IF61C—Introdução ao uso de arquivos de cabeçalho

Suponha o seguinte problema: é preciso fazer um programa para armazenar a data de nascimento de uma pessoa e calcular sua idade em relação a uma data lida do teclado. O seguinte código seria um solução simplificada:

```c
#include<stdio.h>
//struct que representa o tipo Pessoa
typedef struct pessoa{
    int diaP;
    int mesP;
    int anoP;
    int idadeP;
}Pessoa;

int main(int argc, char *argv[])
{
    Pessoa Einstein, Newton;
    int idadeEinstein, idadeNewton,
         anoAT, mesAT, diaAT;

    //preenchimento dos dados das estruturas criadas
    Einstein.diaP = 14; Einstein.mesP = 3; Einstein.anoP = 1879;
    Newton.diaP = 4; Newton.mesP = 1; Newton.anoP = 1643;

    printf("Digite o dia, mes e ano atual: "); //leitura da data atual
    scanf("%d %d %d", &diaAT, &mesAT, &anoAT);

    //calculo da idade de Einstein
    Einstein.idadeP = anoAT - Einstein.anoP;
    if (Einstein.mesP <  mesAT)
       Einstein.idadeP = Einstein.idadeP - 1;
    else
        if (Einstein.mesP == mesAT)
            if (Einstein.diaP < diaAT)
              Einstein.idadeP = Einstein.idadeP - 1;

    //calculo da idade de Newton
    Newton.idadeP = anoAT - Newton.anoP;
    if (Newton.mesP <  mesAT)
       Newton.idadeP = Newton.idadeP - 1;
    else
       if (Newton.mesP == mesAT)
            if (Newton.diaP < diaAT)
              Newton.idadeP = Newton.idadeP - 1;

    printf("Einstein teria %d \n", Einstein.idadeP); //impressao
    printf("Newton teria %d   \n", Newton.idadeP);

    return 0;
}
```

exemploInicialSemFuncao.c

O código acima apresenta como principal problema a falta de modularização, ou seja, do uso de funções. Isso deixa o código confuso e de difícil extensão (observe que é preciso reescrever o código do cálculo da idade para cada pessoa). Portanto, vamos criar uma nova versão, apresentada na sequência. Observe que as estruturas são passadas por referência.

```c
#include<stdio.h>
typedef struct pessoa{
    int diaP;
    int mesP;
    int anoP;
    int idadeP;
}Pessoa;

void preenche_CamposPessoa(Pessoa *p, int diaNa, int mesNa, int anoNa);
void calc_Idade(Pessoa *p, int diaAT, int mesAT, int anoAT);
int informaIdade(Pessoa p);

int main(int argc, char *argv[])
{
    Pessoa Einstein, Newton;
    int idadeEinstein, idadeNewton;

    //invoca a funcao para preencher campos (estrutura eh passada por referencia
       )
    preenche_CamposPessoa(&Einstein, 14, 3, 1879);
    preenche_CamposPessoa(&Newton, 4, 1, 1643);

    //calculo da idade de cada pessoa (a estrutura eh passada por referencia)
    calc_Idade(&Einstein, 7, 10, 2013);
    calc_Idade(&Newton, 7, 10, 2013);

    // impressao da idade calculada
    printf("Einstein teria %d \n", informaIdade(Einstein));
    printf("Newton teria %d   \n", informaIdade(Newton));
    return 0;
}

void preenche_CamposPessoa(Pessoa *p, int diaNa, int mesNa, int anoNa){
    p->diaP   = diaNa; // note que o acesso se dah via ponteiros, dado que
    p->mesP   = mesNa; // a estrutura eh passada por referencia
    p->anoP = anoNa;
}

void calc_Idade(Pessoa *p, int diaAT, int mesAT, int anoAT){
   p->idadeP = anoAT - p->anoP; // note que o acesso se dah via ponteiros, dado
   if (p->mesP <  mesAT)        //  que a estrutura eh passada por referencia
      p->idadeP = p->idadeP - 1;
   else
       if (p->mesP == mesAT)
          if (p->diaP < diaAT)
              p->idadeP = p->idadeP - 1;
}

int informaIdade(Pessoa p){
   return p.idadeP;
}
```

exemploInicialComFuncao.c

Criação de bibliotecas

Agora, iremos separar este programa em três arquivos:

1. `pessoa.h`: contém a definição da estrutura `Pessoa`, bem como o cabeçalho de todas as funções relacionadas;
2. `pessoa.c`: contém a implementação de todas as funções relacionadas à estrutura `Pessoa`;
3. `principal.c`: contém a função `main()`, bem como as chamadas às funções implementadas para atender ao objetivo do programa.

