Document Name		Confidentiality Level
Simulate Memory Pagination		Only for Recipients' Reference
Project Code	Version	Document Code
BP	1.0	BP

Simulate Memory Pagination

Copyright Declaration

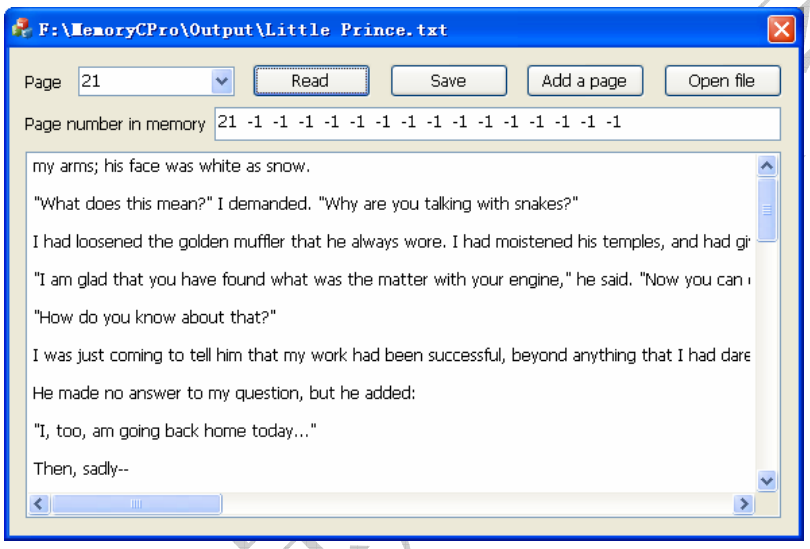
Contents

1 Teaching Tips

(1) Master the basic principle of memory pagination.
(2) Master LRU scheduling algorithm.
(3) Master how to dynamically allocate and release memory in C/C++.
(4) Master how to use malloc(), realloc(), free(), new and delete functions.

2 Functional Requirements

Write a MFC dialog program to paged read, display and update “Little Prince.txt” file, and add new pages. Simulate the paging mechanism of the memory management in the operating system.



The specific requirements are as follows:
1. Specify a file “Little Prince.txt”.
With the file selector, select a txt file, and generate the index page table information for this file.

2. Simulate Paging
Paged read and update the file.

3. Memory Area
Open up a memery area of 64K on the heap as the buffer to read/write file data. Take each 4K as a page to page this memory.
(1) Total memory area: 64*1024Byte
(2) Memory area per page: 4*1024Byte
(3) There are 16 pages totally. Page number: 0~15.

4. Page Replacement Algorithm
For the unused data in the memory, according to LRU algorithm, move it out of the memory and free up space to load other data. LRU replacement algorithm will select the page to be replaced. Write a special stack to determine the least recently used page number, and move the most recently used page number to the stack top.

