", &i);
while(i<=1000) {      |  i++;
```

→ **DO WHILE (FAÇA-ENQUANTO)**: a AVALIAÇÃO da expressão condicional ocorre APÓS a EXECUÇÃO do BLOCO de comando, por isso se diz que o TESTE É FEITO NO FINAL da repetição
↳ Se ela for VERDADE, o bloco de comando é executado novamente e APÓS SEU TÉRMINO, avalia-se novamente a condição. CASO CONTRÁRIO, ele NÃO SERÁ REPETIDO
↳PELO MENOS 1 VEZ ocorre a execução do bloco, INDEPENDENTE do resultado da condição
→ **FOR (PARA):** estrutura composta por 3 EXPRESSÕES entre os (): INICIALIZAÇÃO da variável, CONDIÇÃO e ATUALIZAÇÃO da variável de controle
↳ a INICIALIZAÇÃO é feita 1 ÚNICA VEZ, no INÍCIO da expressão da estrutura para a execução do bloco de comandos, avalia-se a CONDIÇÃO, se ela for VERDADEIRA é executada, se não, é ignorado. após CADA TÉRMINO da execução do bloco, a ATUALIZAÇÃO é realizada
↳ EXEMPLO: escrever os números de 1 a 3

```
for (int i=1; i <4; i++){
```

```
    printf (“%d”, &i)
  }
  ↳ Exemplo:
  int valor = 5
  for (int i=0; i <4; i+=1; valor += 5){
    printf (“%d”, &valor)
  }
  printf (“valor: %d”, valor)
}
```

➢ MODULARIZAÇÃO

→ **PROCEDIMENTO**: módulo SEM retorno de resultado e usa VOID alguma coisa (variável)

```
int menu() {
 int opcao;
 printf(“Escolha uma das opções”);
 printf(“1 –Somar”);
 printf(“2 –Subtrair”);
 printf(“3 –Multiplicar”);
 printf(“4 –Dividir”);
 printf(“Opção : ”);
 scanf(“% d”, &opcao);
 return (opcao);
}
int main() {
 int resposta;
 resposta = menu();
 printf(“A opção escolhida foi % d”, resposta);
 return 0;
}
```

→ **FUNÇÃO**: módulo COM retorno de resultado e usa FLOAT (ou INT/DOUBLE) alguma coisa (variável);

```
void menu() {
 printf(“Escolha uma das opções”);
 printf(“1 – Somar”);
 printf(“2 – Subtrair”);
 printf(“3 – Multiplicar”);
 printf(“4 – Dividir”);
 printf(“Opção : ”);
}
int main() {
 int resposta;
 menu();
 scanf(“% d”, &resposta);
 printf(“A opção escolhida foi % d”, resposta);
 return 0;
}
```

→ **MODULARIZAÇÃO:** após a criação de 1 módulo, ele pode ser adicionado VÁRIAS VEZES e em QUALQUER LUGAR do código, inclusive dentro de outro módulo.

```
double fatorial(int x) {
 double resultado = 1.0;
 if (x > 1) {
  for (int val = 2; val <= x; val++) {
   resultado *= val;
  }
 }
 return resultado;
}
double calculaSerie(int n) {
 double S = 0;
 int sinal = 1;
 for (int i = 1; i <= n; i++) {
```

```
    S += sinal / fatorial(i);
    sinal *= (-1);
  }
  return S;
}
int main() {
  int termos;
  printf("Quantos termos tem a expressão? ");
  scanf(“%d”, &termos);
  printf("O resultado é %.2f", calculaSerie(termos));
  return 0;
}
```

IF ANINHADOS E IF INDEPENDENTE: aninhado é 1 dentro do outro, e o que tá dentro só roda se o que tiver em cima tiver verdadeiro// o independente não é 1 dentro do outro, e se ele tiver o primeiro certo, ele não analisa os outros e pula fora, ou analisa 1 até achar 1 verdadeiro para rodar

IF INDEPENDENTE E IF ELSE IF: IF IND= Analisa cada um de forma independente

IF ELSE IF= Se o IF for verdadeiro o Else If será ignorado, caso contrário o Else If será executado automaticamente

FUNÇÃO RECURSIVA QUE CALCULA DIVISÃO USANDO SUBTRAÇÕES SUCESSIVAS:

```
#include <stdio.h>

int divisao (int numerador, int denominador) {
  int subt=numerador-denominador;

  if(subt!=0) {
    return (1+divisao(subt, denominador));
  } else {
    return 1;
  }
}

int main(void) {
  int n, d;
  printf("Insira dois números inteiros\n");
  scanf("%d %d", &n, &d);
  divisao(n, d);

  printf("O resultado é %d", divisao(n, d));
  return 0;
}
```

FIBONACCI

```
#include <stdio.h>

int main(void) {
  int a=0, b=1, n, s;
  printf("digite um número para obter a serie de fibonacci:\n");
  scanf("%d", &n);
  printf("\na sequência é:\n");
  printf("%d\n", a);
  s=1;
 while (s<n){ //dá a sequência de números até o número digitado
  printf("%d\n", s);
  s=a+b;
  a=b;
  b=s;
 }
return 0;
}
```