实验报告：银行业务管理系统

题目：模拟银行业务管理的离散事件

一、需求分析

1. 通过给定的范围，随机生成银行每天的事件表，即客户数据，包括到达时间、停留时间、金额。
2. 模拟银行的业务流程，采用队列模拟客户排队。队列分为普通队列和等待队列，以模拟银行资金总额充足与不足时的等待情况。我们需要算出每个客户的离开时间。
3. 当银行资金总额不足以处理普通队列的客户时，将客户加入等待队列中；当银行资金总额充足时，优先检查并处理等待队列中的客户。
4. 选做一要求使用循环队列作为银行中两个排队队列的实现形式，要求循环利用内存空间，且在内存空间不足时可以实现扩容。
5. 选做二要求维护多天的运行模拟，要求后一天的事件尽可能填充前一天未完成事件表中的“碎片”，以学习类似操作系统中的内存管理方式。

二、概要设计

拟采用循环队列的数据结构，编写一个模拟函数对银行事件进行模拟，并编写一系列函数来维护事件表插入的正确性。程序中将涉及下列两个抽象数据类型：

1. 定义“客户”类型 `CustNode`：

```
typedef struct CustNode
{
    // 数据对象：
    TimeType arrtime, durtime, leavetime; // 客户的到达时间、业务处理时间、离开时间
    AmountType amount;                    // 客户的业务金额
    struct CustNode* next;                //  （选做二）指向下一个未完成客户的指针
};
```

2. 定义循环队列 `CyQueue`：

```
typedef struct CyQueue
{
    // 数据对象：
    SizeType maxsize;       // 队列当前的最大容量
    IndexType head, tail;   // 队首和队尾的下标
    int full_flag;          // 队列是否已满的标志
    DataType* data;         // 队列的数据数组
```

```
    // 基本操作:
    CyQueue* CyQueue_Init();
      // 操作结果: 构造一个空队列
    void CyQueue_Delete(CyQueue* q);
      // 初始条件: 队列 q 存在
      // 操作结果: 销毁队列 q
    void CyQueue_Clear(CyQueue* q);
      // 操作结果: 将队列 q 清空
    SizeType CyQueue_Size(CyQueue* q);
      // 操作结果: 返回队列 q 的元素个数
    int CyQueue_Empty(CyQueue* q);
      // 操作结果: 若队列 q 为空则返回 1，否则返回 0
    DataType CyQueue_At(CyQueue* q, IndexType ind);
      // 操作结果: 返回队列 q 中（从队首数起）下标为 ind 的元素
    void CyQueue_Resize(CyQueue* q, SizeType newsize);
      // 操作结果: 将队列 q 的容量调整为 newsize
    void CyQueue_Push(CyQueue* q, DataType data);
      // 操作结果: 将元素 data 入队，若容量不足则自动扩容
    DataType CyQueue_Pop(CyQueue* q);
      // 操作结果: 将队首元素出队并返回
    DataType CyQueue_Front(CyQueue* q);
      // 操作结果: 返回队首元素
};
```

3. 维护事件列表 `eventlist`

```
// 数据对象:
CustNode eventlist[MAX];          // 事件列表
CustNode* eventlist_head = NULL;  // 第一个未处理事件的指针
CustNode* eventlist_tail = NULL;  // 最后一个未处理事件的指针
// 基本操作:
void eventlist_Insert(CustNode node);
  // 初始条件: eventlist_tail 已经被正确初始化
  // 操作结果: 将客户 node 插入事件列表 eventlist
void eventlist_Generate();
  // 操作结果: 随机生成银行每天的事件表，并直接存入 eventlist
void eventlist_output(int print_avg_stay);
  // 操作结果: 输出事件列表，若 print_avg_stay 为真则输出平均逗留时间
```

4. 本程序包含三个模块:

    1. 主程序模块。`main()` 函数负责处理命令行输入和调用 `process()` 函数；`process()` 函数负责循环模拟每天的银行业务，更新当天的事件表，调用 `simulate()` 函数以模拟当天的

