《程序设计课程设计》实验报告

实验名称《JOB SHOP 系统程序设计》概要设计＜α终极版＞

班　　级 _____

组　　号 _____

姓　　名 _____ 苇名一心 _____

任务概述

你是一间超级工厂的管理员（BOSS），每天都要在指定时间段内接受客户提交的 n 个产品（Job）$\{J_i\}_{i=1}^{n}$ 加工订单，每个产品都会指定加工步骤（Operation），必须按照 O_{i1}，O_{i2}，…，O_{in_i} 的顺序加工。其中 n_i 为产品 J_i 包含的加工步骤。每道工序（步骤）必须在指定的机器(Machine) $\{M_j\}_{j=1}^{m}$ 上加工，且每道工序的加工时间固定已知。你的任务是通过计算，尽可能使完成所有订单的总加工时间最短，并将你的加工计划展示出来。

完成这一任务必须满足以下约束条件：

约束 1: 程序启动后首先接受产品加工请求输入，确认当天的需求后不再接受新的订单。对每个产品需满足加工工序约束，即只有其前一个工序加工完毕，才能加工其后一个工序。每个工序的加工时间已知，且所需机器固定(每台机器都不能被其他机器替代)，均作为程序的输入。

举例说明，假设你收到 3 个产品的订单，需要在 3 台机器上加工，第 1 个产品有 3 个操作 O_{11}，O_{12}，O_{13}；第 2 个产品有 2 个操作 O_{21}，O_{22}；第 3 个产品有 2 个操作 O_{31}，O_{32}。每个产品的各工序所需机器和加工时间如表 1 所示。括号中的第一项表示工序的加工时间，第二项表示所需机器。例如，产品 1 的第一个工序必须在机器 M_1 上加工，加工时间为 7 个时间单元。表 1 为程序的输入。每个产品需满足加工工序约束。以第 2 个产品为例，只有当工序 O_{21} 加工完毕，才能开始加工 O_{22}。

表 1. 要输入的参数

	产品 1	产品 2	产品 3	...
操作 1	（7，M_1）	（10，M_2）	（7，M_1）	...
操作 2	（12，M_2）	（17，M_1）	（22，M_2）	...
操作 3	（15，M_3）			...
...				...

约束 2: 每台机器同一时间只能加工一个操作，一旦开始加工一个操作就要加工完成，期间不允许中断。

1．用户界面设计

1.1 用户界面

注：目前为命令行版本，暂无用户界面

1.2 操作元素和操作效果

暂无

3 高层数据结构设计

3.1 全局常量/变量定义

```
#pragma once
//////////////////head//////////////////////////
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
////////////////////////////////////////////////

//////////////////////macro/////////////////////////
#define LIMIT 300000/*程序运行的限制时间*/
////////////////////////////////////////////////

//////////////////////common/////////////////////////
struct operation/*一个操作结构体*/
{
	int flag;/*记录该操作是否已完成，‘1’代表未完成，‘0’代表已完成*/
	int shopnum;/*第几号工件*/
	int procedure;/*第几道工序*/
	int machine;/*使用的机器编号*/
	int start_time;/*开始时间*/
	int time;/*该操作的加工时间*/
	int end_time;/*结束时间*/
};
typedef struct operation OP;/*重命名*/

struct product/*一个产品结构体*/
{
	int flag;/*记录该产品是否已完成，‘1’代表未完成，‘0’代表已完成*/
	int sum_operation;/*该产品的总操作数*/
	OP operations[50];/*该产品的操作*/
};
typedef struct product product;/*重命名*/

int machine_num;/*机器数*/
int product_num;/*产品数*/
int operation_num;/*所有产品的总操作数*/
int optimal_num;/*最优操作解的下标*/
int conflict_product_subscript;/*记录冲突的操作结构体的最大下标*/
int max_time;/*所有产品都加工完成的总时间*/
int used_time;/*程序运行时间*/

product* product_ptr;/*产品结构体指针*/
```

```
product* original_ptr;/*存放原始产品结构体指针*/
product* completed_ptr;/*最终完成产品结构体指针*/
OP *active_product_ptr;/*可进行的操作结构体指针*/
OP *conflict_product_ptr;/*冲突的操作结构体指针*/
OP *completed_operations;/*输出时完成的操作*/
OP temp_completed_operations;/*临时变量*/
////////////////////////////////////////////////////////////////
```

3.2 模块常量与变量定义

①input.c：

```
int i,n = 0, j = 0;/*n 为产品的产品序号，以‘-1’结束，j 为操作的计数器*/
//char flag = '0';/*记录该行是否已输入完成，完成的标志是‘\n’*/
```

②create.c：

```
int i;/*计数器*/

int i;/*计数器和计录上一次可以的操作在结构体数组中的位置*/

int i;/*计数器*/

int i/*计数器*/,
j/*计数器和计录上一次冲突的操作的结构体数组的下标*/,
min_time = active_product_ptr->end_time/*操作能最先完成的时间*/,
machine = active_product_ptr->machine;/*能最先完成的操作的机器*/
```

