# Java to C Translator & C Garbage Collector

CIS 706 – Spring 2016

# Design Strategies

- String Templates for converting Java constructs to C constructs
- The Java to C code generator
- Run time exception handling.
- Reuse code from AST, symbol tree and type checking from pervious packages
- Customized Static Library in C.
- Garbage Collector.

# String Template

- Java Classes -> C structures
  - simpleClass(name,members) ::= <<
    typedef struct <name>{
    <members>
    }<name>;<\n>
    >>
- Variable Declarations and Field Declarations
  - FieldDecl(type, name) ::= "<type> <name>;<\n>"
- Field Declarations initialized
  - FieldDeclInit(type,name,val) ::= "<type> <name> = <val>;<\n>"
- Method Declarations
  - methodDecl(name, returnType,args) ::= "<returnType> <name> (<args; separator=\", \">);<\n>”
- Method Definitions
  - methodDef(name, returnType, args, body) ::= <<
    <returnType> <name> (<args; separator=", ">){
    <body><\n>
    }<\n>
    >>

# String Template(Cont.)

- Main Method
  - mainMethod(body) ::= <<int main(int argc, char *argv[]){
    <body><\n>
    return 0;
    }<\n>
    >>
- Assignment Statements
  - assignStmt(lhs,rhs) ::= "<lhs> = <rhs>;<\n>"
- IfElse statements and IfStatements
  - ifelseStmt(cond,thenBlock, elseBlock) ::= <<
    if(<cond>){
    <thenBlock>
    }
    else{
    <elseBlock>
    }<\n>
    >>
  - ifStmt(cond,block) ::= <<
    if(<cond>){
    <block>
    }<\n>
    >>

# String Template(Cont.)

- While Statement
  - whileStmt(cond,block) ::= <<

```
while(<cond>){
        <block>
}<\n>
>>
```

- DoWhile Statements
  - dowhileStmt(cond,block) ::= <<
  - do{
  - <block>
  - }while(<cond>);<\n>
  - >>

- For Statements
  - forStmt(init,term,upd,block) ::= <<

```
    for(<init>;<term>;<upd>){
        <block>
    }<\n>
>>
```

- Invoke, return and indec statements
  - invokeStmt(name,args) ::= "<invokeExp(name,args)>;<\n>"
  - returnStmt(exp) ::= "return <exp>;<\n>"
  - incDecStmt(exp,op) ::= "<postExp(exp,op)>;<\n>"

# String Template(Cont.)

- Expressions
  - binExp(exp1,op,exp2) ::= "<exp1> <op> <exp2>"
  - unaryExp(exp,op) ::= "(<op><exp>)"
  - postExp(exp,op) ::= "<exp><op>"
  - parenExp(exp) ::= "(<exp>)"
  - litExp(exp) ::= "<exp>"
  - idExp(exp) ::= "<exp>"
  - invokeExp(name, args) ::= "<name>(<args>)"
  - arrayAccessExp(array,index) ::= "<array>[<index>]"
  - fieldAccessExp(structName,field) ::= "<structName>-><field>"
  - condExp(cond, ifExp, elseExp) ::= "<cond>?<ifExp>:<elseExp>"
  - newStruct(name) ::= "((<name>*)malloc(sizeof(<name>)))"
  - newArray(type,nElems) ::= "((<type>*)malloc((<nElems>)*sizeof(<type>)))"
  - initArray(name,index,val) ::= "<name>[<index>] = <val>;<\n>"

# JAVA TO C CODE GENERATOR

- A class file that has a generate method.
  - Takes the Compilation Unit, Extended Symbol Table, Extended Type Checker as input.
  - Returns an object of CprogramCode. CProgramCode contains the following stringbuilders:
    - typedefDecls -- constructs that declare structures as typedefs
    - structs – constructs that declare structures.
    - globalVars – global variable declarations (static variables in Java)
    - methodDecls – Method Declaration constructs.
    - methodDefs – Method Definition constructs.
- Visitor methods that extend and override ASTVisitor.
- Test class:
  - Takes the CprogramCode object and writes the stringbuffers into target.c.
  - target.c is then compiled and executed.
  - Tests pass if they execute without error
  - Tests fail if the do not compile or have runtime exceptions
  - Run time exceptions are handled as explained in the next slides.

# Exception Handling

- Type of Exceptions handled:
  - Index Array Out of Bounds
  - Null Pointer Exceptions
  - Divide By Zero Exception
  - Can handle more.
- For every statement maintain a data structure. We use a LinkedHashSet to maintain the order of the if expressions.
  - At array access or field access or a division expression put if else expressions in the linkedHashSet.
  - At the Expression Statement Visitor embed the statement into the if else constructs from the linkedHashSet.
  - Before exiting the visit(ExpressionStatement node) reinitialize the LinkedHashSet.