事件，输出当天的事件表，并整理当天未完成的事件以供下一天使用。`simulate()` 函数负责模拟单日的事件，并返回未完成的事件队列，待处理函数使用。

`process_unfinished(CyQueue *q)` 函数接收未完成的事件队列，并将这些事件在事件表中妥善整理，供下一天使用。

2. 事件列表模块——生成、维护每天的事件列表。
3. 循环队列模块——实现银行的两个排队队列。

除了三个主要模块外，还有部分辅助函数用于生成随机数等功能。

三、详细设计

1. 主程序中需要的全程量

```
#define MAX 200010                          // 事件表的最大容量

// 定义时间、金额、下标、大小等数据类型
typedef int TimeType;
typedef int AmountType;
typedef unsigned int IndexType;
typedef unsigned int SizeType;

// 待输命令行入的全局变量
SizeType CHUNKSIZE = 4;                     // 事件表每次扩容的大小
AmountType total_initial;                   // 银行的初始资金总额
TimeType closetime;                         // 银行的营业结束时间
TimeType  durtime_min,  durtime_max,        // 客户的最短和最长业务处理时间
          interval_min, interval_max;       // 客户的最短和最长到达时间间隔
AmountType amount_min,   amount_max;        // 客户的最小和最大业务金额
SizeType day_number;                        // 模拟的天数

// 模拟过程中维护的全局变量
CustNode eventlist[MAX];                    // 事件列表
AmountType total_amount = 0;                // 银行当前的资金总额
```

2. 循环队列的实现

```
CyQueue* CyQueue_Init()
{
  SizeType size = CHUNKSIZE;
  CyQueue* q = (CyQueue*)malloc(sizeof(CyQueue));
  q->maxsize = size;
  q->head = 0;
  q->tail = 0;
  q->full_flag = 0;
```

```
  q->data = (DataType*)malloc(size * sizeof(DataType));
}

void CyQueue_Delete(CyQueue* q)
{
  free(q->data);
  free(q);
}

void CyQueue_Clear(CyQueue* q)
{
  q->head = 0;
  q->tail = 0;
  q->full_flag = 0;
}

SizeType CyQueue_Size(CyQueue* q)
{
  if(q->tail == q->head && q->full_flag)
    return q->maxsize;
  // (int - int + uint) % uint != (int + uint - int) % uint
  // evil data type bug
  return ((int)q->tail + (int)q->maxsize - (int)q->head) % (int)q->maxsize;
}

int CyQueue_Empty(CyQueue* q)
{
  return q->head == q->tail && !q->full_flag;
}

DataType CyQueue_At(CyQueue* q, IndexType ind)
{
  assert(ind < CyQueue_Size(q));
  return q->data[(q->head + ind) % q->maxsize];
}

void CyQueue_Resize(CyQueue* q, SizeType newsize)
{
  // The overflowed data will be deleted.
  DataType* newdata = (DataType*)malloc(newsize * sizeof(DataType));
  SizeType size = CyQueue_Size(q);
```

```
  SizeType copy_size = newsize < size ? newsize : size;
  for(IndexType i = 0; i < copy_size; i++)
    newdata[i] = CyQueue_At(q, i);
  q->head = 0;
  q->maxsize = newsize;
  if(copy_size == newsize)
  {
    // Queue Full
    q->tail = 0;
    q->full_flag = 1;
  }
  else
  {
    // Queue Not Full
    q->tail = copy_size;
    q->full_flag = 0;
  }
  free(q->data);
  q->data = newdata;
}

void CyQueue_Push(CyQueue* q, DataType data)
{
  SizeType size = CyQueue_Size(q);
  if(size == q->maxsize)
  {
    // Resize First
    size += CHUNKSIZE;
    CyQueue_Resize(q, size);
  }
  q->data[q->tail] = data;
  q->tail++;
  q->tail %= q->maxsize;
  if(q->tail == q->head) q->full_flag = 1;
}

DataType CyQueue_Pop(CyQueue* q)
{
  assert(!CyQueue_Empty(q));
  DataType data = q->data[q->head];
  if(q->tail == q->head) q->full_flag = 0;
```

```
  q->head++;
  q->head %= q->maxsize;
  return data;
}

DataType CyQueue_Front(CyQueue* q)
{
  assert(!CyQueue_Empty(q));
  return q->data[q->head];
}
```

