中国矿业大学计算机学院

系统软件开发实践报告

课程名称 系统软件开发实践

实验名称 实验八 Flex/Bison 综合实验四

学生姓名 胡钧耀

学　　号 06192081

专业班级 计算机科学与技术 2019-4 班

任课教师 张博

成绩考核

编号	课程教学目标	占比	得分
1	**目标 1：**针对编译器中词法分析器软件要求，能够分析系统需求，并采用 FLEX 脚本语言描述单词结构。	15%	
2	**目标 2：**针对编译器中语法分析器软件要求，能够分析系统需求，并采用 Bison 脚本语言描述语法结构。	15%	
3	**目标 3：**针对计算器需求描述，采用 Flex/Bison 设计实现高级解释器，进行系统设计，形成结构化设计方案。	30%	
4	**目标 4：**针对编译器软件前端与后端的需求描述，采用软件工程进行系统分析、设计和实现，形成工程方案。	30%	
5	**目标 5：**培养独立解决问题的能力,理解并遵守计算机职业道德和规范，具有良好的法律意识、社会公德和社会责任感。	10%	
总成绩			

指导教师		评阅日期	

目　录

实验（八） Flex/Bison 综合实验四

8.1 实验要求与目标

使用 flex 和 bison 开发一个具有全部功能的桌面计算器，能够支持变量，过程，循环和条件表达式，使它成为一个虽然短小但具有现实意义的编译器。

学习抽象语法树的用法，它具有强大而简单的数据结构来表示分析结果。

8.2 实验内容

计算器具体需要实现的功能：变量命名；实现赋值功能；实现比较表达式（大于、小于、等于等等）；实现 if/then/else 和 do/while 的流程控制；用户可以自定义函数；简单的错误恢复机制。

8.3 实验思路

向 Android 项目添加 C 和 C++代码，只需将相应的代码添加到项目模块的 cpp 目录中即可。在构建项目时，这些代码会编译到一个可由 Gradle 与应用打包在一起的原生库中。然后，Java 代码即可通过 Java 原生接口(JNI)调用原生库中的函数。

8.4 实验步骤

8.4.1 下载 NDK 和构建工具

打开项目后，依次点击【Appearance & Behavior > System Settings > Android SDK】。点击 SDK Tools 标签页。选中 NDK (Side by side)和 CMake 复选框。安装结果如下。项目会自动同步 build 文件并执行构建。

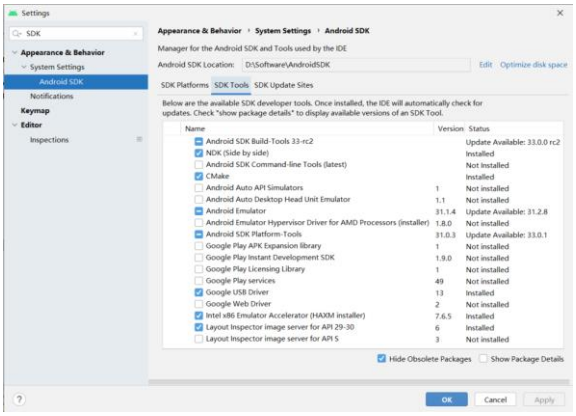

图 1 下载 NDK 和构建工具

8.4.2 创建支持 C/C++的新项目

创建支持原生代码的新项目的步骤与创建任何其他 Android Studio 项目的步骤相似，但前者还需要执行一个额外的步骤。

在向导的 Choose your project 部分中，选择 Native C++项目类型，点击 Next。填写向导下一部分中的所有其他字段，点击 Next。选择 Toolchain Default 可使用默认的 CMake 设置。点击 Finish。在 Android Studio 完成新项目的创建后，请从 IDE 左侧打开 Project 窗格，然后选择 Android 视图。如下所示，Android Studio 会添加 cpp 组。

