竞赛中C++语言~~不完全~~拾遗

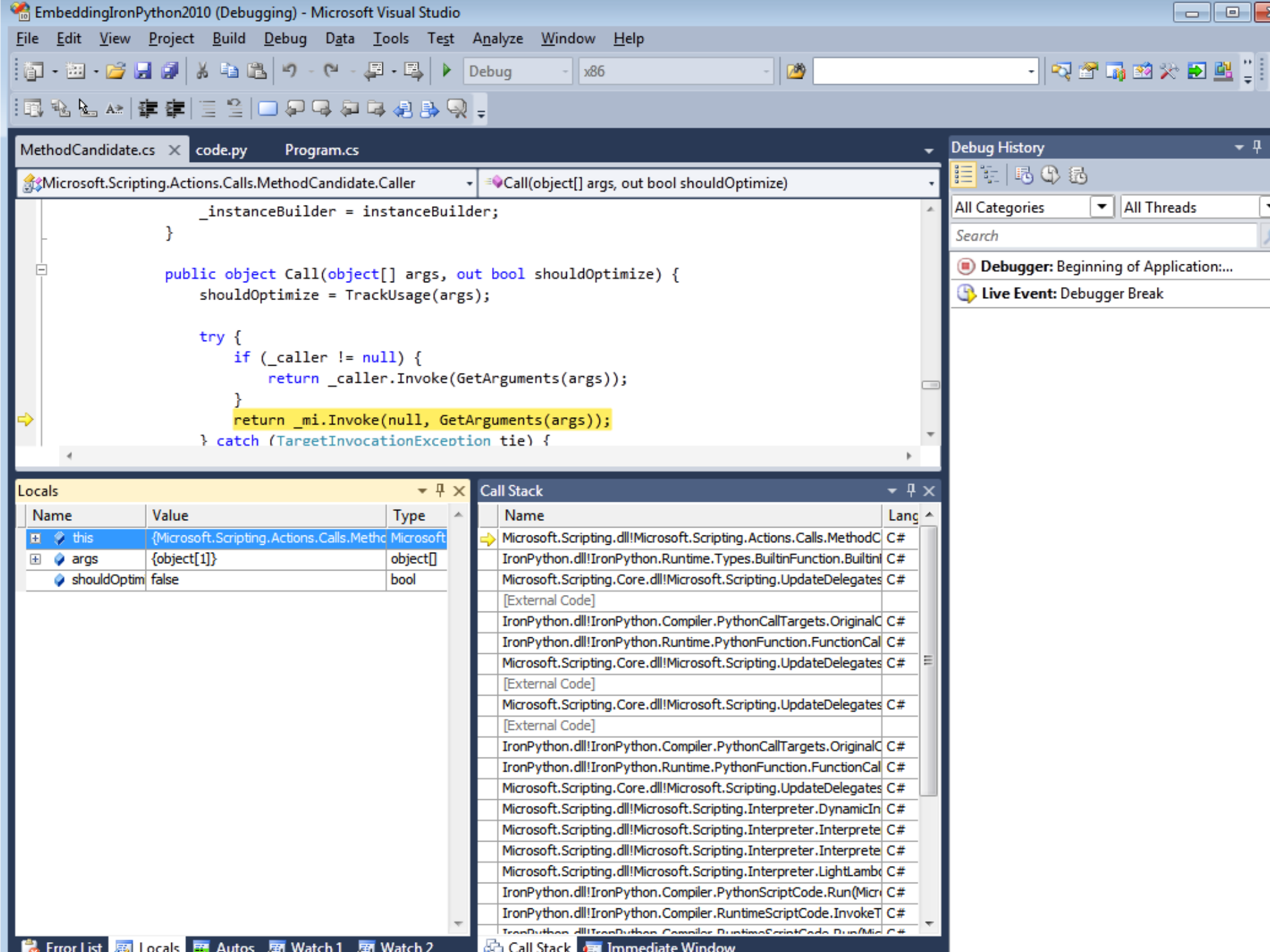
abcdabcd987 @ 2012.08.19

Debug /ˌdi'bʌg/

什么是IDE

- 集成开发环境（Integrated Development Environment，简称 IDE）
- IDE通常包括编程语言编辑器、自动建立工具、通常还包括调试器。有些IDE包含编译器/解释器