3. 每天事件的模拟函数如下：

```
CyQueue* simulate()
{
  CyQueue *qmain = CyQueue_Init(),
      *qwait = CyQueue_Init();

  for(CustNode* i = eventlist_head; i != NULL; i = i->next)
    CyQueue_Push(qmain, i);

  TimeType current_time = 0;
  while(!CyQueue_Empty(qmain))
  {
    CustNode *current = CyQueue_Pop(qmain);
    if(current->arrtime > current_time)
      current_time = current->arrtime;
    if(current_time >= closetime)
    {
      CyQueue_Push(qmain, current);
      break;
    }
    if(current->amount < 0 && total_amount + current->amount < 0)
    {
      // 银行存款不足，排入 wait 队列
      CyQueue_Push(qwait, current);
      ECHO_FAIL;
      continue;
    }
    // 正常办理业务
    AmountType prev_amount = total_amount;
    current_time += current->durtime;
    if(current_time >= closetime)
```

```
{
  CyQueue_Push(qmain, current);
  break;
}
total_amount += current->amount;
current->leavetime = current_time;

ECHO_EVENT;

if(current->amount > 0)
{
  // 处理 wait 队列
  printf("[Process the wait queue]\n");
  SizeType wait_len = CyQueue_Size(qwait);
  int break_flag = 0;
  for(IndexType i = 0; i < wait_len; i++)
  {
    // 无需判断队空，因为至多处理 wait_len 个元素
    current = CyQueue_Pop(qwait);
    if(total_amount + current->amount >= 0)
    {
      current_time += current->durtime;
      if(current_time >= closetime)
      {
        CyQueue_Push(qwait, current);
        break_flag = 1;
        break;
      }
      total_amount += current->amount;
      current->leavetime = current_time;
      ECHO_EVENT;
      if(total_amount <= prev_amount)
      {
        // 不可能再有满足者
        break;
      }
    }
    else
    {
      CyQueue_Push(qwait, current);
      ECHO_FAIL;
```

```
        }
      }
      if(break_flag) break;
      printf("[Process the main queue]\n");
    }
  }
  // 银行营业结束后，所有客户立即离开银行
  CyQueue *qunfinished = CyQueue_Init();
  while(!CyQueue_Empty(qmain))
  {
    CustNode *current = CyQueue_Pop(qmain);
    current->leavetime = closetime;
    CyQueue_Push(qunfinished, current);
  }
  while(!CyQueue_Empty(qwait))
  {
    CustNode *current = CyQueue_Pop(qwait);
    current->leavetime = closetime;
    CyQueue_Push(qunfinished, current);
  }

  CyQueue_Delete(qmain);
  CyQueue_Delete(qwait);
  return qunfinished;
}
```

其中 ECHO_EVENT 和 ECHO_FAIL 为两个输出中间数据的宏。

```
#define ECHO_EVENT \
  printf("Process Event #%d (%+d):\nArrive Time:%d\nTime:%d -> %d\nTotal Amount:%d
-> %d\nMain Queue: ", (int)(current - eventlist), current->amount, current-
>arrtime, current_time - current->durtime, current_time, total_amount - current-
>amount, total_amount);\
  SizeType __qmain_size = CyQueue_Size(qmain), __qwait_size = CyQueue_Size(qwait);
\
  if (__qmain_size == 0) printf("<EMPTY>");\
  for(IndexType __i = 0; __i < __qmain_size; __i++)\
    printf("%d ", (int)(CyQueue_At(qmain, __i) - eventlist));\
  printf("\nWait Queue: ");\
  if (__qwait_size == 0) printf("<EMPTY>");\
  for(IndexType __i = 0; __i < __qwait_size; __i++)\
    printf("%d ", (int)(CyQueue_At(qwait, __i) - eventlist));\
  printf("\n\n");
```

```
#define ECHO_FAIL \
  printf("Process Event #%d (%+d) but Fail (there's only %d)\nMain Queue: ", (int)
(current - eventlist), current->amount, total_amount); \
  SizeType __qmain_size = CyQueue_Size(qmain), __qwait_size = CyQueue_Size(qwait);
\
  if (__qmain_size == 0) printf("<EMPTY>");\
  for(IndexType __i = 0; __i < __qmain_size; __i++) \
    printf("%d ", (int)(CyQueue_At(qmain, __i) - eventlist)); \
  printf("\nWait Queue: "); \
  if (__qwait_size == 0) printf("<EMPTY>");\
  for(IndexType __i = 0; __i < __qwait_size; __i++) \
  printf("%d ", (int)(CyQueue_At(qwait, __i) - eventlist)); \
  printf("\n\n");
```