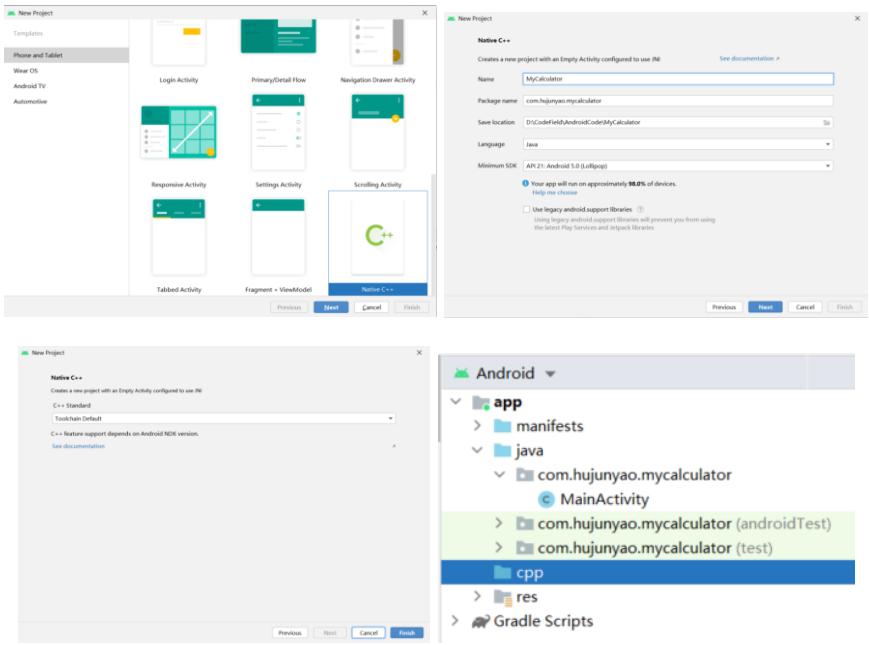

图 2 创建支持 C/C++ 的新项目

在 cpp 组中，可以找到项目中的所有原生源代码文件、头文件、CMake 或 ndk-build 的构建脚本，以及项目中的预构建库。对于新项目，Android Studio 会创建一个示例 C++源代码文件 *native-lib.cpp*，并将其置于应用模块的 src/main/cpp/目录中。此示例代码提供了一个简单的 C++函数 `stringFromJNI()`，它会返回字符串“Hello from C++”。

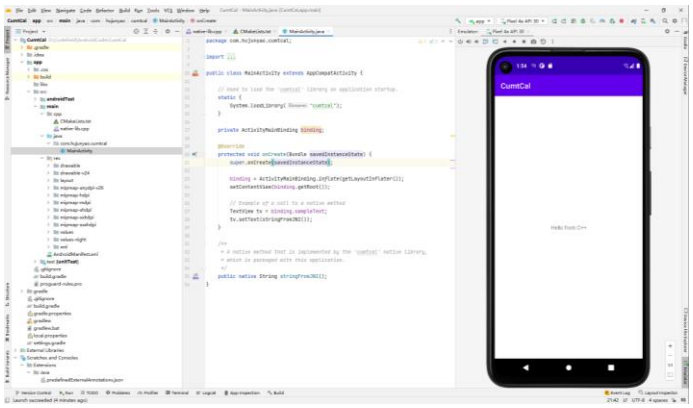

图 3 运行样例文件得到 hello 输出

8.4.3 Flex 与 Bison 源代码调整和重新编译

首先直接把上次实验三的源文件复制备用，命名为 *fb3-4.h*、*fb3-4.l*、*fb3-4.y*、*fb3-4funcs.c*，同时记得修改文件内的引用，把所有的 *3-3* 改为 *3-4*。

