LICENCIATURA EM ENGENHARIA INFORMÁTICA

Princípios de Programação Procedimental

Manual do Programador

• U  C •

UNIVERSIDADE DE COIMBRA

André Martins Carrilho Costa Baptista **| 2012137523**
Guilherme dos Santos Simões Graça **| 2012138932**

1 - Introdução

O principal objectivo deste manual é descrever algumas funções que foram usadas no programa e clarificar a estrutura do mesmo. O programa permite uma gestão dinâmica e eficiente dos espaços de uma instituição, recorrendo a listas ligadas e a ficheiros. Durante a execução, os dados de cada reserva e pré-reserva são armazenados nas listas. Os ficheiros permitem que esses dados possam ser reutilizados cada vez que o programa é iniciado.

2 - Estrutura

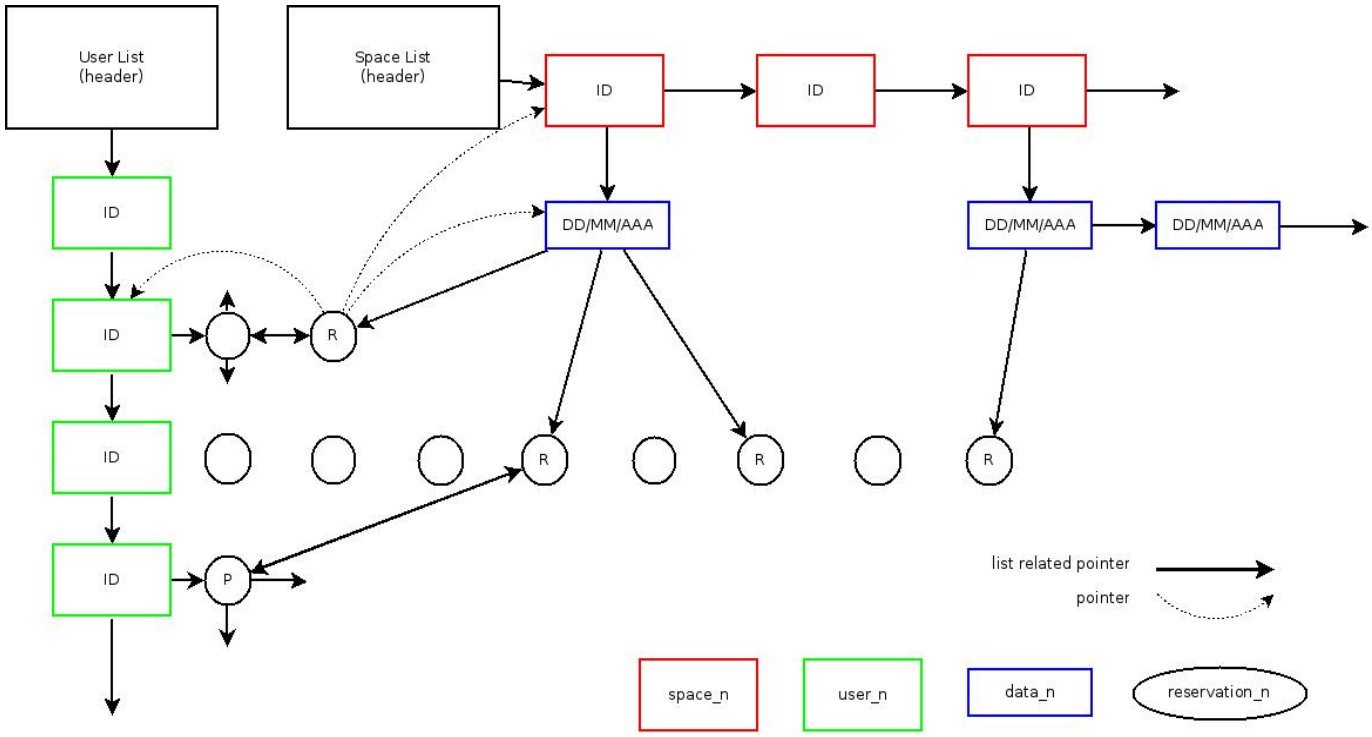
O programa é composto por um ficheiro C principal, "main.c" e por headers, que contêm funções que são utilizadas pelo ficheiro principal. Os headers estão dentro da pasta **src**.