- Microsoft Visual Studio
- Dev C++
- Code::Blocks
- Lazarus
- Free Pascal IDE


EmbeddingIronPython2010 (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Debug
x86
MethodCandidate.cs
code.py
Program.cs
Microsoft.Scripting.Actions.Calls.MethodCandidate.Caller
Call(object[] args, out bool shouldOptimize)
_instanceBuilder = instanceBuilder;
}
public object Call(object[] args, out bool shouldOptimize) {
shouldOptimize = TrackUsage(args);
try {
if (_caller != null) {
return _caller.Invoke(GetArguments(args));
}
return _mi.Invoke(null, GetArguments(args));
} catch (TargetInvocationException tie) {
Debug History
All Categories
All Threads
Search
Debugger: Beginning of Application:...
Live Event: Debugger Break
Locals
Name
Value
Type
this
{Microsoft.Scripting.Actions.Calls.Metho
Microsoft
args
{object[1]}
object[]
shouldOptim
false
bool
Call Stack
Name
Lang
Microsoft.Scripting.dll!Microsoft.Scripting.Actions.Calls.MethodC
C#
IronPython.dll!IronPython.Runtime.Types.BuiltinFunction.BuiltinI
C#
Microsoft.Scripting.Core.dll!Microsoft.Scripting.UpdateDelegates
C#
[External Code]
IronPython.dll!IronPython.Compiler.PythonCallTargets.OriginalC
C#
IronPython.dll!IronPython.Runtime.PythonFunction.FunctionCal
C#
Microsoft.Scripting.Core.dll!Microsoft.Scripting.UpdateDelegates
C#
[External Code]
Microsoft.Scripting.Core.dll!Microsoft.Scripting.UpdateDelegates
C#
[External Code]
IronPython.dll!IronPython.Compiler.PythonCallTargets.OriginalC
C#
IronPython.dll!IronPython.Runtime.PythonFunction.FunctionCal
C#
Microsoft.Scripting.Core.dll!Microsoft.Scripting.UpdateDelegates
C#
Microsoft.Scripting.dll!Microsoft.Scripting.Interpreter.DynamicIn
C#
Microsoft.Scripting.dll!Microsoft.Scripting.Interpreter.Interprete
C#
Microsoft.Scripting.dll!Microsoft.Scripting.Interpreter.Interprete
C#
Microsoft.Scripting.dll!Microsoft.Scripting.Interpreter.LightLambd
C#
IronPython.dll!IronPython.Compiler.PythonScriptCode.Run(Micr
C#
IronPython.dll!IronPython.Compiler.RuntimeScriptCode.InvokeT
C#
Error List
Locals
Autos
Watch 1
Watch 2
Call Stack
Immediate Window

Dev-C++ 4.9.9.2 - [tutorial] - tutorial.dev

Archivo Edición Buscar Ver Proyecto Ejecutar Depurar Herramientas CVS Ventana Ayuda

Nuevo ... Insertar ... Añadir/Quitar Ir a

Proyecto | Clases | Depurar

- GLApplication : class
  - IsEnded() : bool
  - IsFullScreen() : bool
  - GetMsgTitle() : inline char *
  - priv LoadTexture(char *fileName) : int
  - priv GetMsElapsed() : long
  - GetScreenPosition() : Point
  - OnEnd() : void
  - OnIdle() : void
  - OnInit(int windowWidht, int windowHeight, ir
  - OnInitGL() : void
  - OnKeyDown(int keyCode) : void
  - OnKeyUp(int keyCode) : void
  - OnLButtonDown(int mouseX, int mouseY)
  - OnLButtonUp(int mouseX, int mouseY) : v
  - OnMouseMove(int mouseX, int mouseY) :
  - OnRButtonDown(int mouseX, int mouseY
  - OnRButtonUp(int mouseX, int mouseY) : v
  - OnRender() : void
  - OnResize(int width, int height) : void
  - priv RenderObjects() : void
  - SetScreenPosition(int x, int y) : void
  - GLApplication() : Constructor
  - ~GLApplication() : Destructor
  - priv m_bEnd : bool
  - priv m_bFullScreen : bool
  - priv m_bWireframe : bool
  - priv m_KeysPressed[256] : bool
  - priv m_WindowTitle[256] : char
  - priv m_bpp : int
  - priv m_IdTexture : int
  - priv m_WindowHeight : int
  - priv m_WindowWidht : int
  - priv m_WindowPosition : Point
- Image : class
- Point : struct
- CreateRenderContext(HWND hWnd, PIXELF