给 *3-4.h* 加上声明语句。

```
char* result;
```

给 *3-4.l* 加上声明语句。

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

修改*fb3-4.y*的生成式。

```
calclist: /* nothing */
  | calclist stmt ';' EOL {
     sprintf(result,"%4.4g",eval($2));
     treefree($2);
    }
  | calclist LET NAME '(' symlist ')' '{' list '}' EOL {
                        dodef($3, $5, $8);
                        sprintf(result, "Defined %s .", $3-
>name); }
 ;
%%
```

输入如下命令，导出*fb3-4.tab.c*、*fb3-4.tab.h*和*fb3-4.lex.c*。

```
bison -d -o fb3-4.tab.c fb3-4.y
flex -ofb3-4.lex.c fb3-4.l
```

给*fb3-4.lex.c*加上声明语句和`calc()`方法。

```
void yyparse();

char* calc(const char* expr)
{
    size_t len = strlen(expr);
    result = (char*)malloc(sizeof(char)*(len+2));
    yy_switch_to_buffer(yy_scan_string(expr));
    yyparse();
    return result;
}
```

删除的*fb3-4funcs.c*的`main()`方法，同时对`calc()`方法进行声明。

```
extern char* calc(char*);
```

然后对结果进行编译。

```
bison -d fb3-4.y
flex -ofb3-4.lex.c fb3-4.l
```

为了编译成*.so*文件调用函数，在*fb3-4.lex.c*中增加`char* calc(char*)`函数，代码如下。

```
char* calc(const char* expr)
{
    size_t len = strlen(expr);
    result = (char*)malloc(sizeof(char)*(len+2));
    yy_switch_to_buffer(yy_scan_string(expr));
    yyparse();
    return result;
}
```

然后把 *fb3-4.h*、*fb3-4.lex.c*、*fb3-4.tab.c*、*fb3-4.tab.h*、*fb3-4funcs.c* 导入 *cpp/cal* 文件夹。

8.4.4 创建/导入新的 C/C++源代码文件

从 IDE 的左侧打开 Project 窗格，然后从下拉菜单中选择 Project 视图。右键点击 cpp 目录，然后依次选择【New > C/C++ Source File】，或者导入文件，这里直接导入实验三的三个*.c* 文件和两个*.h* 文件。

向项目中添加新的 C/C++文件后，仍需要配置 CMake 以将这些文件包含在原生库中。下面对 cpp 文件夹下的 *CMakeLists.txt* 进行修改。添加 CMake 命令来配置构建脚本，如需指示 CMake 根据原生源代码创建原生库，构建脚本添加`cmake_minimum_required()`和`add_library()`命令。代码如下。

```
cmake_minimum_required(VERSION 3.7.0)
add_library(
        cumtcal
        SHARED
        cal/native-lib.cpp
        cal/fb3-4funcs.c
        cal/fb3-4.lex.c
        cal/fb3-4.tab.c
        )
find_library(
        log-lib
        log)

target_link_libraries(
        cumtcal
        ${log-lib})
```

8.4.5 修改原生 C++文件

完成上述步骤后 *Calculator.java* 的`calc()`函数为红色，查看提示如下，需要修改 JNI 函数，即需要修改创建工程时自带的样例文件 *native-lib.cpp*。