4. 维护多日事件的函数实现:

```
// INITIALIZE eventlist_tail BEFORE CALLING
void eventlist_Insert(CustNode node)
{
  if(eventlist_tail == NULL)
  {
    eventlist[0] = node;
    eventlist_head = &eventlist[0];
    eventlist_tail = &eventlist[0];
    eventlist_tail->next = NULL;
  }
  else if(eventlist_tail->arrtime <= node.arrtime)
  {
    // 贪心尾插优先
    assert((eventlist_tail - eventlist + 1) < MAX);
    eventlist_tail->next = eventlist_tail + 1;
    eventlist_tail += 1;
    (*eventlist_tail) = node;
    eventlist_tail->next = NULL;
  }
  else if(eventlist_head->arrtime >= node.arrtime && (int)(eventlist_head -
eventlist) > 0)
  {
    // 次之先填充头部
    eventlist[0] = node;
    eventlist[0].next = eventlist_head;
    eventlist_head = &eventlist[0];
  }
```

```
else
{
  CustNode* prev = NULL;
  for(CustNode* cur = eventlist_head; cur != NULL; cur = cur->next)
  {
    if(cur->arrtime > node.arrtime)
    {
      // 插入 cur 前
      if(prev != NULL && (int)(cur - prev - 1) > 0)
      {
        IndexType idx = prev + 1 - eventlist;
        eventlist[idx] = node;
        eventlist[idx].next = cur;
        if(cur == eventlist_head)
          eventlist_head = &eventlist[idx];
        else prev->next = &eventlist[idx];
        break;
      }
      else
      {
        // 把事件表中 cur 往后的元素都后移一位，直到遇到空闲空间
        CustNode* nearest_idle = cur;
        while(nearest_idle->next == nearest_idle + 1)
          nearest_idle = nearest_idle->next;
        nearest_idle += 1;  // 得到最近的空闲位置
        assert(nearest_idle - eventlist <= MAX);
        // 将 [cur, nearest_idle) 整体右移一位
        // 将 nearest_idle 移到 cur 的位置
        if((int)(eventlist_tail - nearest_idle + 1) <= 0)
          eventlist_tail += 1;
        while((int)(nearest_idle - cur) > 0)
        {
          *nearest_idle = *(nearest_idle - 1); // Copy

          // 如果被移动的 prev 也在这段连续空间里，那么需要移动 next
          // 但如果不在的话（也就是 prev 变量了），那么它对应的 next 值本来就会被填充，
          // 也就不需要改 next
          if((int)(nearest_idle - 2 - cur) >= 0)
            (nearest_idle - 2)->next += 1;
          nearest_idle -= 1;
        }
```

```
            (*cur) = node;
            cur->next = cur + 1;
            break;
          }
        }
        prev = cur;
      }
    }
}

void eventlist_Generate()
{
  eventlist_tail = eventlist_head;
  if(eventlist_tail != NULL)
  {
    while(eventlist_tail->next != NULL)
      eventlist_tail = eventlist_tail->next;
  }

  TimeType arrtime = 0;
  while(arrtime < closetime)
  {
    CustNode node;
    node.next = NULL;
    node.arrtime = arrtime;
    node.durtime = random_range(durtime_min, durtime_max + 1);
    node.amount = random_range(amount_min, amount_max + 1);
    node.leavetime = -1;

    printf("Random event: \tArrtime:%d \tDurtime:%d \tAmount:%d\n", node.arrtime,
node.durtime, node.amount);
    eventlist_Insert(node);

    TimeType interval = random_range(interval_min, interval_max + 1);
    arrtime += interval;
  }
}

void process_unfinished(CyQueue *q)
{
  // 使用选择排序处理 next 表
```

```
  // O(n^2)
  eventlist_head = NULL;
  while(!CyQueue_Empty(q))
  {
    CustNode *node_max = NULL;
    SizeType size = CyQueue_Size(q);
    // 每次选择一个编号最大的（当然是到达最晚的）
    // 这里之前因为优先选择到达最晚的，导致相同到达时间的会出现
    // 编号偏后的 next 指向编号偏前的值，从而使得移位出错，造成死循环。
    // 这个 bug 调了非常久。惨痛的代价!
    for(IndexType i = 0; i < size; i++)
    {
      CustNode *current = CyQueue_Pop(q);
      if(node_max == NULL) node_max = current;
      else if((int)(current - eventlist) > (int)(node_max - eventlist))
      {
        CyQueue_Push(q, node_max);
        node_max = current;
      }
      else
      {
        CyQueue_Push(q, current);
      }
    }
    node_max->next = eventlist_head;
    eventlist_head = node_max;

    node_max->leavetime = -1;
  }
  printf("Unfinished events: ");
  for(CustNode* i = eventlist_head; i != NULL; i = i->next)
    printf("%d ", (int)(i - eventlist));
  printf("\n\n");
  CyQueue_Delete(q);
}

void process()
{
  total_amount = total_initial;
  for(SizeType i = 1; i <= day_number; i++)
  {
```

```
            printf("--- Day #%u ---\n", i);
            eventlist_Generate();
            eventlist_output(0);
            CyQueue *qunfinished = simulate();
            eventlist_output(1);

            process_unfinished(qunfinished);
        }
    }
```