Headers principais:

- → **headers.h** – Ficheiro que inclui todos os headers, hierarquicamente;
- → **files_read.h** – Funções para leitura e verificação de ficheiros e para carregar a informação para a memória, distribuindo os dados pelas estruturas;
- → **files_write.h** – Funções para escrita em ficheiros;
- → **inputs.h** – Funções que recebem dados do utilizador, de forma segura, e que garantem que os mesmos sejam válidos;
- → **highLevFunctions.h** – Ficheiro que contém funções que são acedidas através do menu principal, ou seja, funções de primeiro nível. Estas funções, para além de realizarem os principais procedimentos do programa, pedem dados ao utilizador e imprimem informações que dependem desses dados;
- → **lowLevFunctions.h** – Funções utilizadas pelas funções de primeiro nível, que carregam dados para as estruturas principais e verificam esses dados;
- → **structs.h** – Header com definições de estruturas;
- → **outros.h** – Contém essencialmente funções que auxiliam a verificação da validade dos dados introduzidos pelo utilizador e funções de sistema.
- → **CreationFunctions/** – Pasta que contém headers que permitem a criação de listas ligadas de vários tipos. Estas funções permitem também procurar elementos dentro de cada lista.

A pasta **data** contém os ficheiros onde são armazenados todos os dados, ao longo da execução do programa, de modo a que não haja perda de informação. Esta pasta contém dois ficheiros, **users.txt** e **spaces.txt**, e a pasta **spaces**. Nos dois últimos ficheiros são guardados todos os nomes de utilizador e nomes de espaços já criados. Dentro da pasta **spaces** são criadas pastas para cada um dos espaços, que contêm o ficheiro **dias.txt** e ficheiros individuais para cada uma das datas em que há reservas para aquele espaço.

As estruturas estão organizadas de acordo com o diagrama seguinte.



3 – Optimizações

3.1 – Multiplataforma

A aplicação é compatível com qualquer plataforma que tenha um compilador de C. Para o programa funcionar em Windows foi necessário acrescentar algumas verificações no ficheiro **headers.h**, para que fosse possível criar directórios através de comandos de sistema e limpar o ecrã:

```
/*headers.h*/
#ifdef WIN32
   #define sys 1
   #define slash "\\"
   #define C_NRM  ""
   #define C_RED  ""
   #define C_GRN  ""
   #define C_ORG  ""
#else
   #define sys 0
   #define slash "/"
   #define C_NRM  "\x1B[0m"
   #define C_RED  "\x1B[31m"
   #define C_GRN  "\x1B[32m"
   #define C_ORG
#endif
```

```
/*outros.h*/
void clear(){
   if (sys == 1) system("cls");
   else system("clear");
   printf("\n");
}
```

3.2 - Escrita de ficheiros de modo eficiente

As funções que permitem guardar os dados em ficheiros são eficientes, na medida em que abrem os mesmos com o modo "**r+**", ou seja, escrevem por cima dos dados já existentes, não sendo necessário voltar a abrir o ficheiro novamente para escrever, após a leitura. O ficheiro é lido para uma array de caracteres de tamanho **fsize(**ficheiro**) + 28*(strlen(**palavra que se vai adicionar**))**, o que diminui a utilização da memória.

Este modo foi utilizado nas funções que acrescentam dados aos ficheiros, porque nas funções que apagam informações é necessário voltar a abrir o ficheiro com o modo "**w**", visto que o tamanho do ficheiro tem que diminuir obrigatoriamente. Esta última situação é inevitável quando se cancela

3.3 – Algoritmo para selecção de blocos no horário

Para cada data, existem 28 blocos disponíveis para reserva, das 10h até às 24h, cada um com 30 minutos.