glapplication.cpp

```
///     OnRender: Esta función es invocada cuando el sistema está listo para renderizar.
///
///     @param  nada
///
///     @return  nada
//////////////////////////////////////////////////////////////////////////////////////
void GLApplication::OnRender()
{
    glClearColor( 0.0f, 0.0f, 0.8f, 1.0f );                 // Color de fondo.
    glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT );     // Limpiamos la pantalla.

    glMatrixMode( GL_PROJECTION );                          // Seleccionamos la Matriz de Proyección
    glLoadIdentity();                                       // Reseteamos la Matriz de Proyección

    // Proyección ortogonal
    glOrtho ( 0.0f, (GLfloat)m_WindowWidht,                 // left, right
              0.0f, (GLfloat)m_WindowHeight,                // bottom, top
             -1.0f, 1.0f );                                 // zNear, zFar

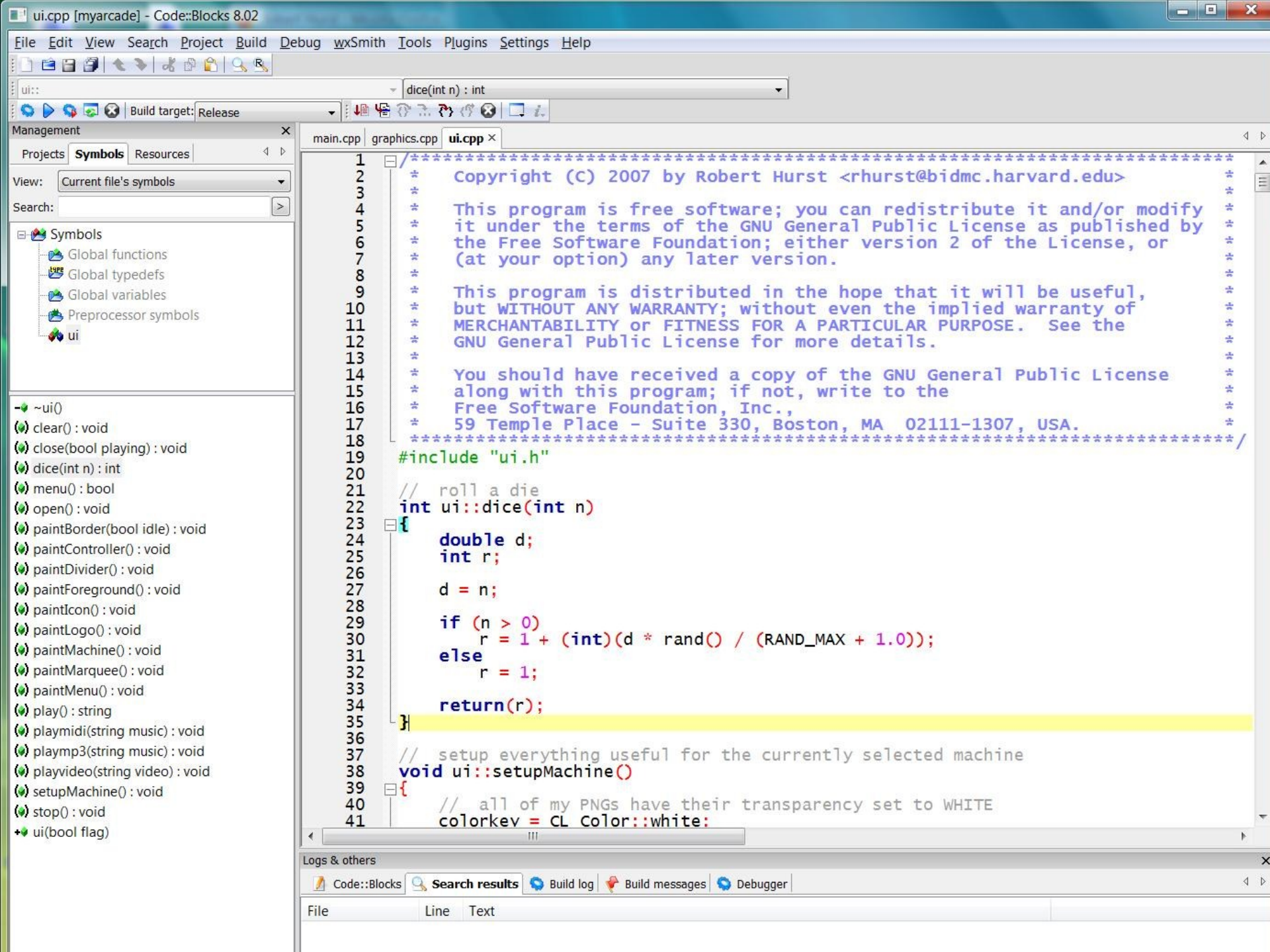
    glMatrixMode( GL_MODELVIEW );                           // Seleccionamos la pila de Matrices Modelo-Vista.
    glLoadIdentity();                                       // Usamos la matriz identidad (no se realiza trans

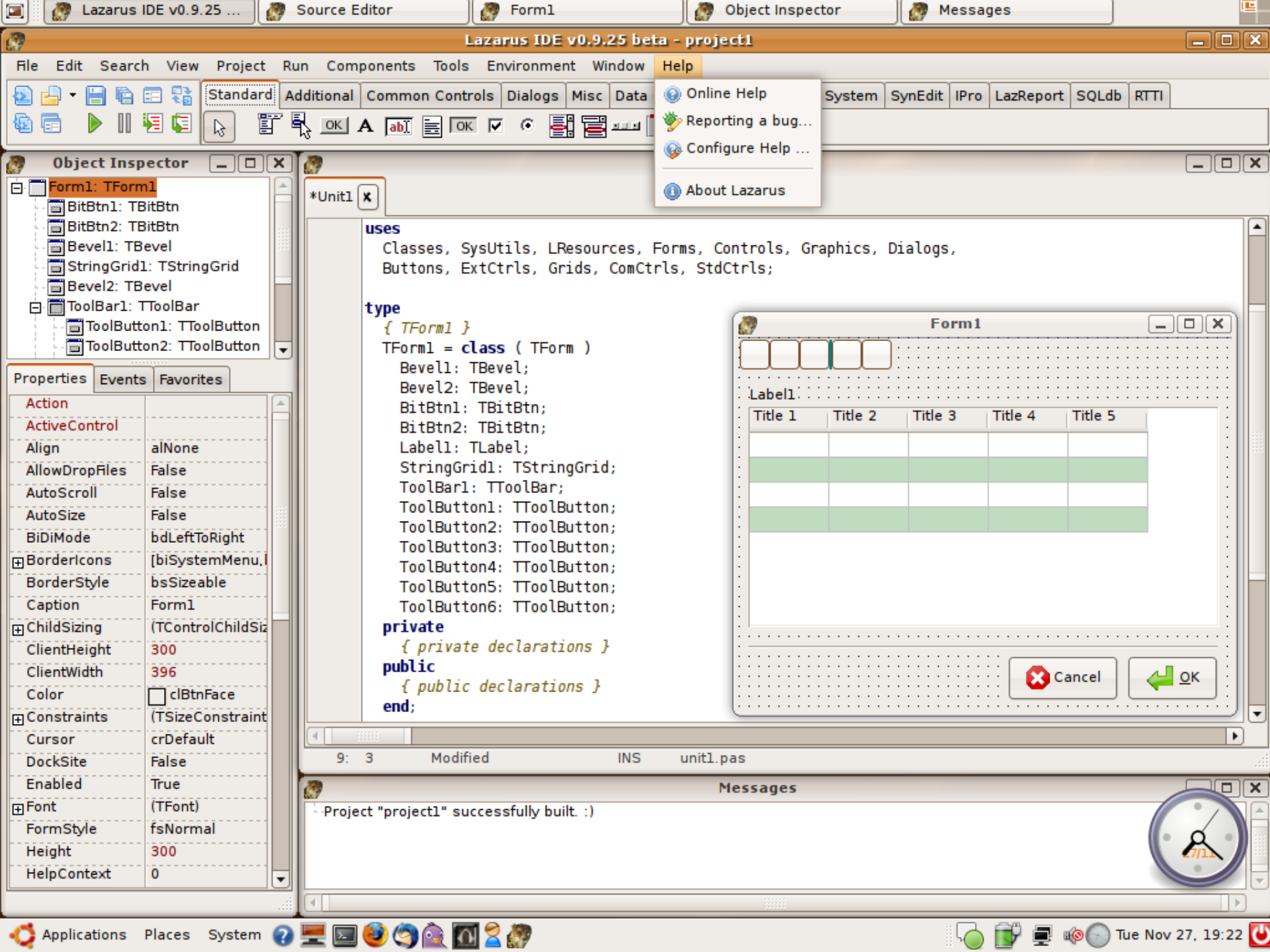
    if ( m_bWireframe )
    {
        glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );        // Renderizado wireframe
        glDisable    ( GL_CULL_FACE  );                     // Desactivamos la selección de caras a render
    }
    else
    {
        glPolygonMode( GL_FRONT_AND_BACK, GL_FILL );        // Renderizado sólido.
        glEnable     ( GL_CULL_FACE );                      // Habilitamos la selección de caras a renderi
        glCullFace   ( GL_BACK      );                      // No se renderizarán las caras traseras.
    }

    RenderObjects();
}


//////////////////////////////////////////////////////////////////////////////////////
///     SetScreenPosition: Fija la posición de la ventana en la pantalla.
```

