基本语法

机制

内存模型和名称空间

内存模型
https://www.cnblogs.com/rednodel/p/9300729.html

const:

const a:

声明变量a为常量，不可改变，默认具有static的性质，链接性为内部，因此可以放到头文件中。

extern const a:

使a具有外部链接性。

名称空间：

把标识符的声明放到特定的空间中，

把全局作用域分为了层级关系，可以嵌套使用

可以随时增加。

和全局作用域和局部作用域的关系和以前保持一致。

```cpp
#include <iostream>
#include "file1.h"
#include "file2.h"
using namespace std;
namespace hetong
{
	namespace heliang1
	{
		int a;
		void func1(void)
		{
			cout<<"ok\n";
		}
	}
}
namespace heliang
{
	int a;
	void func1(void)
	{
		cout<<"lala\n";
	}
}
int main (int argc, char *argv[])
{
	using namespace hetong::heliang1;
	using namespace heliang;
	cout << "hello world" << endl;
	//hetong::heliang1::func1();
	//func1();
	return(0);
}
```

作用域，链接性

1处定义　->　多处声明　->链接到定义

作用域表示：声明的位置决定了有效作用区间

链接性：表示定义的标识符是否可以被其他声明链接到（内部，外部，无）

extern关键字：表示对外部变量的声明。

```
static int a;不会生成符号表
int b;如果有extern 有链接属性，那么会生成符号表
fun()
{
    extern int a;声明变量a 首先会找本文件，然后找其他文件
}
int c=10;
```

通过static可以改变默认具有extern变量的链接性为内部链接性

只有定义可以初始化赋初值

作用域：

代码块作用域

文件作用域

链接性：

外部链接性

无链接性

存储特性：

静态存储

动态存储

全局变量

作用域为文件作用域，

默认具有外部链接性，通过extern修饰声明可以扩展到另一个文件相应的作用域内。 static修饰定义，修改为无链接性。

存储特性为静态存储。

局部变量

作用域为代码块作用域，

无链接性

存储为动态存储，static 可以修改为静态存储。

技巧：

首先弄清楚是声明还是定义。定义才是主体，声明只是对主体的一种延伸，扩展。

下面这种情况：

extern int a; a.c

extern int a; b.c

如果使用，编译器会不知道哪个才是定义。会出错。

因此，最好，定义不要加extern，赋初值。

数据

字面值：

- 整型字面值
  - char ,int ,long ,long long
  - enum
- 浮点型字面值
  - float , double , long double

基本的数据型

C:\Users\hetong\Desktop\mind\数据类型.xmind

全局：

static type

const type

const static type

extern type

函数：
static
const
&
const &

类：
static 成员函数或者成员变量为类属性，需要在类外定义

const 成员函数 不能修改类成员的值
const 成员变量 只能在构造函数初始化列表初始化

static const 可以在类声明处初始化

枚举型

```
enum DAY
{
    MAX = 20
};
```

enum

```
{
    MAX =20
};
```

第一种定义了一种枚举类型 enum DAY; 同时定义了枚举常量MAX;

第二种定义了枚举常量；

指针

练习：

typedef

typedef 的作用就是声明一种新的类型

int a
typedef int a;

void *(*FUNC)(int)
typedef void*(*FUNC)(int)

const关键字总结

注意点：

- 定义时必须初始化
- 默认为文件作用域，必须显示定义extern才能使用

语法

```
const type const a;
```

第一个const对象为type a,第二个const对象为a

```
const int * const a;
```

第一个const：*a,*a为一个常量，通过a来访问的是个常量
第二个const：a为一个指针常量，只能指向固定的内存区域。

```
const function(const va)
```

const来限定函数

```
function() const
```

对象的常量

总结：

- 一旦引用已经定义，它就不能再指向其他的对象

- 非 const 引用只能绑定到与该引用同类型的对象。
- const 引用则可以绑定到不同但相关的类型的对象或绑定到右值。

例子：

static关键字

基本型的相互转换

https://blog.csdn.net/nanfeibuyi/article/details/80811498

字符串和int float转换
https://blog.csdn.net/qq_43010942/article/details/87204306

定义声明和初始化

int a =10;
定义声明初始化a;

extern int a =10;
定义声明初始化a;

extern int a;
声明一个全局变量a;