5. Add Pages
When a page of content is added, allocate new page table memory area with realloc().

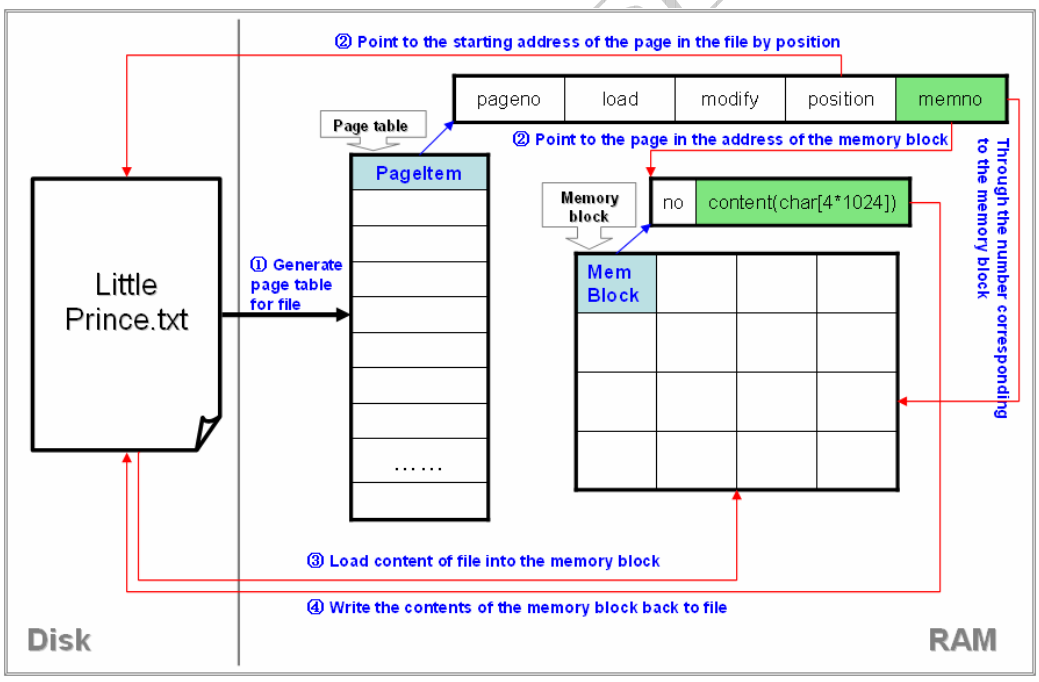
3 Design Ideas

3.1 Technology Selection

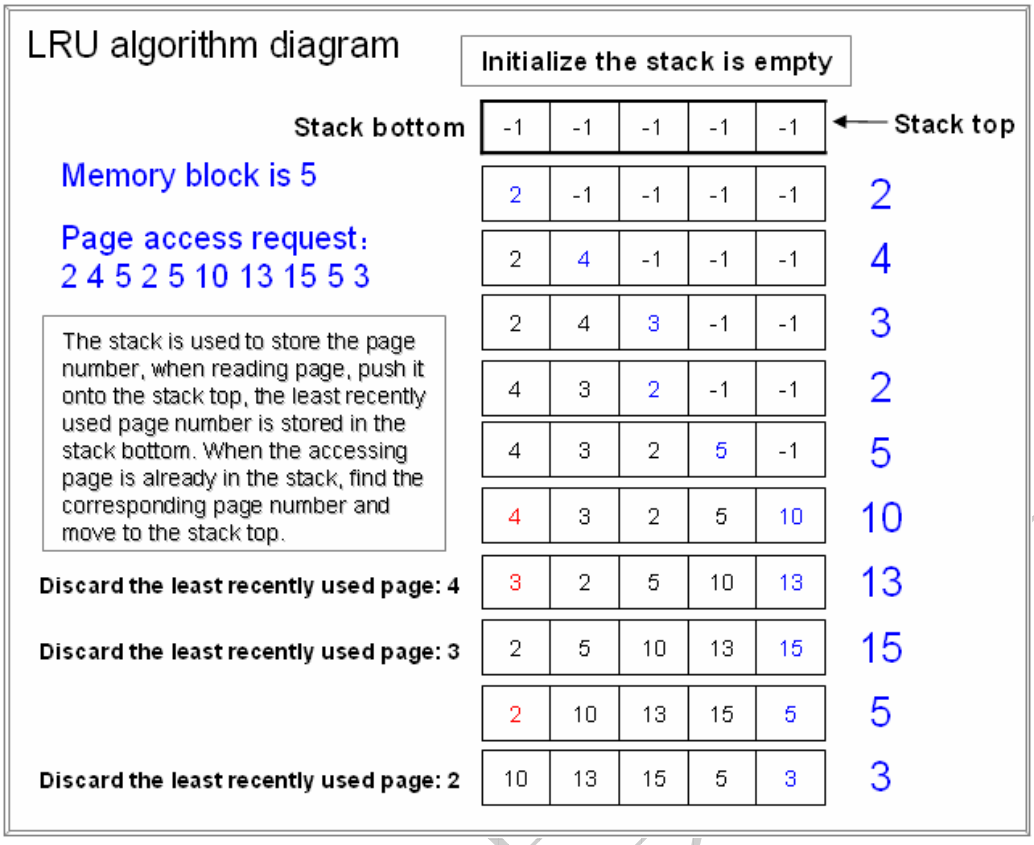
1. **Development tool**: Visual Stdio 2010.
2. **Program type**: MFC dialog program.
3. **Project name**: `MemoryCPro` (the solution has the same name as the project)

3.2 Implementation Idea

By reading a specific file, simulate the program to be loaded in the memory. Create 16 memory blocks of 4*1024Byte as the area to buffer the file content. Create the page table for the file, and simulate the page mapping table in page management.