③refresh.c

```
int i/*计数器*/ ,
j = conflict_product_subscript/*冲突操作集最大下标*/;
int time = (conflict_product_ptr + optimal_num)->time,shopnum =
(conflict_product_ptr+ optimal_num)->shopnum,
procedure=(conflict_product_ptr+optimal_num)->procedure,end_time=(conflict_pr
oduct_ptr + optimal_num)->end_time;/*最优解的时间，产品号，操作步骤号，结束时
间*/
```

④output.c

```
int i, j, k, m, min_start_time;
```

⑤main.c

```
int k , i, n=27, m, j, algorithm_time;
time_t start_t, end_t;
used_time = 0;
max_time = 0;
```

⑥schedule1.c

```
int i, min_time = conflict_product_ptr->time, min_time_optimal_num = 0;
……
```

⑦schedule2.c

```
int NINQ_optimal_num = 0, sum_num = 0, min_num = 0, machine = 0, i, j, k = 1;
```

……

⑧schedule3.c

int i,k,temptime=0,mintime,optimal_num;

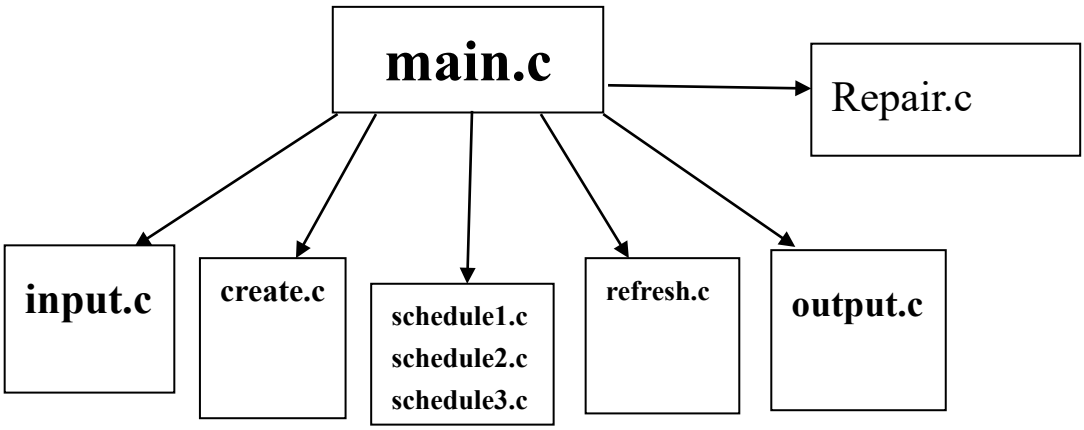
……

⑨repair.c

修理模块，尚未完成

4 系统模块划分

4.1 系统模块结构图



1、input.c

完成对文件或键盘输入的订单数据获取，保存到公共变量中。

2、create.c

创建可进行操作集，冲突操作集。

3、schedule1.c、schedule2.c、schedule3.c

寻找最优解。

4、refresh.c

刷新可进行操作集，冲突操作集，已完成产品操作集。

5、output.c

完成输出。

6、Repair.c

修理机器模块，尚未完成。

4.2 各模块函数说明

```
#pragma once
//////////////////head//////////////////////////
#include"main.h"
////////////////////////////////////////////////

//////////////////input.c/////////////////
void input();/*输入函数的原型*/
//////////////////////////////////////////
```

```
#pragma once
//////////////////head//////////////////////////
#include"main.h"
////////////////////////////////////////////////

//////////////////////create.c///////////////////////
void initialize_product();/*用于初始化产品结构体数组*/
void initialize_completed_product();/*用于初始化最终完成的产品结构体数组*/

OP *initialize_active_product();/*初始化可进行的操作集函数的原型，作用：分配空间并把所有记录了该操作是否已删除的‘flag’重置为‘1’(未删除)*/
void create_active_product(product *product_ptr, OP *active_product_ptr);/*首次创建可进行的操作集函数的原型，作用：把所有产品的首次操作加入可进行的操作集*/

OP *initialize_conflict_product();/*初始化冲突的操作集函数的原型，作用：分配空间*/
void create_conflict_product(OP *active_product_ptr, OP *conflict_product_ptr, int *conflict_product_subscript_address);
/*创建每个冲突的操作集函数的原型，作用：把所有可操作集中与结束时间最短的操作机器冲突的操作取出和它本身一起组成冲突操作集，并更改该冲突操作集的最大下标*/
////////////////////////////////////////////////////
```

```
#pragma once
//////////////////////////////////////////
//////////////////head//////////////////////////
#include"main.h"
////////////////////////////////////////////////

//////////////////////refresh.c///////////////////////
void refresh_active_product(product *product_ptr, OP* conflict_product_ptr, OP
```

```
*active_product_ptr, int optimal_num);
/*刷新可进行的操作集函数的原型，作用：更改未被选中的操作的起始时间，删除已完成的操作并(如果该产品还有下个操作的话)将该产品的下个操作加入该集*/
int refresh_product(product *product_ptr, int optimal_num, OP *conflict_product_ptr, OP *active_product_ptr);
/*刷新已完成的操作集函数的原型，作用：在所有冲突集中以一定规则找到最优先的操作取出放入已完成操作操作集，并修改开始和结束时间*/
//////////////////////////////////////////////////


#pragma once
//////////////////head//////////////////////////
#include"main.h"
//////////////////////////////////////////////////

//////////////////////output.c//////////////////////
void output(product *completed_ptr);
//////////////////////////////////////////////////


#pragma once
//////////////////head//////////////////////////
#include"main.h"
//////////////////////////////////////////////////

//////////////////schedule.c//////////////////////
int end_time_min_rule();
//////////////////////////////////////////////////
int end_time_min_rule1();
//////////////////////////////////////////////////
int time_min_rule();
//////////////////////////////////////////////////
int time_min_rule1();
//////////////////////////////////////////////////
int time_max_rule();
//////////////////////////////////////////////////
int time_max_rule2();
//////////////////////////////////////////////////
int least_work_remaining();
//////////////////////////////////////////////////
int least_work_remaining2();
//////////////////////////////////////////////////
int most_work_remaining();
//////////////////////////////////////////////////
```