如何看不同文件不冲突：
看这个变量定义的区域：

int a;
全局变量区，具有外部链接属性，其他文件可见

static int a； 全局静态，本文件独有，其他文件不可见

const int a; 有静态属性

const statci int a; 有静态属性

语句

操作符和表达式

动态内存分配

calloc和malloc函数

```
void *malloc(long NumBytes)
```

分配内存后，不会初始化，为随机值

```
void *calloc(size_t num,size_t size)
```

分配内存后，初始化为0

```
void free(void *FirstByte)
```

new_delete

1. 分配某个类型的空间

```
T * x = new T
```

1.分配某个类型的数组

```
T  *x = new T[n]
```

3.分配二维数组

知道行数和列数

```
T  * c = new T[n][m]
```

知道列数

```
T (*c)[m] = new T[n][m]
```

行数和列数都不知道

```
x = new T * [n]

for(int i =0; i < numberOfRows; i++)
{
   x[i] = new T[numberOfColumns]
}
```

二维数组a[n][m]

a表示第一个元素a[0]的地址，a[0] 表示 a[0][0]的地址

函数

参数传递

非引用：

传递的是拷贝的副本，const限定了是否可以修改，

如果左值const限定，那么右值就必须是const

引用：

传递的是引用，

非const引用传递时必须保持类型一致。

const引用时，可以不保持类型一致。

注意：const int *&a; 并不是上面说的const引用。

函数模板

函数模板：

```
template <typename T,typename M>
TypeName func(T a,M b)
```

显示具体化：explicit specialization

```
typename<> TypeName<int ,double> func(int a,double b)
```

或者

```
typename<> TypeName func(int a,double b)
```

调用的时候：可以显示实例化　explicit　instantiation

```
func<int,double>(a,b)
```

默认隐式调用：　implicit instantiation

默认函数>显示具体化explicit specialization >explicit instantiation >implicit instantiation

运算符重载

格式：
operator op ()
例如：operator+(const Time & t),重载+运算符。

coding.operator+(fixing)
coding + fixing

lamda表达式

1.
[]()->type{}

[ ]用于变量的限制，= & VA varuable

() 放参数

- > 如果不是一条语句，没法推断返回值类型，需要显示指定返回值类型

{ } 函数的实现体

类

对象初始化

构造函数

1. string　str("hetong"); 构造函数
2. string str = string("hetong");
3. string　*pstr = new string("hetong");
4. string str = “hetong”　拷贝构造函数
5. string str1 = str;　拷贝构造函数
5. string str2; str2 = str1; = 重载

复制构造函数

的复制构造函数和复制

操作

当一个对象生成或者赋值的时候，会从构造函数里面找有没有符合参数的工作函数，然后复制或者调用赋值构造函数

1.
显示对用构造函数
string a("hetong")

2.
显示调用构造构造函数构造临时对象，调用复制构造函数复制对象到a
string a = string("hetong")

3.
void func(const string a)
func("hetong")

调用string构造函数生成临时对象，调用复制构造函数

4.
void func(const string &a)
func("hetong")
调用string构造函数生成临时对象，然后引用对象

5.
对象复制
string v1;
string v2;
v1 = v2;

6.
string v1;
v1 = “hetong”;