Ao separar os arquivos desta forma, estamos criando uma biblioteca. Além de deixar o programa mais claro, a reutilização de código fica mais "prática", dado que não é preciso copiar e colar trechos de código (basta apenas fazer `#include''minhaBiblioteca.h''`). O arquivo com extensão `.h` é denominado arquivo de cabeçalho. O código de cada arquivo é ilustrado na sequência.

```
class Pessoa
{
  private:
       int diaP;
       int mesP;
       int anoP;
       int idadeP;

  public:
     Pessoa(int diaNa, int mesNa, int anoNa);
     void Calc_Idade(int diaAT, int mesAT, int anoAT);
     int informaIdade();
};
```

pessoa.h

```
#include "pessoa.h"

void preencheDataNasc(Pessoa *p, int diaNa, int mesNa, int anoNa){
    p->diaP   = diaNa;
    p->mesP   = mesNa;
    p->anoP = anoNa;
}

void calcIdade(Pessoa *p, int diaAT, int mesAT, int anoAT){

   p->idadeP = anoAT - p->anoP;
   if (p->mesP <  mesAT)
      p->idadeP = p->idadeP - 1;
   else
       if (p->mesP == mesAT)
            if (p->diaP < diaAT)
              p->idadeP = p->idadeP - 1;
}

int informaIdade(Pessoa p){
    return p.idadeP;
}
```

pessoa.c

```c
#include<stdio.h>
#include "pessoa.h"

int main(int argc, char *argv[])
{
    Pessoa Einstein, Newton;
    int idadeEinstein, idadeNewton;

    // invocando a funcao que preenche os dados
    // note que a estrutura eh passada por referencia)
    preencheDataNasc(&Einstein, 14, 3, 1879);
    preencheDataNasc(&Newton, 4, 1, 1643);

    // calculo da idade de cada pessoa
    // note que a estrutura eh passada por referencia
    calcIdade(&Einstein, 7, 10, 2013);
    calcIdade(&Newton, 7, 10, 2013);

    // impressao dos resultados retornados
    printf("Einstein teria %d \n", informaIdade(Einstein));
    printf("Newton teria %d   \n", informaIdade(Newton));

    return 0;
}
```

principal.c

Note que na diretiva `#include` são usadas aspas ao invés dos símbolos <> - eles são usados apenas quando o arquivo de cabeçalho estiver instalado em um diretório padrão do sistema.

Agora vem a pergunta: como devo compilar o arquivo `principal.c`? Ao tentar compilar este arquivo individualmente, temos os seguintes erros:

```
leyza@leyza ~/Desktop/arquivos_h $ gcc principal.c -o nomeExecutavel
/tmp/ccLVoXp4.o: In function ‘main’:
principal.c:(.text+0x29): undefined reference to ‘preencheDataNasc’
principal.c:(.text+0x4d): undefined reference to ‘preencheDataNasc’
principal.c:(.text+0x71): undefined reference to ‘calcIdade’
principal.c:(.text+0x95): undefined reference to ‘calcIdade’
principal.c:(.text+0xb9): undefined reference to ‘informaIdade’
principal.c:(.text+0xee): undefined reference to ‘informaIdade’
collect2: ld returned 1 exit status
```

Note que, embora tenhamos solicitado a inclusão da biblioteca, o compilador não está “enxergando” as funções nela implementadas. Para resolver este problema, existem diversas alternativas:

1. Compilar via linha de comando indicando todos os arquivos envolvidos. Para o exemplo acima:

   ```
   leyza@leyza ~/Desktop/arquivos_h $ gcc principal.c pessoa.c -o nomeExecutavel
   leyza@leyza ~/Desktop/arquivos_h $ ./nomeExecutavel
   Einstein teria 133
   Newton teria 369
   ```

2. Criar um projeto (no CodeBlocks, por exemplo) e incluir neste projetos todos os arquivos (no caso deste exemplo, `pessoa.c`, `pessoa.h` e `principal.c`;

3. Criar um arquivo `Makefile`

Criando Makefiles

Em suma, um arquivo `Makefile` é um arquivo texto composto por alvos (*targets*), dependências (*dependencies*) e comandos (*comandos*) segundo a estrutura:

```
target: dependencies
<TAB> commands
```

em que `<TAB>` denota uma tabulação. Para executá-lo, basta abrir o terminal na mesma pasta em que o arquivo se encontra e digitar `make` (no Windows, é preciso baixar o programa `make`).