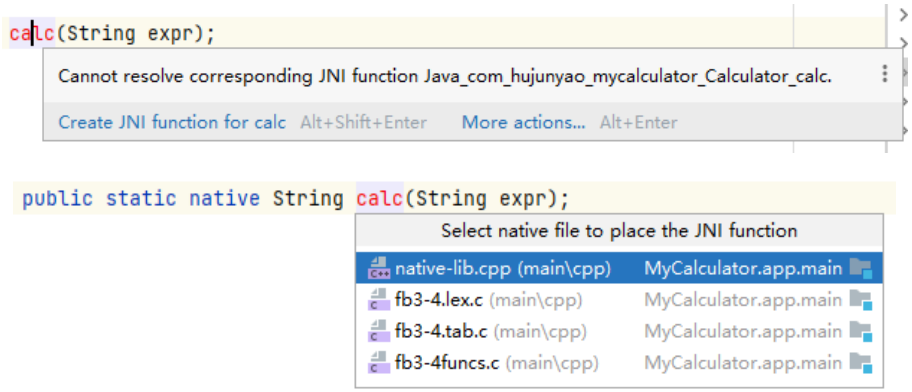

图 4 修改原生 C++文件

模仿样例文件，修改代码如下。

```
#include <jni.h>
#include <string>
#include <cstring>
#include <cstdio>
using namespace std;
extern "C" char* calc(const char* expr);
extern "C"
JNIEXPORT jstring JNICALL
Java_com_hujunyao_cumtcal_MainActivity_calc(JNIEnv *env,
jclass thiz, jstring input) {
    const char *input_char = (env)->GetStringUTFChars(input,
nullptr);
    string output = calc(input_char);
    return env->NewStringUTF(output.c_str());
}
```

8.5 手势识别实现过程

具体的实现过程见本人的百度飞桨项目 PaddleClas 实现数字手势识别。

8.5.1 数据集制作

由于本人对计算机视觉比较感兴趣，想实现通过分析手势来输出对应的数字，借鉴 Github 上已有的数据集 Sign Language Digits Dataset，但是外国的手势和中国的略有不同，不够本土化，因此自己采取了中国北方普遍使用的数字手势制作数据集，文件打包至 *hjy-gesture.zip*。

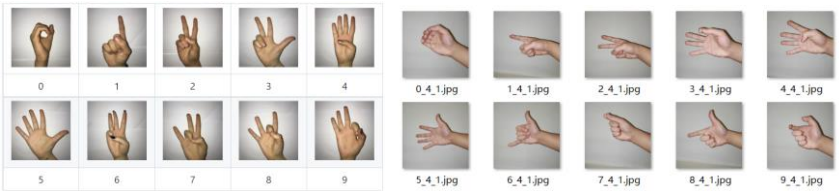

图 5 国外数字手势（左）与中国常用数字手势（右）对比

本数据集每张图像大小为长宽 128 像素，颜色通道为 RGB，共有数字 0-9 十个类别，每个数字自行拍摄五张光线强弱、手势方位不同的照片，通过随机亮度、随机平移、随机缩放生成 640 张照片，十个数字共 6400 张照片。