调用构造函数，然后对象复制

继承

G:\cherry笔记\继承.md

的名称空间和作用域

在类中：

private：
只能通过public的成员函数来访问

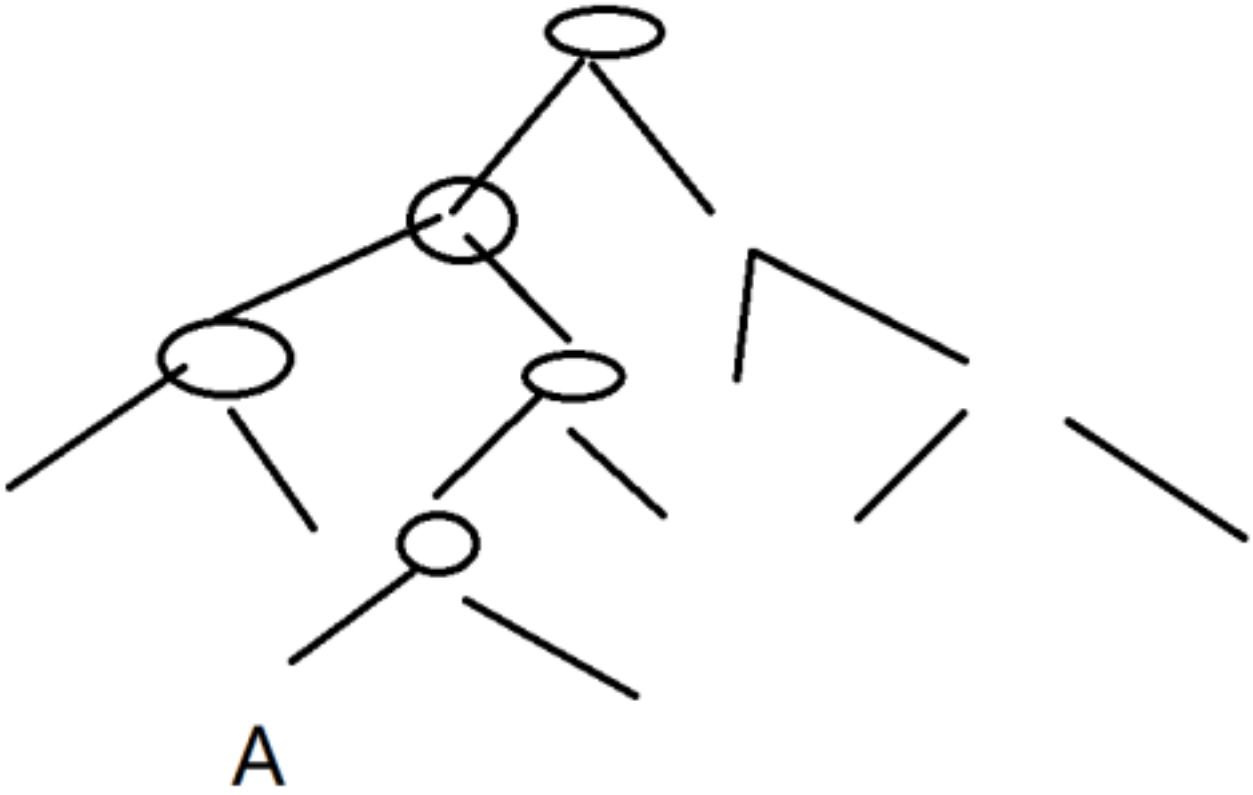
public：
可以通过类对象直接访问

static：表示这个成员是类的属性还时对象的属性
访问规则遵循上面的public 和private；

名称空间：

把全局作用域重新划分层级：

当前层级只能看到从自己到外面的名字解析



A只能看到自己左上遍历的名字，而且后面

的名字会覆盖前面的

抽象数据类型

c数组

a[]

a为一个指针常量，指向a数组的第一个元素

&a表示指向数组的指针

c字符串

string类

string

string类的语法和平常的算术运算很像
创建对象： string name;
getline(cin,str)来从键盘输入值；

用“+”来拼接，用==来比较。

字符串和其他类型之间的转换

atoi()
atol()
atof()

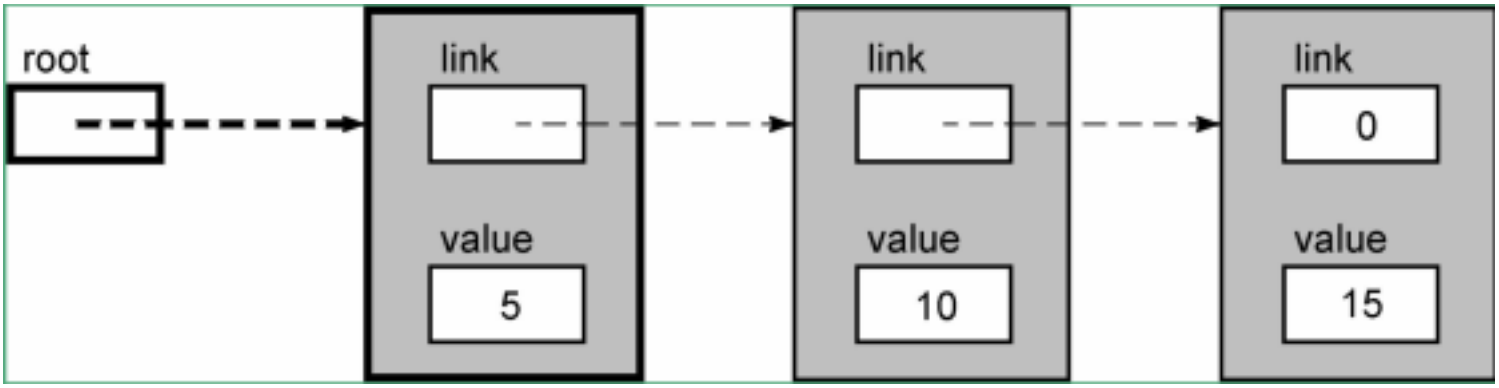
itoa()
ltoa()
ftoa()
sprintf()
sscanf()