四、调试分析

1. 在本程序调试过程中，遇到了一个在不同机器上编译得到了不同结果的问题。问题出在 `CyQueue_Size` 函数中混用有符号类型和无符号类型加减，出现了未定义行为。最后，本问题是通过在运算过程中将数据类型全部转为 `int` 解决的。
2. 从本实验的编写过程中可以看出，线性表的应用广泛。本题中设计了循环队列和事件表两种事件表，充分利用了线性表的特性，使得程序的编写更加简洁。
3. 循环链表算法的时空分析：

   假设循环链表的长度为 n，则循环链表的空间复杂度为 $O(n)$。

   (1) 扩容操作（`CyQueue_Resize`）的复杂度为 $O(n)$，因为扩容操作需要将原先的 n 个数据全部复制到新的内存空间中。因此，内存已经占满情况下的入队操作（`CyQueue_Push`）的复杂度也 $O(n)$。

   (2) 内存未占满时的入队操作的复杂度为 $O(1)$，出队操作（`CyQueue_Pop`）的复杂度为 $O(1)$，因为入出队操作只需要修改队首指针即可。访问操作（`CyQueue_At` 和 `CyQueue_Front`）的复杂度为 $O(1)$，因为只需要根据下标计算出对应的元素即可。

   (3) 队列的大小操作（`CyQueue_Size`）的复杂度为 $O(1)$，因为只需要计算队首和队尾的下标之差即可。
4. 事件表的插入操作的复杂度为 $O(n)$，因为在插入操作中需要遍历链表找到合适的插入位置，且有可能需要移动已经存在于事件表中的元素。事件表的生成操作的复杂度为 $O(n^2)$，因为每生成一次事件都需要插入一次。
5. 事件表中维护 `next` 指针的方法能让插入操作移动已有碎片元素数的最小化，因此在内存管理等应用中也有类似的实现。

五、用户手册

1. 本程序在 Windows 11, Ubuntu 22.04, Alpine Linux v3.1.4 等操作系统上均通过了测试。代码源文件为 `main.c`，编译命令为 `gcc -o main main.c`