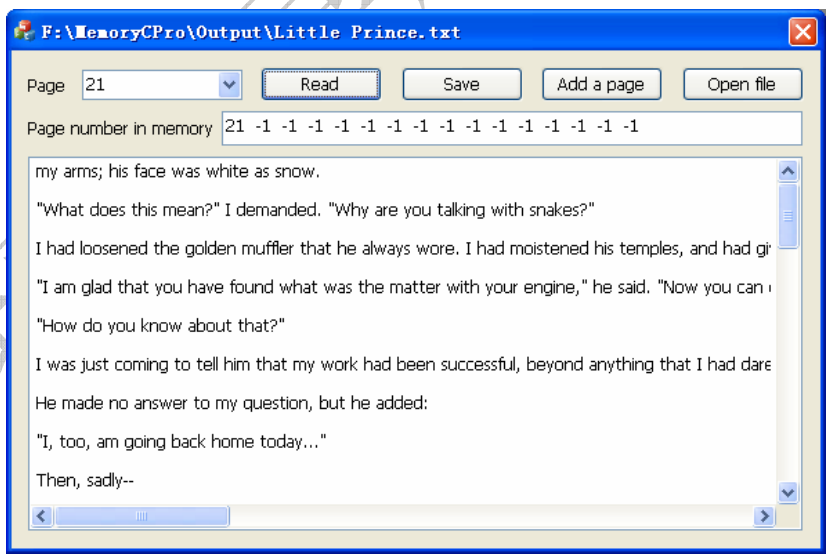


Divide the file by the size of 4*1024Byte, and create the page table. In the page table, it will store whether this page has been loaded into the memory block, whether it has been modified, the start address in the corresponding file, the number of the corresponding memory block. If it has not been loaded into the memory block, the number is -1.
Implement the determination of LRU with LRU replacement algorithm and the stack.



3.3 Interface Design

1. Interface



2. Interface Description

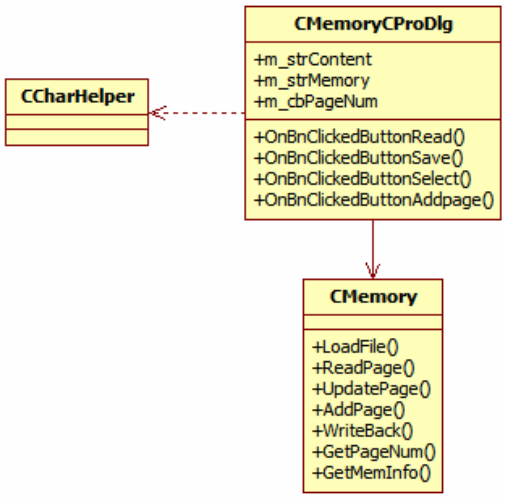
(1) Function buttons: “Read”, “Save”, “Add a Page”, “Open file”.

(2) Page number selection: a combo box. After “Select File”, the program will generate the page number automatically.

(3) Display the memory block information: display the page number of the content in the memory block with the edit control.
(4) Display the page content: display the content with the edit control in multi-line style. The size is 4*1024 byte. It is editable.
(5) When we select the file, we should use the file selector.
(6) The program will prompt the message with the message box.
(7) The menu bar displays the full path of the currently read file.

3.4 Class Structure

The core class diagram structure of the program is shown below:



CMemoryCProDlg class is the main window class of the program. It is used to response the user's operation, and display the memory information, page number information and file content.
CCharHelper class is an auxiliary class. It is used to implement the conversion between char array and CString class in VS.
CMemory class is a core logic class. It is used to simulate the function of page management in memory. Simulate the page of memory with the structure array. Simulate the stack operations with the integer array.

3.4.1 CMemoryCProDlg class

1. Description

The main dialog class. The base class is CDialogEx class. When the program is created, it is automatically generated by the application wizard. It is used to response user's operation and display.

2. Data Member

Type	name	Description
CComboBox	m_cbPageNum	Combo box control of the page number in the map

		window.
CString	m_strContent	Multi-line edit control that displays the file content in the map window.
CString	m_strMemory	Edit control that display the memory page number in the map window

3. Member Function

Function	Description
OnBnClickedButtonRead	Response window “Read” button click event.
OnBnClickedButtonSave	Response window “Save” button click event.
OnBnClickedButtonSelect	Response window “Open File” button click event.
OnBnClickedButtonAddpage	Response window “Add a Page” button click event.

3.4.2 CMemory class

1. Description

This class is a core logic class that is used to simulate the function of page management in the memory.

2. Data Member

Type	Name	Description
MemBlock*	m_pMemory	It is used to point to the start address of the memory area that stores the page information. The size per block is 4*1024Byte. There are 16 blocks totally.
int[MEM_NUM]	m_anStack	It is used to implement the stack of LRU.
PageItem*	m_pPageTable	It is used to point to the start address that saves the page table information.

3. Member Function

Function	Description
LoadFile	Select the file to be loaded, and generate the page table information of the index for this file.
ReadPage	Read the information of a page. When a page exists, read directly from the memory. When a page does not exist, load it. If the replaced page has been modified, write back into the file.
UpdatePage	Update the information of a page into the file, and set the modification identifier as false.
AddPage	Add a new page, add the page table information, and select the new page into the memory. The default content is null.
WriteBack	Write back the modified page information in the memory. Call before the program ends.
GetPageNum	Get the total page number.

GetMemInfo	Get the page number information stored in the current memory block.