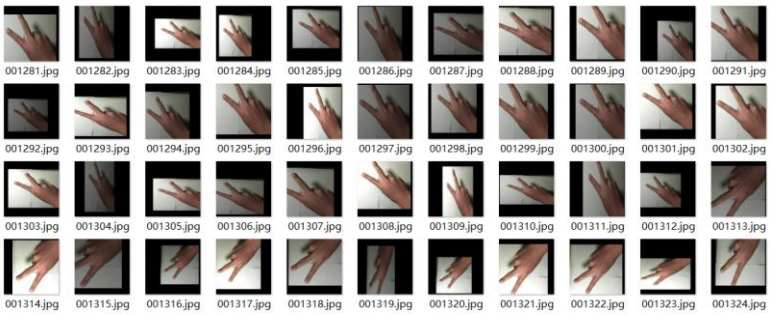

图 6 数字二的自制数据集

生成数训练集和验证集的代码如下。

```
# 分割为训练集 test 和验证集 val
# 得到总的训练数据
def read_all_files():
    all_list = []
    i = 0 # 标记总文件数量
    j = 0 # 标记文件类别
    for root,dirs,files in os.walk(config['dirpath']): # 分别代表根目录、文件夹、文件
        for file in files:
            j = int(root.split('/')[-1]) + 1
            i = i + 1
            # 文件中每行格式： 图像相对路径 图像的 label_id（数字类别）（注意：中间有空格）。
            imgpath = os.path.join(root,file)
            all_list.append(imgpath+" "+str(j)+"\n")
    allstr = ''.join(all_list)
    with open('all_list.txt','w',encoding='utf-8') as f:
        f.write(allstr)
    return all_list , i

all_list,all_lenth = read_all_files()
print('总训练数据有{}条'.format(all_lenth))

# 把数据打乱
all_list = shuffle(all_list)
allstr = ''.join(all_list)
with open('all_list.txt','w',encoding='utf-8') as f:
    f.write(allstr)
```

```
print("打乱成功，并重新写入文本")

# 按照比例划分数据集
train_size = int(all_lenth * config['train_ratio'])
train_list = all_list[:train_size]
val_list = all_list[train_size:]

print('训练集大小为{}, 验证集大小为{}'.format(len(train_list),
len(val_list)))

# 生成训练集 txt
train_txt = ''.join(train_list)
with open('train_list.txt','w',encoding='utf-8') as f_train:
    f_train.write(train_txt)
    print("train_list.txt 生成成功! ")

# 生成验证集 txt
val_txt = ''.join(val_list)
with open('val_list.txt','w',encoding='utf-8') as f_val:
    f_val.write(val_txt)
    print("val_list.txt 生成成功! ")
```

此步完成后将打乱数据，训练集大小为 5120，验证集大小为 1280，并生成对应上图片名称及其标签的 *train_list.txt* 和 *val_list.txt*。

8.5.2 PaddleClas 训练

PaddleClas 可以快速使用 PaddleClas 中全量 134 个模型进行预测，包括 ResNet、HRNet、ResNeSt、MobileNetV1/2/3、GhostNet 等，模型训练时需要修改相应的参数，主要是以下几点：分类数、图片总量、训练和验证的路径、图像尺寸、数据预处理、训练和预测的 num_workers: 0，修改的路径在 *PaddleClas/ppcls/configs/quick_start/new_user/ShuffleNetV2_x0_25.yaml*。其中，采用的网络是 ResNet50。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 7 ResNet50 基本结构

```
# global configs
Global:
  checkpoints: null
  pretrained_model: null
  output_dir: ./output/
  # 使用 GPU 训练
  device: gpu
  # 每几个轮次保存一次
  save_interval: 1
  eval_during_train: True
  # 每几个轮次验证一次
  eval_interval: 1
  # 训练轮次
  epochs: 50
  print_batch_step: 10
  use_visualdl: True #开启可视化（目前平台不可用）
  # used for static mode and model export
  # 图像大小
  image_shape: [3, 100, 100]
  save_inference_dir: ./inference
  # training model under @to_static
  to_static: False

# model architecture
Arch:
  # 采用的网络
  name: ResNet50
  # 类别数 0-9 对应数字 0-9
  class_num: 10

# loss function config for traing/eval process
Loss:
  Train:

    - CELoss:
        weight: 1.0
  Eval:
    - CELoss:
        weight: 1.0


Optimizer:
  name: Momentum
  momentum: 0.9
```

```
  lr:
    name: Piecewise
    learning_rate: 0.001
    decay_epochs: [30, 60, 90]
    values: [0.1, 0.01, 0.001, 0.0001]
  regularizer:
    name: 'L2'
    coeff: 0.0005


# data loader for train and eval
DataLoader:
  Train:
    dataset:
      name: ImageNetDataset
      # 根路径
      image_root: ./dataset/
      # 前面自己生产得到的训练集文本路径
      cls_label_path: ./dataset/digit_sign/train_list.txt
      # 数据预处理
      transform_ops:
        - DecodeImage:
            to_rgb: True
            channel_first: False
        - ResizeImage:
            resize_short: 100
        - CropImage:
            size: 100
        - RandFlipImage:
            flip_code: 1
        - NormalizeImage:
            scale: 1.0/127.0
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''

    sampler:
      name: DistributedBatchSampler
      batch_size: 32
      drop_last: False
      shuffle: True
    loader:
      num_workers: 0
      use_shared_memory: True
```

```
  Eval:
    dataset:
      name: ImageNetDataset
      # 根路径
      image_root: ./dataset/
      # 前面自己生产得到的验证集文本路径
      cls_label_path: ./dataset/digit_sign/val_list.txt
      # 数据预处理
      transform_ops:
        - DecodeImage:
            to_rgb: True
            channel_first: False
        - ResizeImage:
            resize_short: 100
        - CropImage:
            size: 100
        - NormalizeImage:
            scale: 1.0/127.0
            mean: [0.485, 0.456, 0.406]
            std: [0.229, 0.224, 0.225]
            order: ''
    sampler:
      name: DistributedBatchSampler
      batch_size: 32
      drop_last: False
      shuffle: True
    loader:
      num_workers: 0
      use_shared_memory: True

Infer:
  infer_imgs: ./dataset/digit_sign/0/IMG_5950.jpg
  batch_size: 10
  transforms:
    - DecodeImage:
        to_rgb: True
        channel_first: False
    - ResizeImage:
        resize_short: 100
    - CropImage:
        size: 100
    - NormalizeImage:
        scale: 1.0/127.0
```

```
        mean: [0.485, 0.456, 0.406]
        std: [0.229, 0.224, 0.225]
        order: ''
    - ToCHWImage:
  PostProcess:
    name: Topk
    # 输出的可能性最高的前 topk 个
    topk: 5
    # 标签文件 需要自己新建文件
    class_id_map_file: ./dataset/label_list.txt

Metric:
  Train:
    - TopkAcc:
        topk: [1, 5]
  Eval:
    - TopkAcc:
        topk: [1, 5]
```