2. 进入程序后会显示一系列提示信息，以输入题目中所需要的运行参数。输入完毕后程序会自动运行，输出每天的事件表和平均逗留时间。一个可能的输入示例如下:

```
Please input the total amount:
10000
Please input the close time:
600
Please input the min&max of durtime:
5 40
Please input the min&max of amount:
-20000 16000
Please input the min&max of interval:
5 40
Please input the CHUNKSIZE:
20
Please input the number days:
4
```

3. 输入结束后，程序输出一行 `=== START SIMULATION ===`，并开始模拟。

- 首先，程序会随机生成一系列事件，并输出未确定客户离开时间的，当天的事件表;

```
--- Day #1 ---
Random event:   Arrtime:0   Durtime:15  Amount:-10018
(...)
Random event:   Arrtime:578   Durtime:12  Amount:15444
Event | Arrtime   | Durtime   | Amount  | Leavetime
0     |        0  |       15  |   -10018  | -1
(...)
23    |      578  |       12  |   +15444  | -1
```

- 随后，程序会以模拟的时间顺序，对每一次成功或失败的客户时间信息进行输出，并输出每次时间发生时银行的两个队列;

```
Process Event #0 (-10018) but Fail (there's only 10000)
Main Queue: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Wait Queue: 0

Process Event #1 (-15476) but Fail (there's only 10000)
Main Queue: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Wait Queue: 0 1

Process Event #2 (-4058):
Arrive Time:63
Time:63 -> 85
Total Amount:10000 -> 5942
```

```
Main Queue: 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Wait Queue: 0 1

(...)

Process Event #8 (+9755):
Arrive Time:267
Time:267 -> 293
Total Amount:2706 -> 12461
Main Queue: 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Wait Queue: 0 1 3 5 6 7

[Process the wait queue]
Process Event #0 (-10018):
Arrive Time:0
Time:293 -> 308
Total Amount:12461 -> 2443
Main Queue: 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Wait Queue: 1 3 5 6 7

[Process the main queue]
Process Event #9 (+9361):
Arrive Time:292
Time:308 -> 336
Total Amount:2443 -> 11804
Main Queue: 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Wait Queue: 1 3 5 6 7

(...)

[Process the wait queue]
Process Event #5 (-12217):
Arrive Time:169
Time:575 -> 591
Total Amount:12589 -> 372
Main Queue: 18 19 20 21 22 23
Wait Queue: 6 12

[Process the main queue]
```

- 最后，程序会输出填好离开时间的事件表、当天的平均逗留时间及未完成的事件。

```
Event | Arrtime   | Durtime   | Amount  | Leavetime
0     |        0  |       15  |   -10018  | 308
1     |       37  |        8  |   -15476  | 505
2     |       63  |       22  |    -4058  | 85
(...)
22    |      545  |       24  |    +1452  | 600
23    |      578  |       12  |   +15444  | 600

Average stay time: 154.833333 (min).
Unfinished events: 6 12 18 19 20 21 22 23
```