# Exception Handling Sample Code

```
if(a!=NULL && i < 3){
        if(b!=NULL && a[i] * i < 3){
                if(b!=NULL && a[i] < 3){
                        if(a!=NULL && b[a[i]] < 3){
                                b[a[i] * i] = a[b[a[i]]];
                        }else{
                                printf("Error:Index Out Of Bounds near line %d\n",__LINE__);
                                exit(-1);
                        }
                }else{
                        printf("Error:Index Out Of Bounds near line %d\n",__LINE__);
                        exit(-1);
                        }
        }else{
                printf("Error:Index Out Of Bounds near line %d\n",__LINE__);
                exit(-1);
                }
}else{
        printf("Error:Index Out Of Bounds near line %d\n",__LINE__);
        exit(-1);
        }
```

# Static Library for C

- Write a c file equivalent to StaticLibJava and create a library called staticlib.o. We include staticlib.h in our c source code and always run the c code with the following command:

  *–gcc –o target target.c staticlib.o*

# JVM RESULT >< GCC RESULT

- Currently we translate Java code to ”target.c”. If it compiles and executes without error we pass the test. If there is an error we display it and fail the test.
- In the future (before submission) we plan to execute the Java code and the C code and compare the result. If they match the test will pass if not it will fail. We will do this mostly to check exception messages.

# DEMO

# Garbage Collector

- A Large Chunk of memory (512 MB) which acts as our virtual heap.
- Allocate blocks from this virtual heap.
- Mark and Sweep to free memory when memory runs out.
- macro toggle logging.

# GARBAGE COLLECTOR APPROACH

## DATA STRUCTURES USED

- A structure to maintain the metadata about each block that is being allocated. We call this structure Object.
  - The mark bit (to indicate whether this block is free or occupied).
  - A pointer to the start address of the block.
  - A pointer to the end address of the block.
  - A pointer to the address of the pointer the block is being allocated to, in the program.
  - Total of 13 bytes.
- A structure called ‘Reference’:
  - to hold the pointer to ‘Object’
  - a pointer to the next ‘Reference’.
  - Total of 8 bytes.
- A structure called ‘referenceList’ to keep a record of all blocks that are allocated.
  - holds a pointer to the head ‘Reference’.
  - Size of the ‘Reference’ linked list.
- A structure called ‘freeList’ to keep a record of all blocks that are free.
  - Holds a pointer to the head ‘Reference’
  - Size of the linked list.

# GARBAGE COLLECTOR APPROACH(CONT.)

## MEMORY MANAGEMENT

- An “allocate” function that returns a pointer to a block of memory.
  - If referenceList.head == NULL returns a pointer to the beginning of the our virtual heap.
  - If referenceList.head != NULL returns a pointer to the end address of last allocated block.
  - We allocate memory from the beginning of our virtual heap and we keep the referenceList at the end of the virtual heap. When the allocated blocks begin to flow into the referenceList
    - detect ”out of memory”
    - Mark and Sweep.
    - Look for block of appropriate size in freeList.
    - If found return pointer to the startAddress else report “out of memory error”.
- A “gc_malloc” function that will be called from the source code.
  - Takes as arguments
    - size of block
    - address of the pointer this block will be allocated to.
  - Calls “allocate”. Creates the meta-data (“Object”) at the pointer returned by ”allocate”.
    - Computes start address.
    - Computes end address.
    - Sets mark bit
    - Sets the address of the pointer that was passed as argument.
  - Adds the address of the meta-data to referenceList.
  - Returns start address.

# Mark and Sweep

- Mark
  - iterate through the referenceList
  - mark all objects that are not pointing to NULL.
  - Takes O(n) time.
- Sweep
  - iterate through the referenceList
  - If object not marked, remove from referenceList and add to freeList.
  - Iteration takes O(n).
  - Adding/removing to/from referenceList and freeList is O(1).

# Toggle Logging

- Demonstrate without logging
- Demonstrate with logging

# Example of 5 nodes Link List

- Demonstrate running program
- Show the program line by line
- Describe algorithm with picture

# 5 Nodes Circular Link List

- Demonstrate running program
- Show the which part handle circular list
- Describe algorithm with picture

# Allocation and Deallocation

- Shows the number of object
- Demonstrate running program

# Things to do before Submission

- Change the malloc in string templates to our own gc_malloc
- Maybe change singly linked lists to doubly linked lists for better performance.
- Try and garbage collect all children of an object if the parent object is made null.
- Fix the bug in sweep() when adding to free list.

# THANK YOU

**QUESTIONS**