3.5 Data Structure

3.5.1 Memory Block

(1) Page area per page: 4*1024byte
(2) There are 16 pages totally. The memory number: 0~15.
(3) The memory structure is defined as follows:

```
#define PAGE_SIZE (4*1024)
#define MEM_NUM 16

struct MemBlock
{
   int no;                 // Number
   char content[PAGE_SIZE];    // Memory page content
};
```

(4) The total memory area: 16*4*1024byte. Store the structure array.

3.5.2 Page Table Entry

When the file is loaded, generate the page table index information of the file through the file size.
The structure of the index entry is as follows:

```
struct Index
{
   int pageno; // Page number.
   bool load;  // Determine whether it has been loaded into the memory, false:
it has not been loaded, true: it has been loaded into the memory.
   bool modify; // Whether it has been modified, false it has not bee modified,
true it has been modified.
   ULONGLONG position; // Indicate the start address of this page on the file.
   int memno;  // Physical block number.
};
```

The page number starts from 0.

3.5.3 LRU Replacement Algorithm Auxiliary Structure - Stack

int array: int m_anStack[MEM_NUM];

Stack top: int m_nStackSize;
Stack bottom: m_anStack[0];

4 Technical Analysis

4.1 Memory Page Management

Page storage management is to divide the logic address space of a thread into some pieces of the same size, called pages, and add numbers for the pages. The numbers start from 0, for example, page 0, page 1, and so on. Accordingly, divide the memory space into some storage blocks of the same size as the pages, called (physical) block or page frame, too. Also give them numbers, for example, 0# block, 1# block, and so on. When the memory is allocated for the thread, some pages in the thread will be loaded by the unit of block into many physical blocks that can be nonadjacent.

In the page system, it allows to store the pages of the thread into different physical blocks in the memory discretely. But, the system should ensure the proper running of the thread. That is, the corresponding physical block of each page can be found in the memory. For this purpose, the system also creates a page mapping table for each thread, the page table for short. Each page (0~n) in the thread address space has a page table entry in the page table in turn. The physical block number corresponding to the page in the memory is recorded in it. After the page table has been configured, when the thread is executed, the physical block number of each page can be found in the memory by searching this table. We can see that the page table can implement the address mapping from the page number to the physical block number.

4.2 LRU Replacement Algorithm

LRU(Least Recently Used) replacement algorithm is to select the least recently unused page to replace. This algorithm will give each page an access field to record the time t from the last access of a page. When a page must be replaced, select a page of the maximum t value from the existing pages, that is, replace the least recently unused page.

Though LRU replacement is a better algorithm, but it requires the system has more hardware supports. In order to know how long each page of a thread in the memory has not been accessed by the thread, and how to quickly know which page is the least recently unused page, we must have one of the following two kinds of hardware to support:

(1) Register

In order to record the use of each page of a thread in the memory, we must configure a shift register for each page in the memory. It can be represented as R=Rn-1Rn-2Rn-3……R2R1R0. When the thread accesses a physical block, we should set Rn-1 bit of the corresponding register as 1. At this time, the timing signal will shift the register a bit at regular intervals (such as 100ms). If we regard the number of the n-bit register as an integer, the page corresponding to the

register that has the minimum value is the least recently unused page.

(2) Stack

Implement LRU replacement algorithm with the stack: save the page number of each currently used page with a special stack. Every time when the thread accesses a page (read or write), move the page number of this page out of the stack, push it onto the stack top. So, the stack top is always the number of the most recently accessed page, and the stack bottom is the page number of the least recently unused page.

4.3 Memory Management

In C++, the memory is divided into five areas: heap, stack, free storage area, global/static storage area and constant storage area.

1. Stack

It is the variable storage area allocated by the compiler when needed, and cleared automatically when not needed. The variables inside are usually the local variables, function parameters, and so on.

2. Heap

The memory block allocated by new. The release of this area is not managed by the compiler, but it is controlled by the application. Generally, a new corresponds to a delete. If a programmer has not released it, the operating system will automatically recycle it after the program ends.

3. Free Storage Area

The memory block allocated by malloc and so on. It is similar to the heap. But it will end the life with free. The basic concepts and basic usages of malloc() and free():

(1) malloc() function

void *malloc(long NumBytes): this function allocates NumBytes bytes, and returns the pointer that points to this block of memory. If it fails to allocate, return a null pointer (NULL). The failed allocation has many reasons, such as insufficient space.

(2) free() function

void free(void *FirstByte): this function will give back the space allocated by malloc before to the program or the operating system, that is, release this block of memory and let it get free again.

4. Global/Static Storage Area

The global variables and the static variables are allocated in a same block of memory. In C language, the global variables are divided into initialized and uninitialized. In C++, they occupy the same block of the memory area.