to_string()
assign()
通过 =

c_str()然后调用c函数
stringstream 流 #include <sstream>

链表

单项有序链表



代码：

```
typedef struct node
{
	struct node  *link;
	int date;
}NODE_STRUCT;
```

第三行需要用：struct xxx定义

插入函数：

```
int insert_node(NODE_STRUCT **linkp,int newdate)
{
	NODE_STRUCT *current;
	NODE_STRUCT *newnode;

	while ((current=*linkp)!=NULL && current->date < newdate)
	{
		linkp = &(current->link);
	}

	newnode = (NODE_STRUCT *)malloc(sizeof(NODE_STRUCT));
	newnode->date = newdate;

	newnode->link = current;

	*linkp = newnode;
	return 1;
}
```

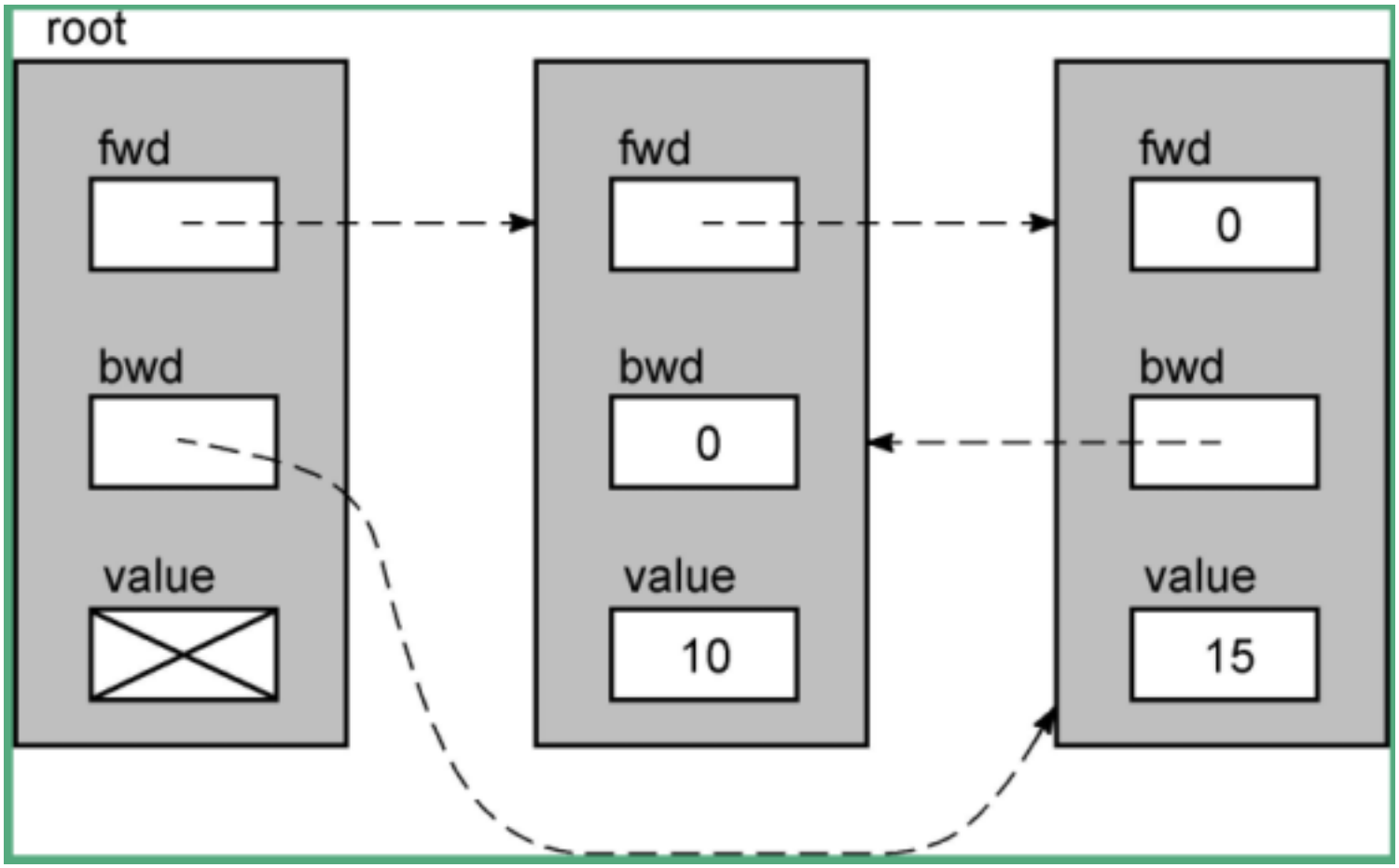
测试遍历函数

```
int main()
{
	//操作
	NODE_STRUCT *root = NULL;
	insert_node(&root, 10);
	insert_node(&root, 5);
	insert_node(&root, 4);
	insert_node(&root, 50);
	insert_node(&root, 3);
	insert_node(&root, 3);

	NODE_STRUCT * current;
	for(current = root;current != NULL; current = current->link)
	{
		printf("single value is %d \n",current->date);

	}
}
```

