コンパイラ構成

課題2：codegen コード生成器

情報工学系
権藤克彦

課題の概要

- XCのコード生成器を実装する
- すること
  - xcc-bison-dist-64以下のcodegen.c または codegen-skel.c を修正・拡張して，codegen () 関数を実装
  - 具体的にはレベル1〜3を解く（どこまでできたか明記）
    - レベル1：test/kadai1.c （大域変数，代入,=,<,+,-式）
    - レベル２：test/kadai2.c（if,if-else,return文，==,&&,*,/式）
    - レベル３：test/kadai3.c（ポインタ型，単項演算子*,&, ポインタ演算）
  - 拡張課題（オプション）
- 基本的にMac（CSC）かLinuxで動作させること
  - それ以外の場合は，レポートで明示すること
- 締切：8/6（火）17:00
  - ∠切厳守．1秒でも遅れたら受け取らない．

実装する関数の型

```
void codegen (void);    // コード生成を実装する関数
struct AST *ast_root;   // ASTの根ノードへのポインタ
```

- codegen()は，xcc.y中のmain()関数が呼ぶ
- その前に，大域変数 ast_root がセット
  - ast_root は，xcc.y に定義あり

拡張課題（オプション）

- 外付けのボーナス点（最大15点）を与える
- 内容は任意．以下は例.
  - char型を実装（tmisc-char.c, tmisc-char2.c）
  - XCに含まれないC言語の文法のコード生成を加える.
- 拡張課題をした場合は
  - レポート中で拡張課題があることをタイトルに明記する
  - 拡張課題のソースコードはxcc-bison-dist-64とは別に提出する（ディレクトリごと別にする）
  - 拡張した機能をテストする入力プログラムも提出する
    - なるべく多くの場合を試すテストが，良いテスト.
    - 短く単純なテストから，複雑なテストまで，いろいろあると良い.

codegen-skel.cとcodegen.c

- 好きな方を使って良い
  - codegen.cは，一部のコード生成を実装済み
  - codegen-skel.cは，コード生成部分は空っぽ
- codegen-skel.cを使う際は，codegen.cにファイル名変更する
- codegen.cにはレベル1～ 3の実装をまとめて書く（ファイルをレベルごとに別にしない）

Flexライブラリとのリンクに失敗したら
Makefile中の LIBS = -ll を要編集

実行してみよう

```
% cd xcc-bison-dist-64
% make clean （念の為，不要なファイルや xcc を削除）
% make   （Makefileの中身に従って，自動コマンド実行）
bison -d -v xcc.y
flex xcc.l
gcc -Wall -Wstrict-prototypes -Wmissing-prototypes -
Wmissing-declarations -g -o xcc AST.c type.c symbol.c misc.c
codegen.c xcc.tab.c lex.yy.c
（2つ警告が出るが気にしなくて良い）
% ./xcc test/kadai0.c > kadai0.s
% gcc kadai0.s
% ./a.out                  // Linux では，kadai0.s 中の _main を main に
hello, world, 10, 20      // _printf を printf に要変更
%
```

```
int printf ();
int main ()
{
    printf ("hello, world, %d, %d\n", 10, 20);
}
```

レベル1：和

- long型の大域変数
- 代入式(=)
- while文
- 2項演算子（<, +, -）

kadai1.c

```
int printf ();
long i;
long sum;
int main()
{
  i = 5;
  sum = 0;
  while (0 < i) {
    printf ("i = %ld\n", i);
    sum = sum + i;
    i = i - 1;
  }
  printf ("sum = %ld\n", sum);
}
```

mrnという名前に特に意味はない

レベル2：簡単な再帰関数（1/2）

- long型の局所変数と関数引数
- if文，if-else文，return文
- 2項演算子（==, ||, &&, *, /）

kadai2.c

```
int printf ();
long mrn (long n) {
    if (n < 0 || n == 0)  return 0;
    else       return 10 * mrn (n - 1) + n;
}
int main () {
  long i; i = 0;
  while (i < 11) {
    printf ("mrn(%ld) = %ld\n", i, mrn(i));
    i = i+1;
  }
}
```