Compilador | Recursos | Resultado de la compilación | Depurar | Ver Resultados


ui.cpp [myarcade] - Code::Blocks 8.02
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
ui::
dice(int n) : int
Build target: Release
Management
Projects Symbols Resources
View: Current file's symbols
Search:
Symbols
Global functions
Global typedefs
Global variables
Preprocessor symbols
ui
~ui()
clear() : void
close(bool playing) : void
dice(int n) : int
menu() : bool
open() : void
paintBorder(bool idle) : void
paintController() : void
paintDivider() : void
paintForeground() : void
paintIcon() : void
paintLogo() : void
paintMachine() : void
paintMarquee() : void
paintMenu() : void
play() : string
playmidi(string music) : void
playmp3(string music) : void
playvideo(string video) : void
setupMachine() : void
stop() : void
ui(bool flag)
main.cpp graphics.cpp ui.cpp
1 /*****
2 * Copyright (C) 2007 by Robert Hurst <rhurst@bidmc.harvard.edu> *
3 *
4 * This program is free software; you can redistribute it and/or modify *
5 * it under the terms of the GNU General Public License as published by *
6 * the Free Software Foundation; either version 2 of the License, or *
7 * (at your option) any later version. *
8 *
9 * This program is distributed in the hope that it will be useful, *
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of *
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the *
12 * GNU General Public License for more details. *
13 *
14 * You should have received a copy of the GNU General Public License *
15 * along with this program; if not, write to the *
16 * Free Software Foundation, Inc., *
17 * 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. *
18 *****/
19 #include "ui.h"
20
21 // roll a die
22 int ui::dice(int n)
23 {
24 double d;
25 int r;
26
27 d = n;
28
29 if (n > 0)
30 r = 1 + (int)(d * rand() / (RAND_MAX + 1.0));
31 else
32 r = 1;
33
34 return(r);
35 }
36
37 // setup everything useful for the currently selected machine
38 void ui::setupMachine()
39 {
40 // all of my PNGs have their transparency set to WHITE
41 colorkey = CL_Color::white;
Logs & others
Code::Blocks Search results Build log Build messages Debugger
File Line Text