8.5.3 PaddleLite 模型保存与部署

上面的步骤完成后，百度飞桨平台会给模型生成*.pdopt* 和*.pdparams* 参数文件，分别存储了优化器参数和模型参数。使用如下命令可以将其保存为*.pdmodel* 模型文件。

```
!python3 tools/export_model.py \
    -c ./ppcls/configs/quick_start/new_user/ \
    ShuffleNetV2_x0_25.yaml \
    -o Global.pretrained_model=output/ResNet50/best_model \
    -o Global.load_static_weights=False \
    -o Global.save_inference_dir=./model/
```

Paddle Lite 是一组工具，可帮助开发者在移动设备、嵌入式设备和 IoT 设备上运行模型，以便实现设备端机器学习。主要特性是支持多平台：涵盖 Android、iOS、嵌入式 Linux 设备等支持多种语言：包括 Java、Python、C++。还有轻量化和高性能：针对移动端设备的机器学习进行优化，压缩模型和二进制文件体积，高效推理，降低内存消耗。使用如下命令生成可以在安卓手机端使用的*.nb* 文件。部署的方法参考了官方 demo 程序及其文档。

```
!paddle_lite_opt \
    --model_file=model/inference.pdmodel \
    --param_file=model/inference.pdiparams \
    --valid_targets=arm \
    --optimize_out_type=naive_buffer \
    --optimize_out=model/model
```

8.6 文字识别实现过程

天行数据是致力于为个人和企业用户提供更标准、简洁、方便、高效的 API 接口平台。天行数据平台陆续开发上线了 220 余款接口，产品包括新闻资讯、微信生态、生活服务、娱乐应用、金融科技、知识问答、数据智能等七大类型的接口服务，内容涵盖互联网的各个方面。

