Compilers exam 2017

Attempt - No guarantee this is correct

plz correct me if I’m wrong

# Question 1

## i.

See Notes for translations: <http://dajuice.netsoc.ie/Assets/TranslationGrammars.pdf#page=6>

{Labelp} generates a symbol table entry for the Label, p points to the symbol table entry.

{Jumpp} generates a jump to label pointed at by p

{Jumpfp,q} generates a jump to label p if condition q is false

newL creates a new symbol table entry for the label

If the {Label} action symbol is encountered 1st the symbol table entry will save the current pc for the label, then when {Jumpx} is reached it checks if the symbol table entry for label and finds the address that it need to jump to.

If the {Jumpx} instruction is encountered 1st a new symbol table entry will be created and the value of the label is set to the current pc+1 (to remove opcode), the Jump is set to jump to 0/Null - then when the {Label} instruction is reached and the table entry for label is already initialised and not 0 that means the instruction at that PC position needs to be fixed (e.g. the current pc pointing to label needs to be inserted). If multiple {Jumpx} instructions precede the label (e.g in a switch) then each subsequent jump finds the previous jump address in the symTab entry. It sets the jump to that address, and sets the symTab to it’s own address. The label instruction will then fix all instructions until it reaches on that is 0/Null.

## 

## ii.

For each new Type the symbol table should store:

* Name of the the type (unique in types)
* Type = new Data Type
* Size = Size required for data type on the stack (sum of sizes of it’s variables) does not need to allocated for the data type blueprint but for later instances
* Number of data type variables/objects
* Lexical level
* List of variables/objects in data type (e.g pointer to them/the first one if Data type acts as scope)

For each variable:

* Name of the variable (unique in its scope)
* Kind = Variable
* Type = Type of variable (e.g pointer to new defined DT)
* Sort = {Scalar | Array}
* Size = Size required on stack
* If Array: Number of Dimensions and List of the Bounds (e.g Dims = 2 Bounds = [Lower 1, Upper 1, Lower 2 Upper 2]) used to index the array & calculate its size
* Lexical level
* Pointer to next variable in the current scope
* Locals (if its a custom data type with its own objects)
* Address on the stack
* ID Link
* Dec Link

(Don’t think this is required in the exam it’s just more visual)

What this Symbol table would roughly look like:

NOTE: missing ID link and DEC link (see notes linked at end)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Kind | type | sort | Size | Dims | Bound | level | next | Locals | Adr |
| CourseDate | DT | - | - | 2 | 2 | - | 1 |  | -> CD |  |
| code | Var | INT | scalar | 1 | - | - | 2 | mark | - |  |
| mark | Var | INT | scalar | 1 | - | - | 2 | none |  |  |
| StudentInfo | DT | - | - |  | 3 | - | 1 |  | -> SI |  |
| name | Var | String | vector | ?? | - | - | 2 | num |  |  |
| number | Var | INT | scalar | 1 | - | - | 2 | course |  |  |
| course | Var | Pointer to CD | vector | 8xCD | 1 | [1,8] | 2 | none | - |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Students | Var | Pointer to SI | vetor | 32xSI | 1 | [1,32] | 1 |  |  |  |

Where

Name = name of the variable/data type

Kind = Data type/Variable/Scope/process

Sort = for variables -> scalar/vector

Size = size of obj on stack

Dims = Data type-> number of vars in DT ; Array -> number of dimensions

Bounds = Array -> [lower bound , upper bound, ...] for all dims

Level = lexical/scope level for access

Next = pointer to next obj

Locals = pointer to local objs of a DT

Adr = obj address on stack

L-attributed translation grammar:

<Declarations> <Types><Vars>p{Allocate}q; q p

<Types> TYPE <TypeDecl>

<TypeDecl>Idenpp = RECORD <VarDecl>q END; {NewType}r,s<MoreTypes> rp sq

<MoreTypes> <TypeDecl>

<MoreTypes>

<IdentList>p Identq <MoreIdents>r {Link IDs}s,t

sr (p,t)q

<MoreIdents>p , Identq <MoreIdents>r {Link IDs}s,t

<MoreIdents>p

<Vars>p VAR <VarDecl>q pq

<VarDecl>p <IdentList>q : <Type>r,s {Set Type}t,u <MoreVars>v {LinkDec}w,x

(p,t,v) q w u sr

<MoreVars>p ; <VarDecl>q  pq

<MoreVars>p

<Type>p,q <TypeSingle>r qr

<Type>p,q <TypeArray>r,s qs rp

<TypeSingle>p STRING pSTRING

<TypeSingle>p INTEGER pINTEGER

<TypeArray>p,q ARRAY [Numberr .. Numbers ] {Set Array}t,u,v OF <TypeSingle>w,x

t r us (v,w)p qx

<TypeSingle>p Identq {Retrieve}r,s

ps rq

Where for <IdentList>p <MoreIdents>p <Vars>p  <VarDecl>p <MoreVars>p <TypeSingle>p p is synthesized, <Type>p,q <TypeArray>p,q p is inherited and q is synthesised

All action symbols attributes are inherited.

Where {Allocate}p allocates space on stack for list of vars pointed at by p

Where {NewType}p,q creates a new symTab Datatype record with name of p, and

Saves list pointed at by q under locals in symTab entry of p, rest of fields e.g. size etc

Are set accordingly

where {Link IDs}p,q links p to front of q

where {Set Type}p,q sets the type field to p of symTab pointed by q (and linked IDs)

Where {LinkDec}p,q links p in front of q

Where {Set Array}p,q,r sets the sort field of the linked list items pointed at by r and set

the dimensions and bounds according to p and q

//Note: this version only works for 1 Dimension

where {Retrieve}p,q finds the symTab with name of p and assigns a pointer to it to q

Lecture Notes on some this: [dajuice.netsoc.ie/Assets/TranslationGrammars.pdf](http://dajuice.netsoc.ie/Assets/TranslationGrammars.pdf#page=7)