查找函数

```
int find_node(NODE_STRUCT **linkp, int find_date)
{
	unsigned num = 0;
	NODE_STRUCT *current;

	while ((current = *linkp) != NULL)
	{
		if (current->date == find_date)
		{
			num++;
		}
		linkp = &(current->link);
	}
	return num;
}
```

双向链表



插入操作：

传入指向根节点的指针
current指针，next指针遍历操作

current指向新节点，newnode指向next

向后指针需要分情况，是否为跟节点，是否为尾部

```
typedef struct doublenode
{
	struct doublenode *fwd;
	struct doublenode *bwd;
	int date;
}DOUBLE_NODE_STRUCT;

int insert_double_node(DOUBLE_NODE_STRUCT *rootp,int value)
{
	DOUBLE_NODE_STRUCT *newDate;
	DOUBLE_NODE_STRUCT *current;
	DOUBLE_NODE_STRUCT *next;

	for (current = rootp; (next = current->fwd) != NULL; current = next->fwd)
	{
		if (next->date == value)
		{
			return 0;
		}
		if(next->date > value)
		{
			break;
		}
	}

	newDate = (DOUBLE_NODE_STRUCT *)malloc(sizeof(DOUBLE_NODE_STRUCT));
	if (newDate == NULL)
	{
		return -1;
	}
	newDate->date = value;

	current->fwd = newDate;
	newDate->fwd = next;

	if (current == rootp)
	{
		newDate->bwd = NULL;
	}
	else
	{
		newDate->bwd = current;
	}
	if (next == NULL)
	{
		rootp->bwd = newDate;
	}
```

```
        else
        {
                next->bwd = newDate;
        }
        return 1;
}
```

遍历操作

```
        int iResult = true;
        DOUBLE_NODE_STRUCT rootNode =
        {
                NULL,
                NULL,
                0
        };
        iResult = insert_double_node(&rootNode, 2);
        iResult = insert_double_node(&rootNode, 4);
        iResult = insert_double_node(&rootNode, 1);
        iResult = insert_double_node(&rootNode, 6);
        //正向
        for (DOUBLE_NODE_STRUCT * current = &rootNode; current->fwd != NULL; current = current->fwd)
        {
                printf("%d\n",current->fwd->date);
        }
        //反向
        for (DOUBLE_NODE_STRUCT *current = &rootNode; current->bwd != NULL; current = current->bwd)
        {
                printf("%d\n",current->bwd->date);
        }
```

https://www.typingclub.com/sportal/program-3/281.play

STL

设计模式

单例模式

目的：创建一个只能实例化一次的类

原理：在private区创建一个类指针。
在public区创建一个statci实例函

数，返回值为类指针。

IO流

输入输出函数

文件缓冲区
https://blog.csdn.net/weixin_43202635/article/details/97928825

https://blog.csdn.net/lucky52529/article/details/90546674

https://www.cnblogs.com/hanxiaoyu/

p/5677677.html

输入输出原理

顺序：

查看终端缓冲区，在字符缓冲区截取需要的字符串并保留剩下的在字符缓冲区，如果为空，终端输入字符串序列，按endl后，在字符缓冲区截取需要的字符串并保留剩下的在字符缓冲区，执行下一条语句。

```
cin.getline(variabel,SIZE)
cin.get(variable,SIZE)
cin.get()
```

