Παπασακελλαρίου Βασίλειος 1762
Παπασακελλαρίου Ιωάννης-Αριστείδης 2520

Ενδιάμεσος κώδικας

Η τετράδες του ενδιάμεσου κώδικα αποθηκεύονται σε ενα λεξικό με όνομα fullCode. Έχουμε επιπλέον δύο δομές λεξικού τις quad kai nextquad οι οποίες δεν κάνουν τίποτα άλλο απτο να υποδεικνύουν την τρέχουσα τετράδα που πρόκειται να αναπαραχθεί και την επόμενη για ευκολία. Οι κλασικές βοηθητικές συναρτήσεις(nextQuad, genQuad, backpatch) περιέχουν ένα επιπλέον όρισμα το οποίο δηλώνει το όνομα μιας συνάρτησης η διαδικασίας. Κάθε φορά που δημιουργείται μια τετράδα με την genQuad περνάμε επιπλέον σαν όρισμα το όνομα της τρέχουσας συνάρτησης που ανήκει αυτή η τετράδα. Αυτό μας διευκολύνει στο να δούμε στο τέλος τι τετράδες έχουν δημιουργηθεί σε κάθε μια συνάρτηση!. Επίσης εκτός από τις βοηθητικές συναρτήσεις έχουμε δημιουργήσει και επιπλέον συναρτήσεις οι οποίες βοήθουν στην παραγωγή του ενδιάμεσου κώδικα:

Η **incrementQuad(name)** απλά αλλάζει τις δομές λεξικού quad , nextQuad. Βάζει στην quad και την τρέχουσα συνάρτηση τον αριθμό της τετράδας που πρόκειται να παραχθεί , αυξάνει τον αριθμό της επόμενης τετράδας που θα δημιουργηθεί και επιστρέφει τον αριθμό της τρέχουσας τετράδας.

Η **printFullCode()** εκτυπώνει τις τετράδες που έχουν δημιουργηθεί σε κάθε μια συνάρτηση.

Η **printQuads()** εκτυπώνει όλες τις τετράδες που έχουν δημιουργηθεί σε σειρά. Για να γίνει αυτό χρησιμοποιούμε την bubblesort για να ταξινομήσουμε τις τετράδες που έχουν ήδη αποθηκευτεί στην fullCode.

Η **createNewCode(name)** καλείται κάθε φορά που συναντάμε όνομα συνάρτησης η διαδικασίας έτσι ώστε να δημιουργηθεί ένα καινούργιο κλειδί στην fullCode το οπόιο θα περιέχει μια λίστα με όλες τις τετράδες που πρόκειται να δημιουργηθούν σε αυτή τη συνάρτηση με το όνομα name.Επιπλέον αρχικοποιεί και τις quad , nextQuad με τις κατάλληλες ετικέτες τετράδων. Στην nextQuad[name] βάζουμε τον επόμενο αριθμό της τελευταίας τετράδας που έχει παραχθεί στο πρόγραμμα.

Η **changeQuads(quads)** χρησιμοποίται με σκοπό να αλλάξει στις τετράδες που έχουν παραχθεί και περιέχουν το “_” να περιέχουν το κενό. Είναι απλά μια βοηθητική συνάρτηση για τις τετράδες που θα εκτυπωθούν στο αρχείο .c

Η **writeInFile(filename)** παίρνοντας σαν όρισμα το τρέχων αρχείο στο οποίο γίνεται parse δημιουργεί δύο καινούρια αρχεία με καταλήξεις .int, .c, στα οποία θα γράψουμε τις τετράδες που έχουν παραχθεί.

Σημασιολογική ανάλυση

Έχουμε δημιουργήσει μια βοηθητική συνάρτηση την **checkReturns** η οποία βγάζει διαγνωστικά μηνύματα σχετικά με το αν κάθε συνάρτηση η διαδικασία στο πρόγραμμα έχει τουλάχιστον ένα return η αν δεν πρέπει να έχει return αντίστοιχα.

Πίνακας Συμβόλων

Ο πίνακας συμβόλων έχει υλοποιθεί με μια λίστα (scopeStack) η οποία περιέχει ένα λεξικό για κάθε καινούργιο scope. Το κάθε scope περιέχει αρχικά το όνομα του (scopeName), το όνομα του γονέα (parentName), τον τυπο του (FUNCTION or PROCEDURE) και το τύπο της τιμής επιστροφής (returnType). Επίσης έχουμε και μια λίστα (stackOffset) στην οποία κάνουμε append το αρχικό offset για κάθε καινούριο scope που δημιουργούμε.

Για τον πίνακα συμβόλων έχουμε δημιουργήσει τις εξής συναρτήσεις:

searchin(identifier)
ο σκοπός της είναι να επιστρέψει τον identifier απο τον πίνακα συμβόλων.
searchinScopeStack(identifier, position)
χρησιμοποιόντας αναδρομή ψάχνει τον identifier στο τρέχων scope.Αν δεν υπάρχει ψάχνει στον γονέα και συνεχίζει με τον ίδιο τρόπο. Όταν τον βρεί τον επιστρέφει.