5. Constant Storage Area

This is a special block of storage area. It stores the constants inside, and can not be modified.

5 Implementation Idea

5.1 Create Project

1. Create MemoryCPro dialog project with VS2010 tool. The solution and the project have the same name.

2. Under the solution folder, create Output folder. Configure the executable files under Debug and Release versions to this folder.

3. Copy “Little Prince.txt” file into Output folder.

5.2 Load File

1. Add “Open file” button response event, CMemoryCProDlg::OnBnClickedButtonSelect() function.
Function prototype: void OnBnClickedButtonSelect();

2. The file is “Little Prince.txt” file in the executable file folder. (Here, we need to convert the relative path to the absolute path to move in).

3. Add CMemory::LoadFile() function to read the file, page the file according to the file size, 4*1024 bytes per page, and generate the page table index information.
Function prototype: bool LoadFile(const CString strFilepath);

5.3 Read File

1. Select the page number that you need to display, read the content of the selected page, and display in the window.
(1)Add “Read” button response event, CMemoryCProDlg::OnBnClickedButtonRead() function.
Function prototype: void OnBnClickedButtonRead();

(2)Read the value on the combo control, and convert into integer.

(3)Add CMemory::ReadPage() function.
Function prototype: int ReadPage(const int pageno, char* pStr, const int nMaxSize);

(4)Add CMemory::QueryMemory() function.
Function prototype: int QueryMemory(const int pageno);
Return the memory block number where this page exists.

(5)Convert the read character array to CString, display on edit control of the interface.

2.Implement the page replacement algorithm with LRU algorithm.
(1)Add the variables related to the stack in CMemory class: integer array and stack size.
int m_anStack[MEM_NUM]; // the number stack (points to the stack bottom)
int m_nStackSize; // the stack top

(2) Add stack operation function.
Push stack: void PushStack(const int pageno);
search whether this page number has existed in the stack. If it has existed, move this page number to the stack top. If it does not exist, add it. If the stack has been full, move out the element of the stack bottom, and put this number on the stack top.

Pop stack: int PopStack();

(3) Call PushStack() function in CMemory::ReadPage() and CMemory::UpdatePage() functions.

(4) Call PopStack() function when you need to replace the page in CMemory::LoadPage() function.

5.4 Update File

1. Add “Save” button response event, CMemoryCProDlg::OnBnClickedButtonSave() function.

2. Add CMemory::UpdatePage() function.
Function prototype: bool UpdatePage(const int pageno, char* pStr, const int nMaxSize);
Query whether this page has existed in the current memory block. If it has existed, update the content in the memory block, and set the modification identifier as true. If it does not exist, load this page.

5.5 Add a Page

1.When a new page is added, allocate the memory space for the page table again with realloc() function to expand the page table information.

2.When the window is logged out, write back all records in the page table.
In CMemory, add WriteBack() function, traverse the whole page table, find the modified page in the memory, and write back into the file.
Add WM_DESTROY message function for the main window, and call CMemory::WriteBack() function in it.