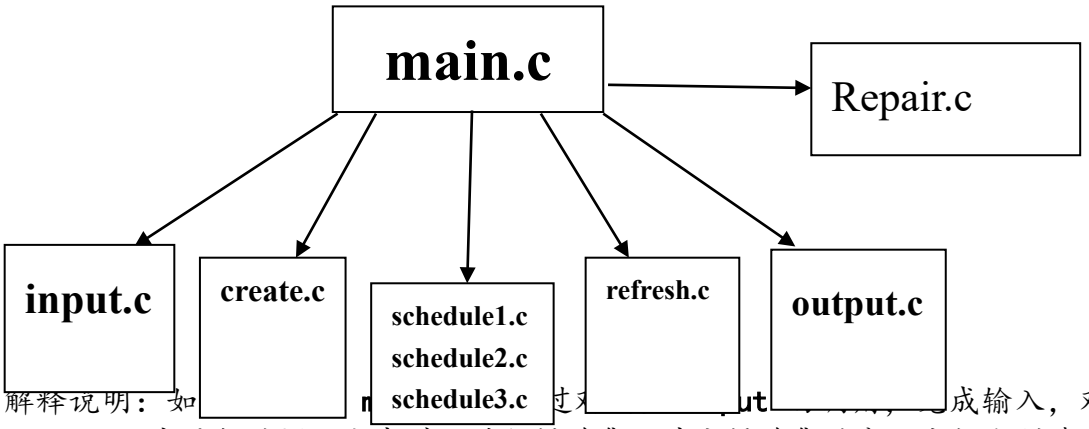
```
int most_work_remaining2();
////////////////////////////////////////////
int fewest_number_of_operations_remaining();
////////////////////////////////////////////
int fewest_number_of_operations_remaining2();
////////////////////////////////////////////
int most_number_of_operations_remaining();
////////////////////////////////////////////
int most_number_of_operations_remaining2();
////////////////////////////////////////////
```

```
#pragma once
///////////////schedule2.c/////////////////
int RANDOM_rule();
////////////////////////////////////////
int NINQ_rule();
////////////////////////////////////////
int NINQ_rule1();
////////////////////////////////////////
int NINQ_rule2();
////////////////////////////////////////
int NINQ_rule3();
////////////////////////////////////////
int WINQ_rule();
////////////////////////////////////////
int WINQ_rule1();
////////////////////////////////////////
int WINQ_rule2();
////////////////////////////////////////
int WINQ_rule3();
////////////////////////////////////////
```

```
#pragma once
///////////////schedule3.c/////////////////
int least_time();
////////////////////////////////////////
int most_time();
////////////////////////////////////////
int least_operation();
////////////////////////////////////////
int most_operation();
////////////////////////////////////////
```

```
int FCFS();
////////////////////////////////////////
int RANDOM_rule1();
////////////////////////////////////////
```

4.3 函数调用图示及说明



解释说明：如 m 过对 ut 成输入，对 create.c 中函数的调用完成对可进行操作集，冲突操作集的分配空间和创建，对 schedule1.c、schedule2.c、schedule3.c 中函数的调用完成最优解的寻找，对 refresh.c 中函数的调用完成对可进行操作集和完成产品操作集的刷新和重创建，对函数 output 的调用完成输出。

5 高层算法设计

- 使用 Giffler & Thompson 算法，
- while（可调度操作集不为空）
  - 选择可进行操作集中完成时间最小的操作 x，取出所用机器 m；
  - 遍历可进行操作集找到每个在机器 m 上加工的操作 i，将其加入到冲突操作集中，并按优先规则计算其优先级 *h*。
  - 从冲突操作集中选出一个优先级最高的操作 y，加入已完成操作中，由此得到新已完成操作。
  - 把选中的操作 y 从可进行操作集中删除，同时把该操作的直接后继操作加入可进行操作集，更新每个操作的开

始时间和结束时间。

教师评语：