Modalità d'esame e progetti per la parte di Linguaggi. L'esame di Linguaggi e Compilatori consiste nello svolgimento di un progetto di Linguaggi, l'esame di Compilatori (da definire a cura del Docente titolare) ed un orale individuale. Il voto finale è la media (1/3 + 1/3 + 1/3) delle tre prove arrotondata per eccesso. La prova orale è cumulativa dell'intero corso.

Per la parte di Linguaggi, l'esame consiste nella realizzazione di uno dei progetti sotto proposti. I gruppi (massimo 3 studenti) dovranno sviluppare il progetto scelto, producendo un elaborato scritto (possibilmente in inglese) che riporti le specifiche e la implementazione del progetto scelto, comprensivo di una fase di testing del progetto su alcuni esempi che si ritengono significativi. I gruppi sono liberi di interpretare quanto non specificato nella consegna del progetto secondo il loro gusto e le loro scelte progettuali. La valutazione del progetto riguarderà: grado di originalità (anche rispetto ad altri progetti), qualità delle scelte progettuali, livello di dettaglio dell'elaborato scritto, e significatività degli esempi proposti per verificare la corretta implementazione del progetto. Agli studenti, in fase di esame orale, sarà richiesto di eseguire le specifiche sugli esempi proposti. I progetti possono essere consegnati in una qualsiasi nelle date di appello definite a calendario, per la parte di Linguaggi a iniziare dal primo appello di Febbraio 2016. Gli orali saranno svolti nelle date di appello definite a calendario. La validità dei progetti di Linguaggi svolti è di 1 anno accademico. Eventuali appelli straordinari possono essere richiesti ed attivati solo su richiesta di almeno 3 gruppi. Contattare il Tutor (Emanuele Falzone) per la composizione dei gruppi. Una volta costituito il gruppo gli studenti dovranno scegliere uno dei 5 progetti sotto indicati e comunicarlo al Tutor per email. Solo i gruppi che hanno comunicato la loro costituzione ed hanno optato per uno dei progetti entro il 20 Gennaio 2016, possono consegnarlo nel primo appello di Febbraio 2016.

Progetto in OCAML A.A. 2016-2017 Il progetto da sviluppare è finalizzato a estendere l'interprete OCAML (denotazionale o operazionale o iterativo a vostra scelta) del linguaggio didattico imperativo e funzionale comprensivo di blocchi, procedure e funzioni (senza quindi la parte a oggetti). Esso consta di una di queste due parti (ovvero per fare il progetto basta aver svolto o 1 o 2 o 1+2). Ovviamente lo svolgimento di entrambe le parti correttamente comporta un possibile voto più alto e la possibilità di accreditare uno stage.

1. Estendere il linguaggio didattico visto a lezione con il tipo stringa (di caratteri) con espressioni su stringhe quali: lunghezza len(s), concatenazione @, sotto-stringa substr(s,i,j) che restituisce la sotto-stringa di s composta dai caratteri dall'i-esino al j-esimo se i<=y altrimenti la stringa vuota, e costante stringa 's' dove s e' una sequenza di caratteri alfanumerici. A vostra scelta di introdurre ulteriori estensioni/operazioni prendendo liberamente spunto da operazioni note in linguaggi di scripting (es PHP o Javascript). Estendere quindi i comandi del linguaggio con il comando reflect(s) che prende in input una stringa s e chiama lo stesso interprete sulla stringa vista come sequenza di comandi.
2. Sviluppate una analisi di information-flow (in gergo: Taint Analysis) come tipaggio dinamico del linguaggio didattico come esteso al punto 1 sopra. Si devono determinare i flussi tainted (untrusted) durante la computazione di un programma P. Immaginate di partizionare la memoria e l'ambiente in input (di avvio del programma P) tra dati trusted (untainted) e untrusted (tainted).

Durante il calcolo la computazione di una operazione (espressione, comando o dichiarazione) che compone dati tainted con dati untainted da come risultato un dato untainted se il risultato è indipendente dall'input tainted. Viceversa se l'output della operazione dipende da un dato tainted il risultato è tainted. Nel caso di reflect(s), se la stringa s risulta tainted bloccare la computazione in quanto si ha un potenziale code injection attack. L'analisi deve fornire una memoria ed un ambiente in output con valori etichettati tainted o untainted e deve controllare dinamicamente la esecuzione in modo da evitare code injection.