cin: 通过空白字符来判断。保留空白字符在缓冲区

cin.getline()，通过endl来判断，并读取endl然后替换为\0。

cin.get()，有参数时通过endl来判断，但

是保留endl，如果没有参数，从屏幕缓冲区读取任意的下一个字符。

stringstream

stringstring
https://blog.csdn.net/fanyun_01/article/details/66967710

STL

https://www.cnblogs.com/mu-tou-man/p/3976990.html
容易的比较
https://blog.csdn.net/saizo123/article/

details/53165832

map

排序

https://www.cnblogs.com/eilearn/p/9473804.html

多key 一个value

G:\cherry笔记\mutilekeyMap.md

https://blog.csdn.net/Lin_QC/article/details/104220929

http://www.cppblog.com/yindf/archive/2009/03/20/77304.aspx

vector

sort

https://www.cnblogs.com/stones-dream/p/10183210.html

string

操作总结：
.date() size()
< > ==
insert() push_back() pop_back
append() +
 iterator []
erase()
replace()
find()
sort

分割：
substr()+find实现 或者用c语言 strtok()
https://blog.csdn.net/qq_34634812/article/details/102407308

```cpp
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
    //char str[]="a,b,c,d";
    //如果输入是string要转成char数组[strcpy(buf,str.c_str())]
    //或者用const_cast转换成char指针
    string str="a,b,c,d";
    const char *delim=",";
    char *p=strtok(const_cast<char *>(str.c_str()),delim);
    while(p)
    {
        cout<<p<<endl;
        p=strtok(NULL,delim);
    }
    return 0;
}
```

https://blog.csdn.net/weibo1230123/article/details/80177898 讲解strtok()

网页总结的成员函数

https://blog.csdn.net/qq_37941471/article/details/82107077

http://c.biancheng.net/view/1441.html

大小写转换

https://blog.csdn.net/AffectiveComputing/article/details/96342791

转换

<sstream>
<iostream>
<fstream>

<string>
stoi stol stod ... (atoi atol atod)
to_string (itoa itoa不是一个标准的c函数，他是windows特有的，跨平台写程序，要用sprintf。)
get_line (gets)

.c_str

stringstream（类似于sprintf sscanf ）

typeid

https://blog.csdn.net/gatieme/article/details/50947821

G:\cherry笔记\typeid.md

static cast dynamic cast

dynamic

算法

双线性差值算法

https://blog.csdn.net/sinat_33718563/article/details/78825971

多线程**pthread**

配置

https://www.cnblogs.com/lzhu/p/12032783.html
https://blog.csdn.net/user11223344abc/article/details/80536280

window

钩子hook

http://blog.sina.com.cn/s/blog_140f86bd70102xlxz.html

https://blog.csdn.net/zi_wu_xian/article/details/8857487

进程

https://blog.csdn.net/DLUTBruceZhang/article/details/9080157?utm_source=copy
https://blog.csdn.net/DLUTBruceZhang/article/details/9058583