GetCurrentScope()
Επιστρέφει το όνομα του τρέχοντος scope.

addScope(scopeName, functionType, nestingLevel)
δημιουργεί ένα καινούργιο scope το οποίο θα περιέχει όπως αναφέραμε το όνομα του το όνομα του (scopeName), το όνομα του γονέα (parentName), τον τυπο του (FUNCTION or PROCEDURE) και το τύπο της τιμής επιστροφής (returnType) και το βάζουμε στη στοίβα που κρατάει τα scopes (scopeStack). Επίσης βάζουμε στην stackOffset το αρχικό offset για το καινούριο scope.

addEntity(identifier, identifierType)
παίρνει το τρέχον scope , κάνει pop το τρέχων offset του scope και δημιουργεί μια καινούργια εγγραφή(κλειδί) με τιμή ένα λεξικό, αν δεν υπάρχει ήδη. Ανάλογα με το identifierType ενεργεί κατάλληλα. Αν είναι FUNCTION ή PROCEDURE αυτο σημαίνει ότι βρέθηκε μια καινούργια συνάρτηση οπότε το λεξικό της εγγραφής θα γεμίσει με τον τύπο της(type), την λίστα παραμέτρων (listOfArguments) το οποίο θα περιέχει μια λιίστα με τις παραμέτρους της συνάρτησης, την αρχική ετικέτα τετράδας της συνάρτησης (startQuad) και το frameLength. Αν δεν είναι FUNCTION ή PROCEDURE τότε αυτό σημαίνει ότι βρέθηκε απλά μια καινούργια μεταβλητή, οπότε θα καταγραφεί το offset της και ο τύπος της. Τέλος αυξάνουμε το offset + 4.

addEntityAttr(identifier, key, value)
ψάχνει τον identifier και αλλάζει σε value την τιμή του key

getEntityAttr(identifier, key)
ψάχνει τον identifier και επιστρέφει την τιμή του κλειδιού του key

addEntityAttrToCurrentScope(identifier, key, value)
στο τρέχον scope δημιουργεί στην τιμη(λεξικό) του identifier μια καινούργια εγγραφή key με τιμή value. Αφορά τα ορίσματα της συνάρτησης (και συγκεκριμένα για το όρισμα με όνομα identifier)που έχει δημιουργηθεί και το key θα έχει τιμή (“par”)
και το value θα είναι ή “in” ή “inout” ανάλογα με το αν η παράμετρος είναι με τιμή η αναφορά.

addEntityArgToPreviusScope(identifier, key, value)
Κάθε φορά που συναντάμε μια καινούργια συνάρτηση η διαδικασία δημιουργούμε και ένα καινούργιο scope το οποίο το κάνουμε append στην καθολική μεταβλητή που κρατάει τα scopes scopeStack. Άρα το τρέχων scope είναι για την καινούργια συναρτήση που συναντήσαμε στο οποίο θα εισάγουμε τα κατάλληλα entities για κάθε μεταβλητή που διαθέτει η συνάρτηση ή παράμετρο.Αν η καινούργια συνάρτηση λοιπόν είναι εμφωλιασμένη στην προηγούμενη

συνάρτηση(previus scope), και έχει κάποια ορίσματα , αυτά θα πρέπει να τα καταγράψουμε στη προηγούμενη συνάρτηση σαν “listOfArguments” της τρέχοντος συνάρτησης.Άρα ο σκοπός αυτής της βοηθητικής μεθόδου είναι να καταγράφει τα ορίσματα μια καινούργιας συνάρτησης στην προηγούμενη συνάρτηση στην οποία είναι φωλιασμένη.

getEntityAttrFromCurrentScope(identifier, key)
Ο σκοπός αυτής της συνάρτησης είναι να μας επιστρέψει το γνώρισμα είται μιας μεταβλητής(πχ την τιμή του offset, αν είναι περασμένη με αναφορά ή τιμή αν είναι όρισμα συνάρτησης(parMode) ή τον τύπο της(type)) είται μιας συνάρτησης(πχ framelength, listOfArguments, startQuad, type).

identifierExists(identifier, nestingLevel)
Ελέγχει αν η μεταβλητή ή η συνάρτηση identifier υπάρχει δηλωμένη στο βάθος φωλίασματος nestingLevel.
Ο σκοπός της είναι να μας βοηθήσει να μην αφήσουμε να δηλώσουμε ίδιο όνομα μεταβλητής ή συνάρτησης στο ίδιο βάθος φωλίασματος.

identifierExistsInCurrentScope(identifier)
Ελέγχει αν ο identifier υπάρχει ήδη στο τρέχων scope.

DeleteCurrentScope()
διαγράφει το τρέχων scope και το τοποθετεί στην σφαιρική μεταβλητή stackHistory η οποία περιέχει ολα τα scope που έχουν δημιουργηθεί και ουσιαστικά είναι ο τελικός πίνακας συμβόλων που θα εκτυπωθεί στο τέλος.

PrintSymbolTable()
εκτυπώνει το τελικό πίνακα συμβόλων μετά την διαγραφή όλων των scope και την τοποθέτησή τους στη σφαιρική μεταβλητή stackHistory. Για το σκοπό αυτό έχουμε χρησιμοποιήσει τη βιβλιοθήκη pprint.

Τελικός κώδικας

Για την παραγωγή του τελικού κώδικα έχουν προστεθεί οι συναρτήσεις gnlvcode, loadvr kai storevr. Επίσης έχουμε δημουργήσει κάποιες βοηθητικές συναρτήσεις οι οποίες χρειάστηκαν στην παραγωγή αυτών των συναρτήσεων.

getScope(identifier)
Κάνει αναζήτηση στην στοίβα (scopeStack) που περιέχει τα scopes και επιστρέφει το scope(dict) στο οποίο βρίσκεται η μεταβλητή identifier.

searchInStack(identifier, position)
Χρησιμοποιεί αναδρομή στο scopeStack για να βρεί το scope στο οποίο περιέχεται η μεταβλητή identifier.

getScopeAttribute(scope, attribute)
Από ένα συγκεκριμένο scope επιστρέφει την τιμή του attribute(“nestingLevel, startQuad etc...)