- 在当天的模拟结束后，程序会自动开始下一天的模拟，并输出信息，直到模拟结束。

六、测试结果

对于可能的输入数据：

```
10000
600
10 40
-20000 16000
60 90
20
2
```

一组可能的输出结果如下：

```
Please input the total amount:
Please input the close time:
Please input the min&max of durtime:
Please input the min&max of amount:
Please input the min&max of interval:
Please input the CHUNKSIZE:
Please input the number days:

=== START SIMULATION ===
--- Day #1 ---
Random event:   Arrtime:0   Durtime:11  Amount:44
Random event:   Arrtime:83  Durtime:33  Amount:-851
Random event:   Arrtime:158   Durtime:36  Amount:-17526
Random event:   Arrtime:241   Durtime:10  Amount:-15728
Random event:   Arrtime:323   Durtime:24  Amount:6886
Random event:   Arrtime:405   Durtime:21  Amount:12253
Random event:   Arrtime:495   Durtime:35  Amount:-12909
```

```
Random event:   Arrtime:580   Durtime:10  Amount:-19583
Event   | Arrtime   | Durtime   | Amount  | Leavetime
   0  |        0  |       11  |      +44  | -1
   1  |       83  |       33  |     -851  | -1
   2  |      158  |       36  |   -17526  | -1
   3  |      241  |       10  |   -15728  | -1
   4  |      323  |       24  |    +6886  | -1
   5  |      405  |       21  |   +12253  | -1
   6  |      495  |       35  |   -12909  | -1
   7  |      580  |       10  |   -19583  | -1

Process Event #0 (+44):
Arrive Time:0
Time:0 -> 11
Total Amount:10000 -> 10044
Main Queue: 1 2 3 4 5 6 7
Wait Queue: <EMPTY>

[Process the wait queue]
[Process the main queue]
Process Event #1 (-851):
Arrive Time:83
Time:83 -> 116
Total Amount:10044 -> 9193
Main Queue: 2 3 4 5 6 7
Wait Queue: <EMPTY>

Process Event #2 (-17526) but Fail (there's only 9193)
Main Queue: 3 4 5 6 7
Wait Queue: 2

Process Event #3 (-15728) but Fail (there's only 9193)
Main Queue: 4 5 6 7
Wait Queue: 2 3

Process Event #4 (+6886):
Arrive Time:323
Time:323 -> 347
Total Amount:9193 -> 16079
Main Queue: 5 6 7
Wait Queue: 2 3
```

```
[Process the wait queue]
Process Event #2 (-17526) but Fail (there's only 16079)
Main Queue: 5 6 7
Wait Queue: 3 2

Process Event #3 (-15728):
Arrive Time:241
Time:347 -> 357
Total Amount:16079 -> 351
Main Queue: 5 6 7
Wait Queue: 2

[Process the main queue]
Process Event #5 (+12253):
Arrive Time:405
Time:405 -> 426
Total Amount:351 -> 12604
Main Queue: 6 7
Wait Queue: 2

[Process the wait queue]
Process Event #2 (-17526) but Fail (there's only 12604)
Main Queue: 6 7
Wait Queue: 2

[Process the main queue]
Process Event #6 (-12909) but Fail (there's only 12604)
Main Queue: 7
Wait Queue: 2 6

Process Event #7 (-19583) but Fail (there's only 12604)
Main Queue: <EMPTY>
Wait Queue: 2 6 7

Event   | Arrtime   | Durtime   | Amount  | Leavetime
   0  |        0  |       11  |      +44  | 11
   1  |       83  |       33  |     -851  | 116
   2  |      158  |       36  |   -17526  | 600
   3  |      241  |       10  |   -15728  | 357
   4  |      323  |       24  |    +6886  | 347
```

```
   5  |      405  |       21  |   +12253  | 426
   6  |      495  |       35  |   -12909  | 600
   7  |      580  |       10  |   -19583  | 600

Average stay time: 96.500000 (min).
Unfinished events: 2 6 7

--- Day #2 ---
Random event:   Arrtime:0   Durtime:37  Amount:-17188
Random event:   Arrtime:77  Durtime:38  Amount:-18238
Random event:   Arrtime:152   Durtime:11  Amount:10162
Random event:   Arrtime:232   Durtime:40  Amount:-6056
Random event:   Arrtime:322   Durtime:15  Amount:4177
Random event:   Arrtime:384   Durtime:18  Amount:4898
Random event:   Arrtime:464   Durtime:16  Amount:1109
Random event:   Arrtime:541   Durtime:19  Amount:-3853
Event   | Arrtime   | Durtime   | Amount  | Leavetime