Lazarus IDE v0.9.25 ...
Source Editor
Form1
Object Inspector
Messages
Lazarus IDE v0.9.25 beta - project1
File Edit Search View Project Run Components Tools Environment Window Help
Online Help
Reporting a bug...
Configure Help ...
About Lazarus
Standard Additional Common Controls Dialogs Misc Data System SynEdit IPro LazReport SQLdb RTTI
Object Inspector
Form1: TForm1
BitBtn1: TBitBtn
BitBtn2: TBitBtn
Bevel1: TBevel
StringGrid1: TStringGrid
Bevel2: TBevel
ToolBar1: TToolBar
ToolButton1: TToolButton
ToolButton2: TToolButton
Properties Events Favorites
Action
ActiveControl
Align alNone
AllowDropFiles False
AutoScroll False
AutoSize False
BiDiMode bdLeftToRight
BorderIcons [biSystemMenu,
BorderStyle bsSizeable
Caption Form1
ChildSizing (TControlChildSiz
ClientHeight 300
ClientWidth 396
Color clBtnFace
Constraints (TSizeConstraint
Cursor crDefault
DockSite False
Enabled True
Font (TFont)
FormStyle fsNormal
Height 300
HelpContext 0
*Unit1
uses
Classes, SysUtils, LResources, Forms, Controls, Graphics, Dialogs,
Buttons, ExtCtrls, Grids, ComCtrls, StdCtrls;
type
{ TForm1 }
TForm1 = class (TForm)
Bevel1: TBevel;
Bevel2: TBevel;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
Label1: TLabel;
StringGrid1: TStringGrid;
ToolBar1: TToolBar;
ToolButton1: TToolButton;
ToolButton2: TToolButton;
ToolButton3: TToolButton;
ToolButton4: TToolButton;
ToolButton5: TToolButton;
ToolButton6: TToolButton;
private
{ private declarations }
public
{ public declarations }
end;
Form1
Label1
Title 1 Title 2 Title 3 Title 4 Title 5
Cancel
OK
9: 3 Modified INS unit1.pas
Messages
Project "project1" successfully built. :)
Applications Places System
Tue Nov 27, 19:22

编译器 Compiler /kəm'paɪlə/

- 把某种语言的代码转换为目标平台的目标码

- Pascal:
  - Free Pascal
  - Delphi
- C:
  - GCC(GNU Compiler Collection)
- C++:
  - Microsoft Visual C++
  - G++
- ……

链接器 Linker /'lɪŋkə/

- 将一个或多个由编译器或汇编器生成的目标文件外加库链接为一个可执行文件

- ld(GNU Linker)

编辑器 Editor /'edɪtə/

- Notepad
- Notepad2
- Notepad++
- Gedit
- Vim
- Emacs /ˈiːmæks/

调试器 Debugger /diːˈbʌgə/

- Microsoft Visual Studio Debugger
- GDB(GNU Debugger)

- 大部分IDE的调试功能由GDB提供

- $IDE \approx Compiler + Editor + Debugger$
- 因此几乎可以说，IDE的编译功能不如编译器强，编辑功能不如编辑器强，调试功能不如调试器强

- 福州大学软工机房提供的IDE：
- Microsoft Visual Studio 2005 (Windows)
- Dev-C++ (Windows)
- Free Pascal IDE (Windows)
- Anjuta (Linux)
- GUIDE (Linux)
- Lazarus (Both)

获得G++和GDB (Windows)

- 下载MinGW
  (http://www.sourceforge.net/projects/mingw)
- 如果机子上有Dev-C++、Code::Blocks等IDE的话，一般有G++和GDB

- 建议把g++和gdb所在目录加入%Path%

用G++编译单文件程序

- **$ g++ foo.cpp -o foo -g -Wall**
- foo.cpp 是待编译文件
- -o foo 表示生成的可执行文件名
- -g 表示生成调试信息，方便GDB调试
- -Wall 显示所有警告

用GDB调试程序

- `$ gdb foo`
- 或者
- `$ gdb`
- `(gdb) file foo`

基本

- 在37行添加断点： b(reak) 37
- 查看变量var： p(rint) var
- 单步（不进入）： n(ext)
- 单步（进入）： s(tep)
- 删除所有断点： d(elete)
- 终止程序运行： k(ill)
- 自动查看变量var： disp(lay) var
- 退出GDB： q(uit)

略高级方法

- 当var==x时停在37行：b 37 if (var == x)
- 查看var的二进制形式：p /t var
- 查看var的十六进制形式：p /x var

- ……
- http://blog.csdn.net/haoel/article/details/2879

C++ Syntax /'sɪntæks/

指针 Pointer

- 指向内存中某个类型的变量的类型
- 声明时用*，解引用用*，取地址用&
- 数组即指针
- (*ptr).member == ptr->member

```
int a, b, arr[100] = {123};
int *pa = &a, *pb = &b, *ptr = arr;
*pa = 1, *pb = 2;
cout << *pa << " " << *pb << endl
  << a << " " << b << endl
  << *ptr << " " << *arr << endl
  << *(ptr+3) << " " << *(arr+3) << endl;
```

指针与常量

- `int* p;` p可改，*p可改
- `const int* p;` p可改，*p不可改
- `int* const p;` p不可改，*p可改
- `const int* const p;` p不可改，*p不可改

- 看const在星号前面还是后面

数组模拟版链表

```
struct Edge
{
  int v, w, next;
} g[MAXE];
int header[MAXV];
void AddEdge(const int x, const int y, const int w)
{
  static int LinkSize;
  g[LinkSize].v = y;
  g[LinkSize].w = w;
  g[LinkSize].next = header[x];
  header[x] = LinkSize++;
}

for (int e = header[u]; e; e = g[e].next)
    cout << g[e].v << ' ' << g[e].w << endl;
```