レベル2：簡単な再帰関数（2/2）

```
% ./a.out
mrn(0) = 0
mrn(1) = 1
mrn(2) = 12
mrn(3) = 123
mrn(4) = 1234
mrn(5) = 12345
mrn(6) = 123456
mrn(7) = 1234567
mrn(8) = 12345678
mrn(9) = 123456789
mrn(10) = 1234567900
```

レベル3 ：バブルソート (1/2)

- ポインタ型
  - 大域変数，局所変数，関数引数，関数の返り値
- 単項演算子（*, &）
- ポインタ演算
  - ポインタ型の式に対する2項演算（+, -）

レベル3：バブルソート (2/2)

- 実行には引数が1つ必要（ソートするデータ個数）
  - 例：./a.out 10

kadai3.c

```
void bubble_sort (long *data, long size)
{
    long i; long j;
    i = size - 1;
    while (0 < i) {
        j = 0;
        while (j < i) {
            if (*(data + (j+1)) < *(data + j))
                swap (data + j, data + (j + 1));
            j = j + 1;
        }
        i = i - 1;
    }
}
```

data[j]と同じ

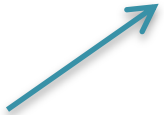
&data[j]と同じ

他の関数は略

提出方法

- T2SCHOLAの課題一覧から提出.

- 提出物一式（次スライド参照）をtarで1つのファイルに固めて，アップロードすること.
  - 学籍番号と同じディレクトリを作り，そこに送るファイルをすべてコピーする.
    % mkdir 22B12345
    % cp ファイル名 22B12345
  - tar で1つのファイルにまとめる.
    % tar cvzf 22B12345.tgz 22B12345



このファイルを
アップロード.

提出物

ファイル分割あり

- プログラム一式
  - codegen.cを含めて，xcc-bison-dist-64以下のファイルすべて
  - ただし，バイナリファイル，*.sは含めないこと
    - make clean すると不要なファイルを消せる
- レポート
  - report.txt または report.pdf
    （図が必要な場合は PDF形式の report.pdf中に含めること）

カンニングはダメ

- アイデアレベルでの議論はＯＫ．
- ソースコードを見るのはカンニングと見なす．
- 警告：剽窃チェッカーを使います．
  見せた方も同罪．不合格にします

- 見せない努力も必要．
  - ソースコードを不用意に印刷して放置しない．
  - ソースコードを表示したまま，席を離れない．
  - ファイルやディレクトリの他人の読み取りを不許可にする．
  - 議論する際にソースコード（疑似コード）を使わない．
  - デバッグを助けてもらう時もソースコードは見せない．
  - Github等でパブリックに公開しない．

入力と出力のお約束

- こちらでテストするために必要
- 守らなかった場合は減点

- XCCは入力ファイル名を argv[1] から受け取る
- XCCはコンパイル結果を標準出力に出力

```
% ./xcc test/kadai0.c > kadai0.s
%
```

デバッガを使うべし

データ構造が複雑なので，
printfデバッグでは大変

```
%  lldb ./xcc
(lldb) b codegen    ブレークポイント設定
Breakpoint 1: where = xcc`codegen + 26 at codegen.c:297,
address = 0x0000000100002c3a
(lldb) r test/kadai0.c  実行開始
-> 297      ast = search_AST_bottom (ast_root,
"AST_translation_unit_single", NULL);
(lldb) print *ast_root  データ表示
(AST) $0 = {
  ast_type = 0x0000000100008829 "AST_translation_unit_pair"
  (lldb) print *ast_root->child[0]  データ表示
(AST) $1 = {
  ast_type = 0x000000010000880d "AST_translation_unit_single"
(lldb) quit  デバッガ終了
%
```