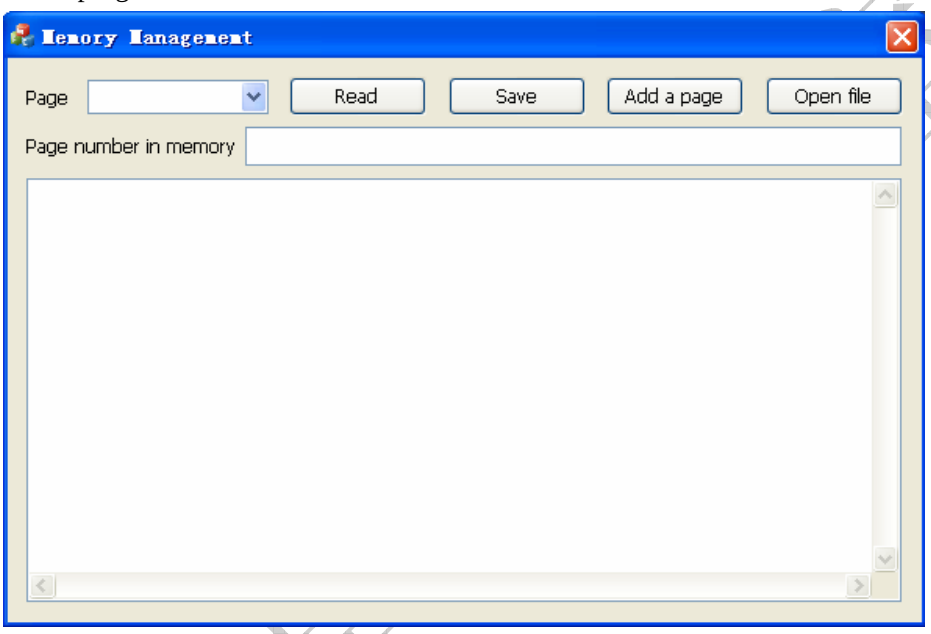
6 Running

6.1 Program Import

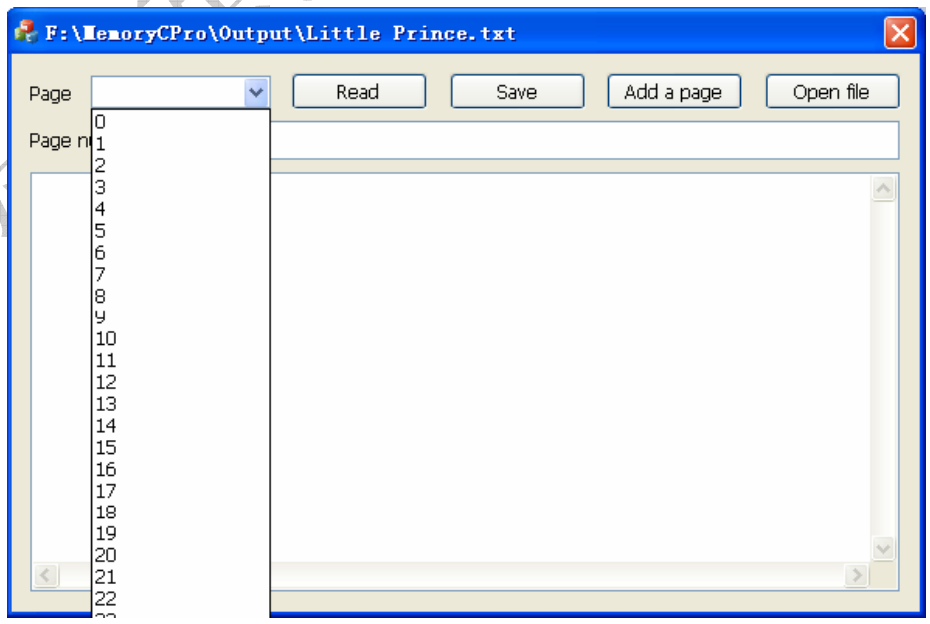
Open VS2010 tool, open MemoryCPro.sln file.