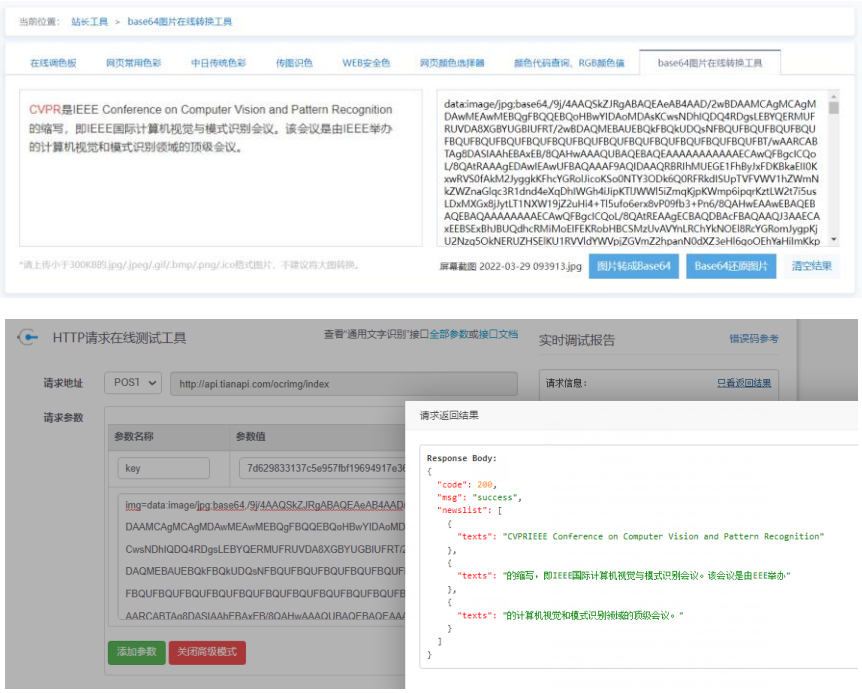

图 8 天行数据文字识别 API 接口测试

8.6.1 图片转 base64

主要代码如下。在 `onActivityResult()`函数中，如果接收到 resultCode 是 RESULT_OK，同时 requestCode 是 REQUEST_CODE_DOCUMENT，且 data 非空，执行如下操作，从相册获取照片，转换为 base64，然后使用 `httpPost()` 方法进行下一步转换文字操作，获取文字后，显示在相应的部件上。主要代码如下所示。

```
try {
    ContentResolver resolver = getContentResolver();
    Uri uri = data.getData();
    Bitmap image = MediaStore.Images.Media.getBitmap(resolver,
uri);
    String[] proj = {MediaStore.Images.Media.DATA};
    Cursor cursor = managedQuery(uri, proj, null, null, null);
    cursor.moveToFirst();

    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    image.compress(Bitmap.CompressFormat.JPEG, 100, baos);
```

```
    baos.flush();
    baos.close();
    byte[] bitmapBytes = baos.toByteArray();
    String base64 = Base64.encodeToString(bitmapBytes,
Base64.DEFAULT);
    String result = PostMethod.httpPost(base64);
    EditText et = (EditText) findViewById(R.id.et_input);
    et.setText(res2input(result));
} catch (IOException e) {
    Log.e("emmm", e.toString());
}
```

8.6.2 base64 转文字

`httpPost()`方法实现了输入一个图片的 base64 编码，通过调用天行数据网站的 API 接口，返回图片上的文字，从而达到快速输入函数定义的效果。主要使用了 HttpURLConnection 库。

```
public class PostMethod {
    public static void main(String[] args) throws IOException
{
        String base64string = "xxx";
        String res = httpPost(base64string);
        System.out.println(res);
    }

    public static String httpPost(String base64string) throws
IOException {
        StrictMode.ThreadPolicy policy=new
StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);
        URL url = new
URL("http://api.tianapi.com/ocrimg/index");
        Map<String, Object> params = new LinkedHashMap<>();
        params.put("key", "7d629833137c5e957fbf19694917e36f");
        params.put("img", base64string);
        //开始访问
        StringBuilder postData = new StringBuilder();
        for (Map.Entry<String, Object> param :
params.entrySet()) {
            if (postData.length() != 0) postData.append('&');
            postData.append(URLEncoder.encode(param.getKey(),
"UTF-8"));
            postData.append('=');
```

```
postData.append(URLEncoder.encode(String.valueOf(param.getValu
e()), "UTF-8"));
        }
        byte[] postDataBytes =
postData.toString().getBytes(StandardCharsets.UTF_8);
        HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
        conn.setRequestProperty("Content-Length",
String.valueOf(postDataBytes.length));
        conn.setDoOutput(true);
        conn.getOutputStream().write(postDataBytes);
        Reader in = new BufferedReader(new
InputStreamReader(conn.getInputStream(),
StandardCharsets.UTF_8));

        StringBuilder sb = new StringBuilder();
        for (int c; (c = in.read()) >= 0; )
            sb.append((char) c);
        return sb.toString();
    }
}
```