指针版链表

```
struct Edge
{
  int v, w;
  Edge *next;
} g[MAXE], *header[MAXV];
void AddEdge(const int x, const int y, const int w)
{
  static int LinkSize;
  Edge *node = g+(LinkSize++); // Equal to &g[LinkSize++]
  node->v = y;
  node->w = w;
  node->next = header[x];
  header[x] = node;
}

for (Edge* e = header[u]; e; e = e->next)
  cout << e->v << ' ' << e->w << endl;
```

在SAP中两种链表写法

```
int d = SAP(e->v, std::min(delta-sum, e->f));
e->f -= d;
e->op->f += d;

int d = SAP(g[e].v, std::min(delta-sum, g[e].f));
g[e].f -= d;
g[g[e].op].f += d;
```

- 指针简洁，**速度快**，嵌套容易
- 建议：用指针实现链表等结构

动态分配内存

- new和delete用来申请和释放单个变量
- new[]和delete[]用在数组上
- 分配的内存位于堆(Heap)中

```
int *a = new int(10);
string *s = new string("Hello");
int *arr = new int[100];

cout << *a << *s << arr[3] << arr[31];

delete a;
delete s;
delete [] arr;
```

系统分配内存池

```
struct Edge
{
  int v, w;
  Edge *next;
  Edge(const int node, const int weigh, Edge* const succ):
    v(node), w(weigh), next(succ) { }
} *header[MAXV];
void AddEdge(const int x, const int y, const int w)
{
  Edge *node = new Edge(y, w, header[x]);
  header[x] = node;
}

for (int u = 0; u < MAXN; ++u)
  for (Edge *e = header[u]; e; e = e->next)
  {
    Edge* const bak = e;
    delete bak;
  }
```

手动分配内存池

```
struct Edge
{
  int v, w;
  Edge *next;
} g[MAXE], *header[MAXV];
void AddEdge(const int x, const int y, const int w)
{
  static int LinkSize;
  Edge *node = g+(LinkSize++);
  node->v = y;
  node->w = w;
  node->next = header[x];
  header[x] = node;
}

for (Edge* e = header[u]; e; e = e->next)
  cout << e->v << ' ' << e->w << endl;
```

- 动态分配麻烦，不正确地delete，或者错用delete[]会造成内存泄漏
- 所以能手动分配的手动分配，不能的用vector等封装好的容器

模板 Template

```
template<typename T>
  inline T sqr(const T x)
{
  return x*x;
}

template<typename T1, typename T2>
  struct Pair
{
  T1 first;
  T2 second;
  Pair(const T1& a, const T2& b): first(a), second(b) { }
};

const long double delta = sqr(B)-4*A*C;
const Pair<int, int> fraction(3, 4);
```

类 Class

- struct和class关键字都可以声明一个类，区别在于：struct默认标号为public，class默认标号为private
- class MyClass
- {
- //不可以被非成员函数访问到的东西
- public:
- //可以被非成员函数访问到的东西
- private:
- //不可以被非成员函数访问到的东西
- };
- struct MyClass
- {
- //可以被非成员函数访问到的东西
- private:
- //不可以被非成员函数访问到的东西
- public:
- //可以被非成员函数访问到的东西
- };

类的构造函数 Constructor

```
struct Point
{
  int x, y;
  Point(const int a, const int b): x(a), y(b) { }
};

Point p1(1, 2);                    //OK!
Point p2;                          //Error!
Point parr[10];                    //Error!
```

- 错误：对‘Point::Point()’的调用没有匹配的函数

默认参数

```
struct Point
{
  int x, y;
  Point(const int a = 0, const int b = 0):
    x(a), y(b) { }
};

Point p1(1, 2);                    //OK!
Point p2;                          //OK!
Point parr[10];                    //OK!
```

- 可以用于任何函数，但是必须在形参表的最后面，并且赋给的值必须为常量或者静态变量

静态变量 static

- 和全局变量一样放在静态存储区，自动被初始化，不占用栈的空间，但是和局部变量一样有作用域限制

```
void AddEdge(const int x, const int y, const int w)
{
  static int LinkSize;
  Edge *node = g+(LinkSize++);
  node->v = y;
  node->w = w;
  node->next = header[x];
  header[x] = node;
}
```

Habit && Optimization
/ˈhæbɪt/ /ˌɑptɪmaɪˈzeɪʃən/

Effective C++ Item02

- Prefer consts, enums, and inlines to #defines

- #define PI 3.14
- const long double PI = 3.14

- 用#define的话出错的信息非常难以理解，因为编译器会告诉你3.14错了而不是PI错了

Effective C++ Item02

- `#define PLUS(A, B) A+B`
  - `3*PLUS(1*2,2*3) // ERROR! 3*1*2+2*3 = 12`
- `#define PLUS(A, B) ((A)+(B))`
  - `3*PLUS(1*2,2*3) // 3*((1*2)+(2*3)) = 24`
- `#define MAX(A, B) ((A)>(B)? (A) : (B))`
  - `a = MAX(++x, ++y)`
    `// if x=10, y=3 => x=12, y=4, a=12`

```
inline int plus(const int a, const int b)
  { return a+b; }
inline int max(const int a, const int b)
  { return a > b ? a : b; }
```

