Explanation of TAC Generation for miniMatlab

Design Of Grammar Augmentations

Non terminals M, N and O have been used for augmentation. They are used as follows :

M : It has been used for remembering the index of the quad to be generated just after the reduction M - > Epsilon is carried out. This is used for back patching the dangling gotos. Further details have been commented at appropriate places in .y file.

N : It has been used for generating an unconditional goto just after the reduction N - > Epsilon is carried out. This is used for control transfers. Further details have been commented at appropriate places in .y file. This has been used in augmenting statements and if else statements.

O : It has been used as an indicator of the fact that a function definition is starting. This is used to change the current symbol table pointer to point to the symbol table of the appropriate function.

Design of Attributes

Attribute	*Type*	*Terminals*	*Non terminals*
expAttr	expressionAttr(struct type)	NA	primary_expression expression expression_opt postfix_expression constant_expression statement compound_statement selection_statement iteration_statement jump_statement block_item_list block_item_list_opt block_item expression_statement unary_expression cast_expression multiplicative_express additive_expression shift_expression relational_expression equality_expression AND_expression exclusive_or_operation inclusive_or_operation logical_AND_expression logical_OR_expression

			conditional_expression assignment_expression
varAttr	varDeclaration(struct type)	NA	type_specifier declaration_specifiers direct_declarator declarator parameter_declaration init_declarator init_declarator_list pointer
initAttr	initializerAttr(union)	NA	initializer initializer_row initializer_row_list
unaryOpAttr	quadEnum	NA	unary_operator
instr	int	NA	M
NAttr	listNode(struct type)	NA	N
paramAttr	listParam(struct type)	NA	argument_expression_list
idAttr	identifierAttr(struct type)	IDENTIFIER	NA
intVal	int	INTEGER_NO	NA
doubleVal	double	FLOAT_NO	NA
charVal	char	CHARACTER	NA
strVal	char *	STRING	NA

Associated Data Structures

Data Structure	***Type and Members***	***Purpose***
typeSpecifier	Enum	Enum for data type
quadEnum	Enum	Enum for op field of quad
treeNodeType	Struct 1. middle – typeSpecifier 2. left – int * 3. right – treeNodeType*	Data type for a node of the expression tree
listNode	Struct 1. dangling_index – int 2. next – listNode*	Data type for a node of the list
symbolTableRow	class 1. name – char * 2. Type – treeNodeType* 3. initialValue – void*	Data type for one row of the

	4. size – int 5. offset – int 6. nestedTable - symbolTable*	symbol table
listParam	struct parameter – symbolTableRow * next – listParam *	Data type for list of parameters
quadArrayRow	Struct arg1 – char* arg2 – char* res – char* op – quadEnum arg1Loc – symbolTableRow* arg2Loc – symbolTableRow* arg3Loc – symbolTableRow*	Data type for one row of the quad_array
initializerAttr	Struct intData – int doubleData – double charData – char p – vector<double> *	Data type for attribute of initializer
expressionAttr	Struct loc – symbolTableRow * TL- listNode * FL- listNode * NL- listNode * type – treeNodeType* array – symbolTableRow* loc1 – symbolTableRow* val – initializerAttr isMatrix - bool	Data type for expAttr
varDeclaration	Struct type – treeNodeType* width – int var – char *	Data type for varAttr
identifierAttr	Struct loc – symbolTableRow* var – char *	Data type for idAttr
symbolTable	Class table – symbolTableRow * curr_length – int	Data type for implementing symbol table
quadArray	quad_Table – quadArrayRow*	Data type for implementing quad array

Design of Symbol Table

1. Implemented by symbolTable class whose data members have been shown in the

table above.

2. Member Methods –
    - ***symbolTableRow *lookup(char *)*** : looks up an entry in the table and returns a pointer to the row if found, null otherwise
    - ***void insert(symbolTableRow &)*** : Inserts a row in the symbolTable
    - ***symbolTableRow *gentemp(typeSpecifier)*** : generates a temporary
    - ***void update(symbolTableRow *, treeNodeType *, int)*** : updates the type and size of a predefined symbol table row entry
    - ***void update(symbolTableRow *, initializerAttr)*** : updates the initial value of a predefined symbol table row entry
    - ***void print*** : prints the table by printing each row

Design of Quad Array

1. Implemented by quadArray class whose data members have been shown in the table above.
2. Member Methods -
    - ***void emit(quadArrayRow &)*** : spits a three Address Code
    - ***void print()*** : prints the quad Array
    - ***void fillDanglingGoto(int,int)*** : fills the dangling gotos at indices specified by first parameter with the data specified by second parameter.

Global Functions

1. ***symbolTable *createSymbolTable() :*** creates a symbol table
2. ***void printTree(treeNodeType *) :*** prints the expression tree
3. ***void printInitialValue(void *,treeNodeType *) :*** prints the initial value
4. ***treeNodeType *newNode(typeSpecifier ,int) :*** cretates a new node of the tree
5. ***treeNodeType *mergeNode(treeNodeType *, treeNodeType *) :*** merges two trees

6. ***listNode *makelist(int) :*** makes a list
7. ***listNode *merge(listNode *, listNode *) :*** merges two lists
8. ***void backpatch(listNode *, int) :*** backpatches list with data specified in second parameter

9. ***int typecheck(treeNodeType *,treeNodeType *) :*** Checks whether two trees are of same type
10. ***void conv2Bool(expressionAttr *) :*** converts an expression to boolean type
11. ***int compute_width(treeNodeType *) :*** computes the width of a type
12. ***listParam *makeParamList(symbolTableRow *) :*** creates a list of parameters
13. ***listParam *mergeParamList(listParam *, listParam *) :*** merges two lists of parameters

Limitations

1. Every function before being defined needs to be declared.
2. Need for an explicit return statement at the end of every function
3. Polymorphism is not taken care of
4. Code generates a few redundant gotos while backpatching
5. Double to char conversion is not supported
6. Only 2D matrices are allowed
7. Though matrices have been implemented as an array of Matrix type, the semantic rules will not generate the desired codes for arrays hence though arrays will be parsed, opcodes will likely not be generated.
8. Quad for initialization is not generated(that is it is directly given in the initial value, no quad is generated for it)
9. Initial Value of declarations involving a variable on the right side has been kept 0
10. Matrix initialization is only with double values, integer values may give error, hence use double values while initialization of matrices
11. Direct pointer assignments are not allowed, hence pointer assignments are allowed only during declaration phase