https://blog.csdn.net/junbopengpeng/article/details/16830715

https://blog.csdn.net/chance_yin/article/details/8943650

总结：
磁盘　虚拟内存空间　物理内存 进程
1.加载磁盘的exe文件，映射一个4GB的虚拟内存空间，数据段，代码段，堆，栈，文件映射（DLL）

2.同一个动态链接库在物理内存中保留一份，映射到自己的虚拟内存中，当对其中的数据修改
时，产生相应的复制内存，

3.同一时间只有一个进程在运行，CPU只能访问一个虚拟内存空间

4.虚拟内存技术是分页的，物理内存只存储了需要的页数据

dll依赖关系

https://blog.csdn.net/libaineu2004/article/details/89670726

visual studio

报为定义标识符：

把pch.h放到最前面

dll工程建立及调用

https://blog.csdn.net/awyyauqpmy/article/details/80710288

vs2017

MFC窗口的建立

https://blog.csdn.net/qq_36556893/article/details/83178202

离线文档的下载

https://cloud.tencent.com/developer/article/1565141

https://www.cjjjs.com/article/20158313181160

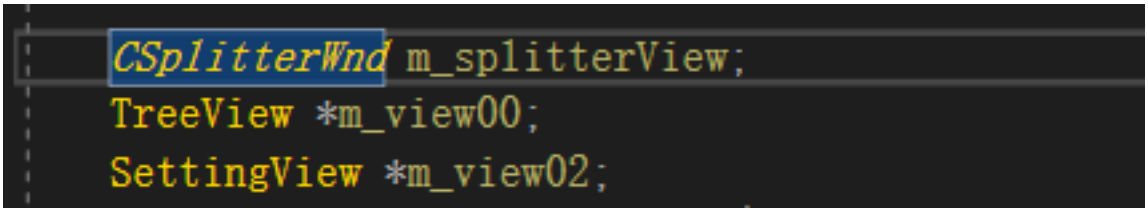
mfc

G:\cherry
笔记\mfcgetitem.md

控件的访问

1.七种基本的访问
2.通过全局变量
3.view 分割视图的访问

- MainFrm类中添加CSplitterWnd成员变量



- 重写虚函数onCreateClient

```
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
};
```

在虚函数onCreateClient中添加静态分割代码

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    // TODO: 在此添加专用代码和/或调用基类

    CRect rc;
    GetClientRect(&rc);
    m_splitterView.CreateStatic(this, 1, 3);

    m_splitterView.CreateView(0, 0, RUNTIME_CLASS(TreeView), CSize(rc.Width()/3, rc.Height()/3), pContext);
    m_splitterView.CreateView(0, 1, RUNTIME_CLASS(CViewTestView), CSize(rc.Width()/3, rc.Height()/3), pContext);
    m_splitterView.CreateView(0, 2, RUNTIME_CLASS(SettingView), CSize(rc.Width()/3, rc.Height()/3), pContext);

    m_view00 = (TreeView *)m_splitterView.GetPane(0, 0);
    m_view02 = (SettingView *)m_splitterView.GetPane(0, 2);
    return true;
    //return CFrameWnd::OnCreateClient(lpcs, pContext);
}
```

tab控件

http://blog.sina.com.cn/s/blog_6163bdeb0102dy9a.html

https://www.shuzhiduo.com/A/QW5Y82DOJm/

https://www.jianshu.com/p/0324359b567c

https://www.bilibili.com/video/av52921336?p=18

使用 tabsheet 扩展封装

```
public:

    virtual BOOL OnInitDialog();
    afx_msg void OnBnClickedButton1();
    CTabSheet m_tab11;
    TabDlg1 m_pag1;
    TabDlg2 m_pag2;
    afx_msg void OnTcnSelchangeTab2(NMHDR *pNMHDR, LRESULT *pResult);
};
```

```
39
40  BOOL DlgOfView::OnInitDialog()
41  {
42      CDialogEx::OnInitDialog();
43
44      // TODO:  在此添加额外的初始化
45      m_tab11.AddPage(TEXT("tab1"), &m_pag1, IDD_TABDIALG1);
46      m_tab11.AddPage(TEXT("tab2"), &m_pag2, IDD_TABDLG2);
47
48      m_tab11.Show();
49
50
51      return TRUE;  // return TRUE unless you set the focus to a control
52                    // 异常: OCX 属性页应返回 FALSE
53  }
54
55
56  void DlgOfView::OnBnClickedButton1()
```

菜单

基于对话框程序添加菜单

https://www.cnblogs.com/zerotoinfinity/p/6382356.html

编辑框

追加文本

```
void MainDlg::OnBnClickedButton1()
{
	// TODO: 在此添加控件通知处理程序代码
	int iLen;
	CString strMsg("");
	g_MainDlg->m_logEdit.GetWindowTextW(strMsg);
	strMsg.Format(_T("%s\r\n%s"),strMsg, _T("hetong"));
	//iLen = g_MainDlg->m_logEdit.GetWindowTextLength();
	//g_MainDlg->m_logEdit.SetSel(iLen,iLen,TRUE);
	//g_MainDlg->m_logEdit.ReplaceSel(strMsg, FALSE);
	g_MainDlg->m_logEdit.SetWindowTextW(strMsg);
}
```

view单文档窗口子窗口

1.view 父类选择 CFormView

2.

添加对话框成员变量

添加消息函数 oncreate