Effective C++ Item02

- `#define pointer int*`
- `const pointer p;  // const int* p`
  - ERROR! p可变，*p不可变
- `pointer const p; // int* const p`
  - p不可变，*p可变

- `typedef int* pointer;`
- `const pointer p; // int* const p`
- `pointer const p; // int* const p`

- `typedef int Array[100];`
- `Array a, b, c;`

Effective C++ Item03

- Use const whenever possible

1. 编译器可以优化
2. 防止错误

```
void foo(int a, int b, int c)
{
  if (c = a*b) // OK, but I want if (c == a*b)
    ...
}

void foo(const int a, const int b, const int c)
{
  if (c = a*b) // ERROR!
    ...
}
```

- 错误：向只读形参‘c’赋值

Effective C++ Item03

```
class Bigint
{
  int _data[MAXLEN];
  //...
public:
  int& operator[](const int index) { return _data[index]; }
  const int operator[](const int index) const { return _data[index]; }
  //...
};
```

Effective C++ Item20

- Prefer pass-by-reference-to-const to pass-by-value

```
struct Matrix
{
  int a[100][100];
  //...
};
void foo1(Matrix m) { }
void foo2(const Matrix m) { }
void foo3(Matrix& m) { }
void foo4(const Matrix& m) { } // Prefer This
```

- 使用const可以防止修改并且允许字面值，&可以防止复制

Effective C++ Item53

- Pay attention to compiler warnings

```
void foo(int a, int b, int c)
{
  if (c = a*b)
    ...
}
```

- 警告：建议在作用真值的赋值语句前后加上括号 [-Wparentheses]

More Effective C++ Item02

- Prefer C++-style casts

- C风格转换：
  - (T) expr
- C++风格转换：
  - static_cast<T>(expr)　　最常用，如double -> int型
  - const_cast<T>(expr)　　去掉const，如const int -> int
  - dynamic_cast<T>(expr)　　与类的继承有关
  - reinterpret_cast<T>(expr)　　与编译平台有关
- C风格的抓换仍被保留，可以处理 static_cast const_cast reinterpret_cast 能处理的转型
- C++风格转型操作符多，分类明确。优点一是语义明确，对程序员和对编译器都很友好；二是方便编译器查错
- const long double average = static_cast<long double>(sum)/n;

More Effective C++ Item06

- Distinguish between prefix and postfix forms of increment and decrement operators

- i++ 后自增式 postfix i+1并且返回原来的i
- ++i 前自增式 prefix i+1并且返回新的i

- 一般后自增式会有额外的开销，即备份原来的数值，因此在能使用前自增式的时候尽量使用前自增式
- 但是对于内置类型（比如int）编译器会自动作出优化

```
for (int i = 0; i < n; i++) ...          // OK
for (int i = 0; i < n; ++i) ...          // recommended
for (vector<int>::iterator iter = v.begin(); iter != v.end(); iter++)
  ...                                    // OK, but NOT recommended
for (vector<int>::iterator iter = v.begin(); iter != v.end(); ++iter)
  ...                                    // recommended
```

操作符重载建议

- 只将会改变第一个参数的值的操作符(如：+=)定义为成员函数，而将返回一个新对象的操作符(如：+)定义为非成员函数(并使用 += 来实现)。
- 对一元操作符，为避免隐式转换最好将其重载为成员函数。
- 对二元操作符，为能在左操作数上能进行和右操作数一样的隐式转换，最好将其重载为非成员函数。
- 重载 operator[] 之类的操作符，应尽量提供 const 版本和非 const 版本。

Trick /trɪk/

离散化

```
int a[MAXN];
vector<int> v;

for (int i = 0; i < n; ++i)
{
  cin >> a[i];
  v.push_back(a[i]);
}

sort(v.begin(), v.end());
v.resize(unique(v.begin(), v.end()) - v.begin());

for (int i = 0; i < n; ++i)
  a[i] = lower_bound(v.begin(), v.end(), a[i]) - v.begin();
```

std::ostream_iterator

- 使用std::ostream_iterator和std::copy，可以实现一句话输出迭代器范围内的元素

```
#include <iostream>  // for cin
#include <iterator>  // for ostream_iterator  /ɪtə'reɪtə/
#include <algorithm> // for copy             /ˈælgərɪðəm/
int a[MAXN];
vector<int> v;

copy(a, a+n, ostream_iterator<int>(cin, " "));
cin << endl;
copy(v.begin(), v.end(), ostream_iterator<int>(cin, "\n"));
```

memset

- 正无穷： `memset(a, 0x3F, sizeof(a))`
- 负无穷： `memset(a, 0xC0, sizeof(a))`
- -1： `memset(a, 0xFF, sizeof(a))`
- 自定义长度： `memset(a,.., sizeof(*a)*n)`

- 原理：
- int为32位有符号整型，最高位为1时为负数，否则为正数，数值大小为其二进制的补码
- 1个十六进制数可以表示4个二进制位
- memset0x3F，每个数会变成0x3F3F3F3F，即1061109567
- memset0xC0，每个数会变成0xC0C0C0C0，即-1061109568
- 有个好处，两数相加不会溢出：
- $0x3F3F3F3F * 2 = 2122219134 < 2^{31}-1$
- $0xC0C0C0C0 * 2 = -2122219136 > -2^{31}$

字符串转型

- C++没有提供字符串到数值类型如int，double等的转换，但是可以利用stringstream来完成

```
#include <string>   // for string
#include <sstream>  // for stringstream
#include <iostream> // for cin
template<typename T>
  T Convert(const string& s)
{
  stringstream ss(s);
  T res;
  ss >> res;
  return res;
}

string s1("123");
const char* s2 = "3.14";
cout << Convert<int>(s1) << ' ' << Convert<long double>(s2);
```

End of Line?? eoln ??