8.7 运行效果

程序主要运行效果截图如下。

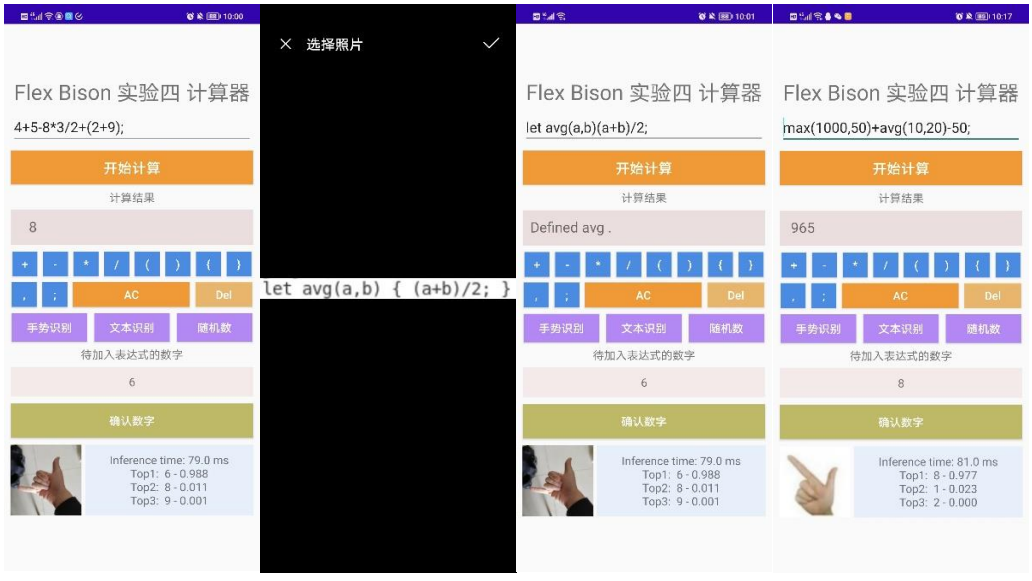

图 9 程序操作截图

8.8 实验总结

8.8.1 问题

实验期间遇到了很多问题，如修改后的文件无法编译，或者编译为*.so* 库后就马上闪退，期间咨询了陆玺文学长等人，通过分析，降低软件配置版本以及修改部分代码最后成功运行。此外，仔细阅读文档也能帮助自己解决问多问题。特别是使用飞桨部署模型的过程，大多数博客写的不是很详细，但是文档的步骤很清楚。

8.8.2 评价

本次实验主要实现了基于 flex 和 bison 的计算器，同时还加入了手势识别和文字识别的拓展，丰富了程序的内容。但是由于手势识别的数据集数量有限，以及模型训练迭代次数较少，部署到手机后的识别率较低，还有待提升，但模型可以落地实现还是花了很长时间的。

8.8.3 收获

实验四完成了将 Flex 与 Bison 部署在安卓端的步骤，期间搜集了大量资料，对于安卓调用 C、C++文件有了直观的体验和感受。也正是在实际编写软件过程中，发现了许多细节上的不到位，比如对于输入不符合语法的错误处理等，还需要进一步加强。同时，我也了解到了百度飞桨平台这样的 AI 在线平台对于人工智能的学习有很大帮助，在线 GPU 节省了很多费用和精力，自己也可以多花信息了解了解。