6.2 Results

1. Launch the program.

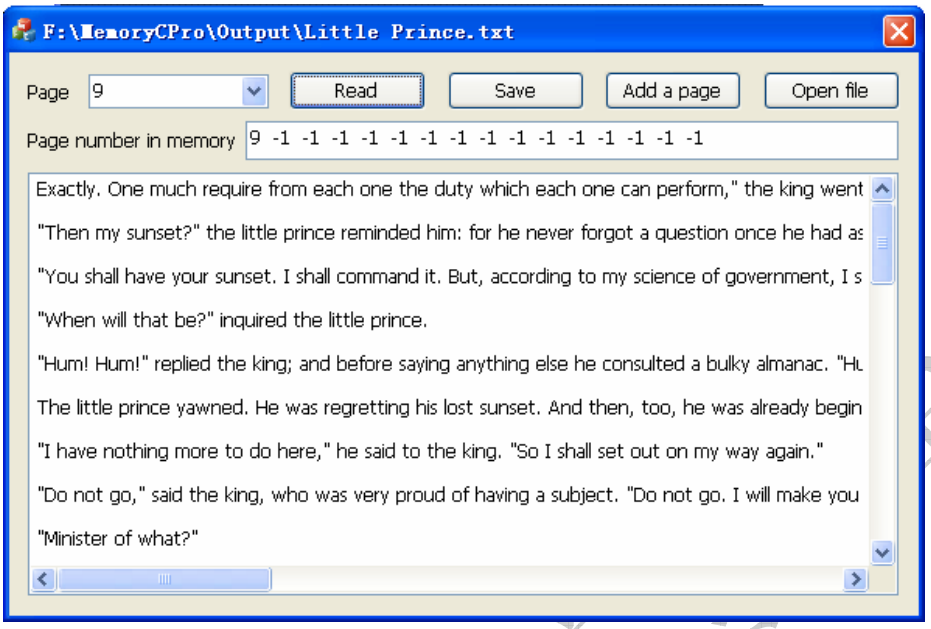


2. Click “Open file” button, open “Little Prince.txt” file.



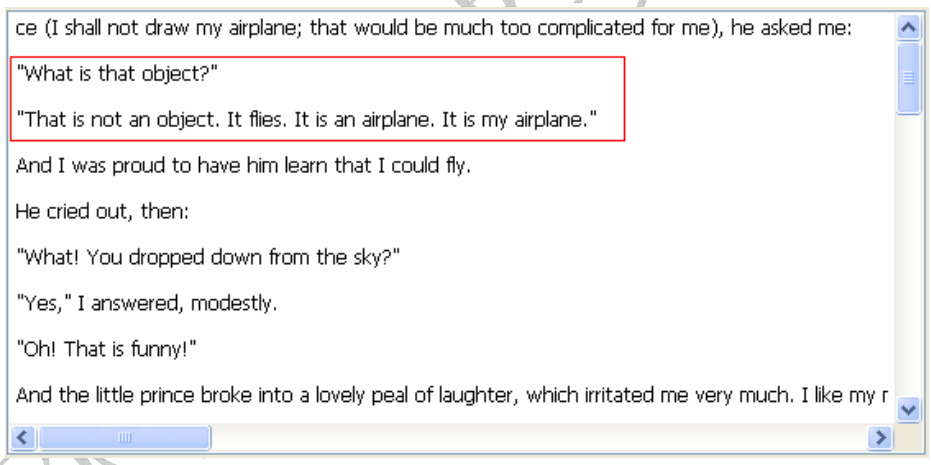
Display all page numbers of the page in the page number combo box..

3. Select page “9”, click “Read” button.

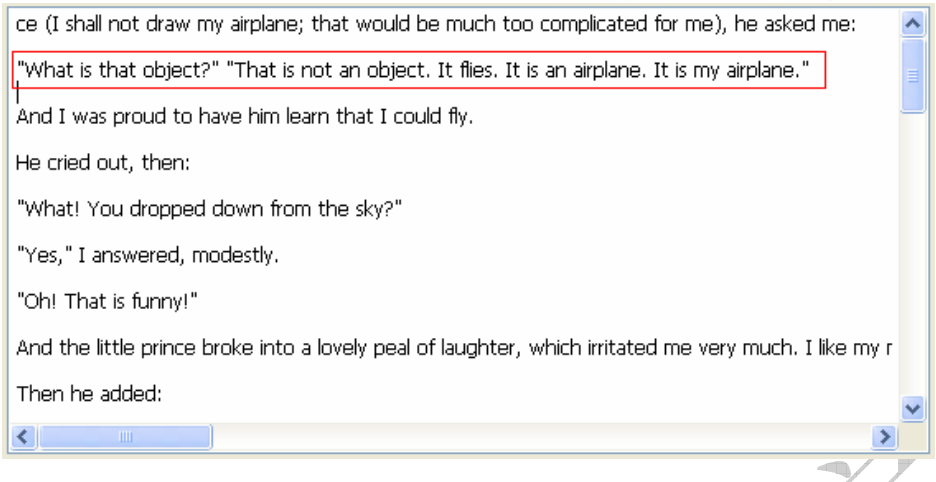


4. Select page “2”, click “Read” button, modify the content, click “Save” button.

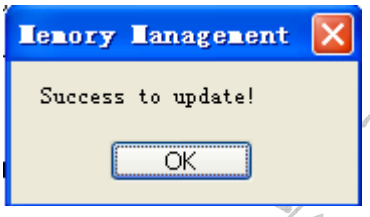
Original content:



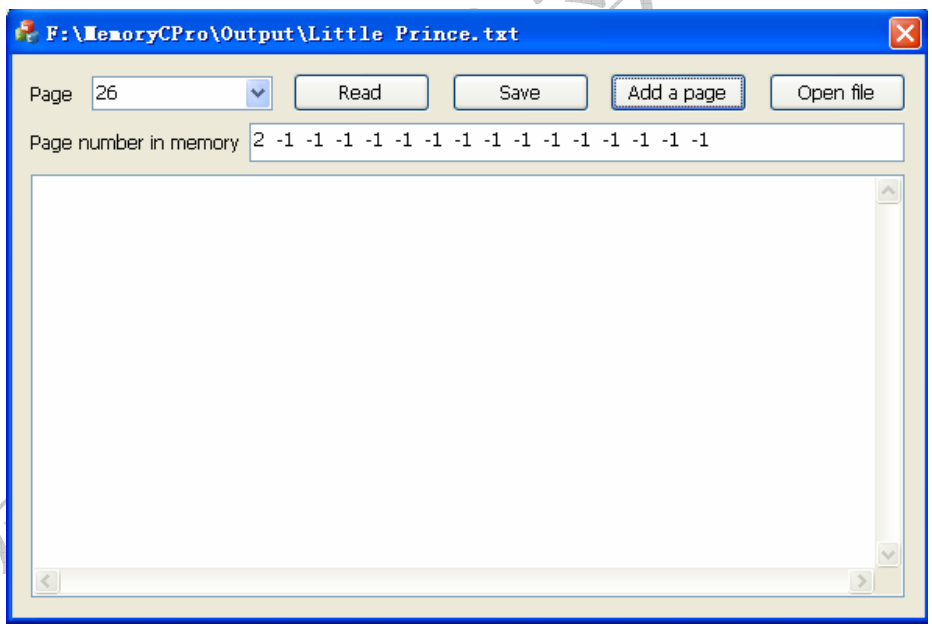
New content (original 2 and 3 lines have been merged into a line):