- C++和C中都没有提供类似Pascal中eoln的东西，所以如果需要读到行末，比较方便的方法就是用stringstream

```
#include <string>   // for string
#include <sstream>  // for stringstream
#include <iostream> // for cin

string s;
getline(cin, s);
stringstream ss(s);
for (n = 0; ss >> a[n]; ++n)
  //...;
```

双关键字排序

```
struct Point
{
  int x, y;
  Point(const int a = 0, const int b = 0): x(a), y(b) { }
};
inline bool operator<(const Point& lhs, const Point& rhs)
{ return lhs.x < rhs.x || (lhs.x == rhs.x && lhs.y < rhs.y; }

Point points;
sort(points, points+n);
```

Trap /træp/

std::pair

- std::pair和std::make_pair十分方便，甚至连比较操作都有。但是由于某种未知的原因，使用std::pair和std::make_pair会比自己写一个Pair来的慢
- 因此建议手写Pair
- 代码在前几页中有

std::set::erase

- std::set std::multiset std::map std::multimap
  都是红黑树写的，在插入和删除节点的时候会使迭代器失效

```
for (set<int>::iterator iter = s.begin(); iter != s.end(); ++iter)
  if (condition to delete s)
    s.erase(iter); //ERROR! iter doesn't exist after erase, can’t ++iter
```

```
for (set<int>::iterator iter = s.begin(); iter != s.end(); )
  if (condition to delete s)
    s.erase(iter++); //OK!
  else
    ++iter;
```

using namespace std

- 这句话会引入std::的所有东西

- ```
  #include <iostream>
  using namespace std;
  int left, right;
  int main()
  {
    cin >> left >> right;
    //...
  }
  ```
- 错误：对‘left’的引用有歧义

- 原因是在ios_base.h中有一个函数叫做left
  ```

using namespace std

- 解决办法：

- 不引入
- ```
  #include <iostream>
  int left, right;
  int main()
  {
    std::cin >> left >> right;
  }
  ```

- 或者using需要的东西
- ```
  #include <iostream>
  using std::cin;
  int left, right;
  int main()
  {
    cin >> left >> right;
  }
  ```

#include <cmath>

- 引入cmath后，会发现如x1，y1，x2，y2等**全局变量**不起作用了，原因是cmath中有同名函数

- `#include <cmath>`

- `int x1, y1, x2, y2;`
- `int main()`
- `{`
- `  x1 = 0, y1 = 1;`
- `  x2 = 2, y2 = 3;`
- `}`

- 错误：‘int y1’被重新声明为不同意义的符号

- 解决办法：更改变量名或者改成局部变量

string.h

- 由于历史原因，C++选择保留了C的头文件，并且新建了一组新的头文件，去掉.h并以c开头，把C的头文件包含在std::中
- math.h => cmath
- stdio.h => cstdio
- string.h => cstring
- ...
- 而C++新的头文件无后缀
- string
- iostream
- algorithm
- ...

- 强烈建议使用cstring等替换string.h等头文件，因为有命名空间的保护，在不引入std::的情况下不容易出错，而且cstring等在细节上有对C++进行修改

cstring && string

- #include <cstring> ??
- #include <string> ??

- cstring也就是旧的string.h，包含C风格字符串（字符数组）的处理函数
- strcmp
- strcpy
- ...
- 也包含一些对内存的操作
- memset
- memcpy
- ...

- string是C++的std::string类（C++风格字符串）的头文件，包含了std::string类的声明和实现，两种字符串不同

C风格字符串&& C++风格字符串

- C风格字符串也就是字符数组：
  - `char *`
  - `const char *`
- 不支持< == > !=等逻辑操作
- 不支持=赋值操作
- 上面操作需要使用cstring头文件中相应的函数
- 支持`scanf("%s", s)`
- 支持`cin >> s`
- 需要设置缓冲区大小，防止溢出
  - `const int BUFSIZE = 1024;`
  - `char s[BUFSIZE];`

C风格字符串&& C++风格字符串

- C++风格字符串也就std::string是一个类
- 各种操作随意搞
- 不支持scanf("%s", s)
- 支持cin >> s
- 动态大小，类似vector

Reference

Reference

- *Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition)*

  Scott Meyers
- *More Effective C++: 35 New Ways to Improve Your Programs and Designs*

  Scott Meyers
- *The C++ Standard Library: A Tutorial and Reference*

  Nicolai M. Josuttis
- *C++ Primer (4th Edition)*

  Stanley B. Lippman, Josée Lajoie, Barbara E. Moo
- *C++操作符重载手册*

  http://www.adintr.com/myarticle/operator.html
- *Standard Template Library Programmer's Guide*

  http://www.sgi.com/tech/stl/
- *C++ Reference*

  http://www.cplusplus.com

THE END

Thanks!