   0  |        0  |       37  |   -17188  | -1
   1  |       77  |       38  |   -18238  | -1
   2  |      152  |       11  |   +10162  | -1
   3  |      158  |       36  |   -17526  | -1
   4  |      232  |       40  |    -6056  | -1
   5  |      322  |       15  |    +4177  | -1
   6  |      384  |       18  |    +4898  | -1
   7  |      464  |       16  |    +1109  | -1
   8  |      495  |       35  |   -12909  | -1
   9  |      541  |       19  |    -3853  | -1
  10  |      580  |       10  |   -19583  | -1

Process Event #0 (-17188) but Fail (there's only 12604)
Main Queue: 1 2 3 4 5 6 7 8 9 10
Wait Queue: 0

Process Event #1 (-18238) but Fail (there's only 12604)
Main Queue: 2 3 4 5 6 7 8 9 10
Wait Queue: 0 1

Process Event #2 (+10162):
Arrive Time:152
Time:152 -> 163
Total Amount:12604 -> 22766
```

```
Main Queue: 3 4 5 6 7 8 9 10
Wait Queue: 0 1

[Process the wait queue]
Process Event #0 (-17188):
Arrive Time:0
Time:163 -> 200
Total Amount:22766 -> 5578
Main Queue: 3 4 5 6 7 8 9 10
Wait Queue: 1

[Process the main queue]
Process Event #3 (-17526) but Fail (there's only 5578)
Main Queue: 4 5 6 7 8 9 10
Wait Queue: 1 3

Process Event #4 (-6056) but Fail (there's only 5578)
Main Queue: 5 6 7 8 9 10
Wait Queue: 1 3 4

Process Event #5 (+4177):
Arrive Time:322
Time:322 -> 337
Total Amount:5578 -> 9755
Main Queue: 6 7 8 9 10
Wait Queue: 1 3 4

[Process the wait queue]
Process Event #1 (-18238) but Fail (there's only 9755)
Main Queue: 6 7 8 9 10
Wait Queue: 3 4 1

Process Event #3 (-17526) but Fail (there's only 9755)
Main Queue: 6 7 8 9 10
Wait Queue: 4 1 3

Process Event #4 (-6056):
Arrive Time:232
Time:337 -> 377
Total Amount:9755 -> 3699
Main Queue: 6 7 8 9 10
```

```
Wait Queue: 1 3

[Process the main queue]
Process Event #6 (+4898):
Arrive Time:384
Time:384 -> 402
Total Amount:3699 -> 8597
Main Queue: 7 8 9 10
Wait Queue: 1 3

[Process the wait queue]
Process Event #1 (-18238) but Fail (there's only 8597)
Main Queue: 7 8 9 10
Wait Queue: 3 1

Process Event #3 (-17526) but Fail (there's only 8597)
Main Queue: 7 8 9 10
Wait Queue: 1 3

[Process the main queue]
Process Event #7 (+1109):
Arrive Time:464
Time:464 -> 480
Total Amount:8597 -> 9706
Main Queue: 8 9 10
Wait Queue: 1 3

[Process the wait queue]
Process Event #1 (-18238) but Fail (there's only 9706)
Main Queue: 8 9 10
Wait Queue: 3 1

Process Event #3 (-17526) but Fail (there's only 9706)
Main Queue: 8 9 10
Wait Queue: 1 3

[Process the main queue]
Process Event #8 (-12909) but Fail (there's only 9706)
Main Queue: 9 10
Wait Queue: 1 3 8
```

```
Process Event #9 (-3853):
Arrive Time:541
Time:541 -> 560
Total Amount:9706 -> 5853
Main Queue: 10
Wait Queue: 1 3 8

Process Event #10 (-19583) but Fail (there's only 5853)
Main Queue: <EMPTY>
Wait Queue: 1 3 8 10

Event   | Arrtime   | Durtime   | Amount  | Leavetime
   0  |        0  |       37  |   -17188  | 200
   1  |       77  |       38  |   -18238  | 600
   2  |      152  |       11  |   +10162  | 163
   3  |      158  |       36  |   -17526  | 600
   4  |      232  |       40  |    -6056  | 377
   5  |      322  |       15  |    +4177  | 337
   6  |      384  |       18  |    +4898  | 402
   7  |      464  |       16  |    +1109  | 480
   8  |      495  |       35  |   -12909  | 600
   9  |      541  |       19  |    -3853  | 560
  10  |      580  |       10  |   -19583  | 600

Average stay time: 137.636364 (min).
Unfinished events: 1 3 8 10
```

七、附录

提交的文件结构如下:

data/data1.in // 测试数据 1

data/data2.in // 测试数据 2

data/data3.in // 测试数据 3

src/main.c // 本次实验的源代码

report.pdf // 实验报告