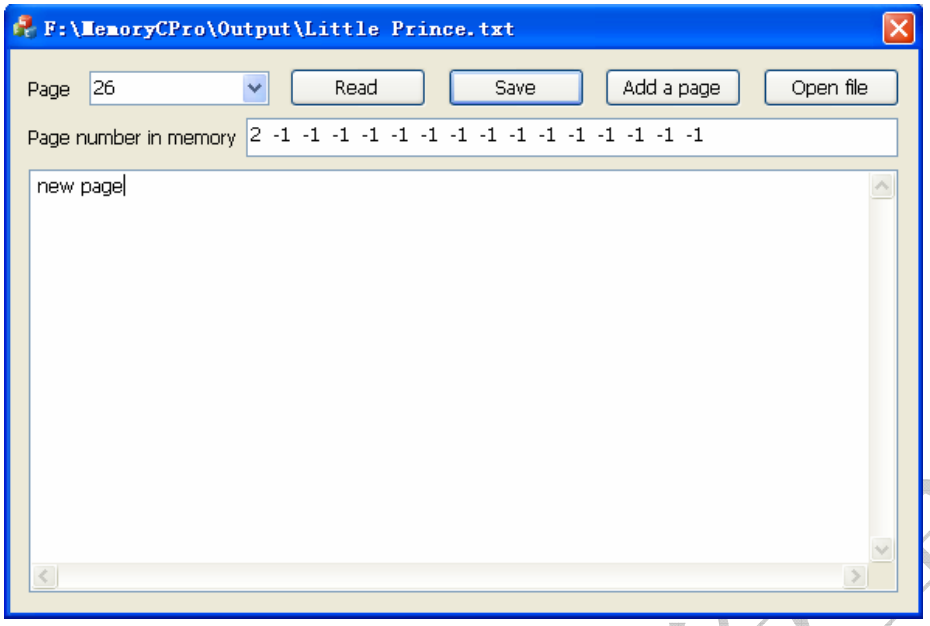
Update successfully. Prompt: “Update successfully!”



5. Click “Add a page” button.

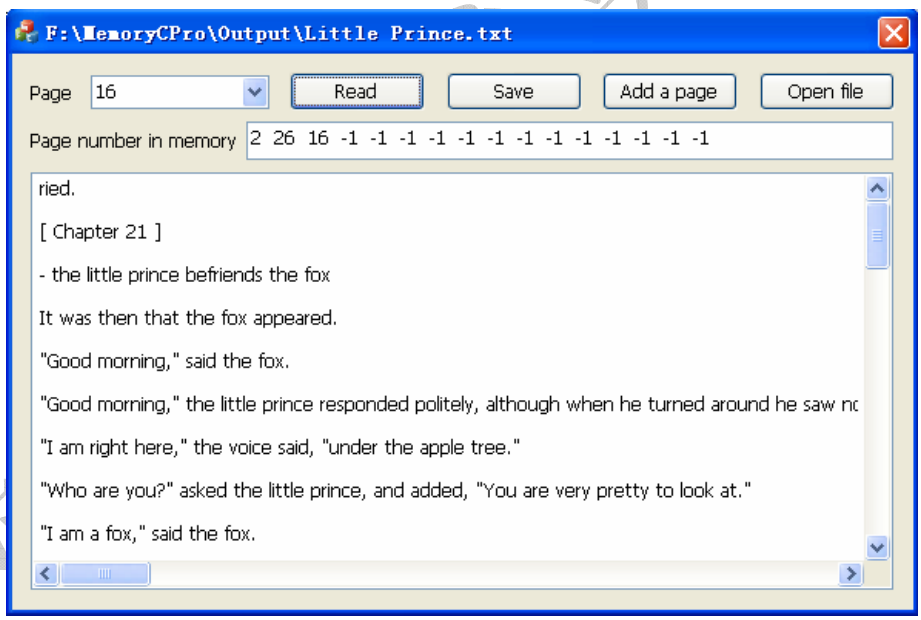


Click “Save” button.



Select page “25”, click “read”, display the content.
Switch to page “9”, and then, select page “25” again.

6. Select and view any different pages.



7. Close the program, start again, and view whether the new page has been saved.

