C−− Language

Tsan-sheng Hsu

tshsu@iis.sinica.edu.tw

`http://www.iis.sinica.edu.tw/~tshsu`

Definition

- **The C−− language is a subset of the standard C language.**
- **Its purpose is to act like a universal intermediate language.**
- **C−− is a STACK based language.**
- **A C−− program consists of the following parts.**
  - **#define MAX_S maximun_stack_size**
    - ▷ ***allocate the size of the STACK.***
  - **#include "cmm.c"**
    - ▷ ***this line is required and the file "cmm.c" contains system defined functions and variables.***
  - **procedure_1**
  - **procedure_2**
  - **. . .**
  - **procedure_n**

Procedure Definition

- **Each procedure_i is a standard C procedure without parameters.**
  - **procedure_i**
  - **{**
  - **...**
  - **}**
- **Procedure_1 must be *main*.**
- **The first statement of *main* is INIT__S();**
- **Inside each procedure, the followings rules are enforced.**
  - **No variable declaration is allowed.**
  - **All operations are integer-based (32-bit).**
  - **Constants are zero, positive or negative integers.**
  - **Ten global 32-bit integer variables can be used, they are R__0, ..., R__9.**
    - ▷ ***These variables are called* registers.**

Statements

- Each line contains exactly one statement.
- Null statement — blank lines containing white spaces.
- comments of the form
  $/ * \cdots * /$
- STACK oriented operations.
- Assignment statements.
- A C label of the form
  label:
- Jump statements.
- I/O statements.
- Procedure call statements
  - procedure_i();

STACK operations

- INIT__S();
  - used only in the first statement of $main$.
  - Initialize the stack.
- register = TOP__S();
  - returns the current stack pointer.
  - Initial value is 0.
- register = VAL__S(i);
  - returns the value at stack pointer $+i$.
- SETSP__S(i);
  - set new stack pointer to be current stack pointer $+i$.
- SSET__S(i,k);
  - set the value at stack pointer $+i$ to k.
- PUSH__S(k);
- register = POP__S();
- Note that i and k are registers or constants.

Assignment statements

- **register = (register | constant)** $(+|-|*|/|\%)$ **(register | constant);**
- **register = (register | constant);**

Jump statements

- **Conditional jump**
  - **if '(' (register | constant)** $(> \mid < \mid == \mid >= \mid <=)$ **0 ')' goto label;**
- **Unconditional jump**
  - **goto label;**

I/O statements

- **Read an interger into a register**
  - **scanf("%d",&register);**
- **Print an integer, stored in a register, and a space**
  - **printf("%d ",register);**
- **Print a string**
  - **printf("string");**
- **Print a newline**
  - **printf("\n");**

A Sample C−− program

```
#define MAX__S 10000
#include "cmm.c"
main()
{
        INIT__S();
        R__0 = 1;
        scanf("%d",&R__1);
        if(R__1 <= 0) goto done;
        PUSH__S(R__1);
/* compute factorial */
        factorial();
compute:
        R__1 = POP__S();
        R__1 = R__1 - 2;
        if(R__1 <= 0) goto done;
        PUSH__S(R__1);
        R__0 = R__0 * R__1;
        goto compute;
```

```
done:
        printf("%d ",R__0);
        printf("\n");
}
factorial()
{
        R__2 = 1;
loop:
        R__3 = POP__S();
        if(R__3 == 0) goto ends;
        R__2 = R__2 * R__3;
        R__3 = R__3 - 1;
        PUSH__S(R__3);
        goto loop;
ends:
        PUSH__S(R__2);
}
```

The file “cmm.c”

```
#include <stdio.h>
#define S__TYPE int  /* stack element type */
S__TYPE *STACK__S; /* stack */
int SP__S; /* stack pointer */
 /* registers */
S__TYPE R__0,R__1,R__2,R__3,R__4,R__5,R__6,R__7,R__8,R__9;

/* initial stack */
void INIT__S(void)
{
  STACK__S = (int *) malloc(sizeof(S__TYPE) * (MAX__S+1));
  SP__S = 0;
}

/* return top of stack pointer */
S__TYPE TOP__S(void)
{
  return(SP__S);
```

```
}

/* returns the value at stack pointer + i */
S__TYPE VAL__S(i)
S__TYPE i;
{
  return(STACK__S[SP__S+i]);
}

/* set new stack pointer to be current stack pointer $+ i$ */
void SETSP__S(i)
S__TYPE i;
{
  SP__S += i;
}

/* set the value at stack pointer $+ i$ to $k$ */
void SSET__S(i,k)
S__TYPE i,k;
{
```

```
  STACK__S[SP__S+i] = k;
}

/* push k into stack */
void PUSH__S(k)
S__TYPE k;
{
  SP__S += 1;
  STACK__S[SP__S] = k;
}

/* pop from stack */
S__TYPE POP__S(void)
{
  return(STACK__S[SP__S--]);
}
```