Na sequência, serão ilustrados alguns possíveis arquivos `Makefiles` (para uma explicação mais detalhada, leia o material complementar disponibilizado no moodle):

1. Versão mais simples, onde considera-se apenas o alvo padrão, `all`. Ele é executado sem qualquer dependência.

   ```
   all:
       gcc principal.c pessoa.c -o nomeExecutavel01
   ```

2. Versão com múltiplos alvos, com diferentes dependências:

   ```
   principal: depPessoa depPrincipal
       gcc principal.o pessoa.o -o nomeExecutavel
   depPrincipal: principal.c
       gcc -c principal.c
   depPessoa: pessoa.c
       gcc -c pessoa.c
   ```

   Neste exemplo, para o alvo denominado `principal` ser executado, é preciso resolver duas dependëncias: `depPessoa` e `depPrincipal`. Portanto, os respectivos alvos precisam ser executados antes. Neste exemplo, ao executarmos `make`, a seguinte sequência de comandos é considerada:

   ```
   leyza@leyza ~/Desktop/arquivos_h $ make
   gcc -c pessoa.c
   gcc -c principal.c
   gcc principal.o pessoa.o -o nomeExecutavel
   ```

3. Criando variáveis para representar o compilador e as opções de compilação.

   ```
   # Comentario do makefile
   # utilizaremos a variavel CP para definir o compilador a ser utilizado
   CP=gcc

   # a variavel COPT sera utilizada para adicionar opcoes a compilacao
   COPT= -c

   principal: depPessoa depPrincipal
       $(CP) principal.o pessoa.o -o nomeExecutavel
   depPrincipal: principal.c
       $(CP) $(COPT) principal.c
   depPessoa: pessoa.c
       $(CP) $(COPT) pessoa.c

   # apaga todos os arquivos com extensao .o criados
   clean:
       rm -rf *o programa
   ```

4. Veja no material complementar disponibilizado no moodle mais detalhes sobre como fazer um arquivo `Makefile` quando os arquivos estão em diretórios diferentes. Abaixo, uma possibilidade para o nosso exemplo (supondo que os arquivos `.c` estão no diretório `src` e que o arquivo `.h` está no diretório `include`):

```
# Comentario do makefile
# utilizaremos a variavel CP para definir o compilador a ser utilizado
CP=gcc

# a variavel COPT sera utilizada para adicionar opcoes a compilacao
COPT= -c

INCLUDE=./include
SRC=./src
OBJ=./obj

programa: depPessoa
    $(CP) $(COPT) $(SRC)/principal.c -I$(INCLUDE) -o nomeExec
depPessoa:
    $(CP) $(COPT) $(SRC)/pessoa.c -I$(INCLUDE) -o $(OBJ)/pessoa.o
clean:
    rm -rf *o programa
```

Introdução a OO

Compare os arquivos anteriormente definidos com os abaixo (a discussão detalhada sobre os arquivos será feita em sala de aula):

```
class Pessoa
{
  private:
      int diaP;
      int mesP;
      int anoP;
      int idadeP;

  public:
     Pessoa(int diaNa, int mesNa, int anoNa);
     void Calc_Idade(int diaAT, int mesAT, int anoAT);
     int informaIdade();
};
```

Pessoa.h

```
#include "Pessoa.h"

Pessoa::Pessoa(int diaNa, int mesNa, int anoNa){
    diaP = diaNa;
    mesP = mesNa;
    anoP = anoNa;
}

void Pessoa::Calc_Idade(int diaAT, int mesAT, int anoAT){
   idadeP = anoAT - anoP;
   if (mesP <  mesAT)
      idadeP = idadeP - 1;
   else
       if (mesP == mesAT)
            if (diaP < diaAT)
                 idadeP = idadeP - 1;
}

int Pessoa::informaIdade(){
    return idadeP;
}
```

Pessoa.cpp

```
#include "Pessoa.h"
int main(int argc, char* argv[]){
    Pessoa Einstein(14, 3, 1879);
    Pessoa Newton(4, 1, 1643);

    Einstein.Calc_Idade(8, 2, 2007);
    Newton.Calc_Idade(8, 2, 2007);

    printf("Einstein teria %d \n", Einstein.informaIdade());
    printf("Newton teria %d   \n", Newton.informaIdade());

    return 0;
}
```

ProjetoOOExemplo.cpp