Foram implementados dois algoritmos simples, um para obter os índices do bloco inicial e final através da intervalo, e outro inverso para um intervalo através de um índice.

```
/*inputs.h*/
void timeInput(int array[ ], int actual){
   (…)
   array[0] = (time1.h-10) * 2;
   if(time1.m == 30) array[0] += 1;
   array[1] = (time2.h-10) * 2;
   if(time2.m == 30) array[1] += 1;
   array[1] -= 1;
}
```

```
/*outros.h*/
void printBlock(int n){
   int m = 0;
   if (n % 2 != 0)
      m = 3;
   n /= 2;
   n += 10;
   printf("%d:%d0",n,m);
}
```

3.4 – Utilizadores, espaços, reservas e pré-reservas ilimitados

Enquanto existir memória para armazenar os dados, é possível adicionar sempre mais um elemento a cada lista. Se não for possível alocar mais memória, o utilizador não consegue fazer mais reservas.

4 – Funções extra

4.1 – Verificação da data e horário

Foram implementadas funções que verificam se a data é válida. Estas verificações passam por verificar se a existência da mesma é possível, não esquecendo o facto de haver anos bissextos (Ver **outros.h**).

No momento da reserva, só é possível escolher o próprio dia ou datas futuras, bem como um horário que ainda não tenha sido ultrapassado, com uma tolerância de 15 minutos. Para estas funções foi utilizada a biblioteca <**time.h**>.

```
/*outros.h*/
int validTime(my_time input){
   time_t current = time(NULL);
   struct tm *t = localtime(&current);
   if (input.h > t->tm_hour)
      return 1;
   if (input.h == t->tm_hour)
      if (input.m >= (t->tm_min - 15)) return 1;
   return 0;
}
```

4.2 – Verificação de agenda

Caso o utilizador pretenda reservar um espaço e já tenha uma reserva para o mesmo horário mas para outro espaço, o programa informa que a reserva coincide com a agenda. Se a reserva for feita para o mesmo espaço e para o mesmo intervalo, o programa ignora automaticamente a reserva, porque já existe uma reserva nesse bloco de tempo

```
/*lowLevFunctions.h*/
int checkReservation(user u, space s, date d, int hora){
   int v = 0;
   reservation temp;
   temp = u->rl;
   while(1){
      if(temp == NULL) break;
      v = compareDatesAndHoras(d, hora, temp->date, temp->hora);
      if(v < 0) break;
      else if(v == 0){
         if(temp->space->id == s->id) return 3;
         else return 1;
      }
      temp = temp->next_u_r;
   }
   if(d->rl[hora] == NULL)  return 0;
   else return 2;
}
```

5 – Segurança

Quando é feita ou cancelada uma reserva ou pré-reserva, a informação é armazenada instantaneamente no ficheiro correspondente, o que evita a perda de informação, caso o programe termine de forma inesperada.

Todos os inputs estão protegidos, evitando quando possível função **scanf**, e limpando o **stdin** (Standard Input). Assim, foi implementada uma função, **stdinClear()**, que limpa esses caracteres indesejados.

```
/*inputs.h*/
void stdinClear(){
   char c;
   while ((c = getchar()) != '\n' && c != EOF);
}
```

Há também protecções indispensáveis ao nível da alocação de memória e de ponteiros para ficheiros.

```
/*Exemplo 1:*/
FILE *fp;
fp = fopen(str,"r");
if (fp == NULL) return;

/*Exemplo 2:*/
char *c = (char*)malloc(sizeof(char));
if (c == NULL) return;
```

Ao nível dos ficheiros, a função **check_files()** verifica se os ficheiros indispensáveis estão presentes na pasta **data**. Esta função evita que o programa tenha erros e termine de forma inesperada, caso esses ficheiros tenham sido apagados acidentalmente. A função também cria esses ficheiros e pastas na primeira execução do programa.

```
/*files_read.h:*/
void check_files(){
   /*verifica se data e data/spaces/ existem, caso contrario cria*/
   system("mkdir data");
   system("mkdir data" slash "spaces");
   FILE *p;
   p = fopen("data/users.txt","r");
   if (p == NULL)  p = fopen("data/users.txt","w");
   fclose(p);
   p = fopen("data/spaces.txt","r");
   if (p == NULL)  p = fopen("data/spaces.txt","w");
   fclose(p);
}
```