```
53      DlgOfView m_dlgOfView;
54      virtual BOOL OnCreateAggregates();
55      virtual BOOL Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwSty
56      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
```

在oncreate中创建窗口。参数为IDC

```
40  int CForViewTestView::OnCreate(LPCREATESTRUCT lpCreateStruct)
41  {
42      if (CFormView::OnCreate(lpCreateStruct) == -1)
43          return -1;
44
45      // TODO:  在此添加您专用的创建代码
46      m_dlgOfView.Create(IDD_DIALOG1, this);
47      return 0;
48  }
```

3.显示窗口

```
117 {
118     // TODO: 在此添加命令处理程序代码
119     m_dlgOfView.ShowWindow(SW_SHOWNORMAL);
120
121 }
```

视图分割

1.

mainFrm 添加窗口分割对象

```
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
public:

    CSplitterWnd m_splitterView;
    TreeView *m_view00;
    SettingView *m_view02;
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
};
```

2.

添加onclient 虚函数，并分割窗口

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    // TODO: 在此添加专用代码和/或调用基类

    CRect rc;
    GetClientRect(&rc);
    m_splitterView.CreateStatic(this, 1, 3);

    m_splitterView.CreateView(0, 0, RUNTIME_CLASS(TreeView), CSize(rc.Width()/3, rc.Height()/3), pContext);
    m_splitterView.CreateView(0, 1, RUNTIME_CLASS(CViewTestView), CSize(rc.Width()/3, rc.Height()/3), pContext);
    m_splitterView.CreateView(0, 2, RUNTIME_CLASS(SettingView), CSize(rc.Width()/3, rc.Height()/3), pContext);

    m_view00 = (TreeView *)m_splitterView.GetPane(0, 0);
    m_view02 = (SettingView *)m_splitterView.GetPane(0, 2);
    return true;
    //return CFrameWnd::OnCreateClient(lpcs, pContext);
}
```

编码方式

G:\cherry笔记\字符集和编码方式.md

socket

https://blog.csdn.net/ludw508/article/details/8565203

算法题目

平衡字符串

题目描述：
在一个「平衡字符串」中，'L' 和 'R' 字符的

数量是相同的。

给出一个平衡字符串 s，请你将它分割成尽可能多的平衡字符串。

返回可以通过分割得到的平衡字符串的最大数量。

```
示例 1：

输入：s = "RLRRLLRLRL"
输出：4
解释：s 可以分割为 "RL", "RRLL", "RL", "RL", 每个子字符串中都包含相同数量的 'L' 和 'R'。
```

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/split-a-string-in-balanced-strings

```
int balancedStringSplit(char * s){
    int num = 0;
    int j = 0;
    for(int i=0;i<strlen(s);i++)
    {
        if(s[i] == 'R')
        {
            j++;
        }
        else
        {
            j--;
        }
        if(j == 0)
        {
            num++;
        }
    }
    return num;
}
```

数字图像处理

https://www.cnblogs.com/studylyn/p/6613278.html

G:\c++学习\数字图像处理\代码\vc++杨淑莹图像处理

标准库

文件操作

c

fgetc

fputc

fgets

fputs

fprintf

fscanf

c++

<sstream>
<iostream>
<fstream>

<string>
stoi stol stod ... (atoi atol atod)
to_string (itoa itoa不是一个标准的c函数，他是windows特有的，跨平台写程序，要用sprintf。)
get_line (gets)

.c_str

stringstream（类似于sprintf sscanf)

异常处理

1. throw 抛出异常
2. try尝试执行
3.catch捕获异常

throw可以抛出特定类型的异常，try捕获异常后函数退出，catch捕获判断类型然后处理异常

```
int abc(int a, int b, int c)
{
	if (a < 0 && b < 0 && c < 0)
	{
		throw 1;
	}
	else if (a == 0 && b == 0 && c == 0)
	{
		throw 2;
	}
	return  a + b * c;
}

int main()
{
	try { abc(0, 0, 0); }
	catch (char *e)
	{
		std::cout << "catch exception is" << e << std::endl;
	}
	catch (int e)
	{
		std::cout << "catch exception is" << e << std::endl;
	}
	catch (...)
	{
		std::cout << "catch some exception" << std::endl;
	}
}
```