**Poznámky k prezentaci:**

• Github

* Verzování(git)
* Vzdálený repozitář(Github.com)
* Rozdělená práce
* Sledování postupu

• Messenger

• Časté osobní schůzky

* **Lex. analyzátor**
  + na základě konečného automatu
  + Scanner dále komunikuje se syntaktickou analýzou prostřednictvím pomocné struktury *Token*, která představuje předávaný token, tato struktura obsahuje veškeré informace o konkrétním tokenu – jeho hodnotu, typ, alokovaný prostor, atd... Scanner postupně načítá jednotlivé znaky ze vstupu (*stdin*) dokud neprojde do koncového stavu, v tomto případě posílá hotový token, v opačném případě dochází k chybě s kódem 1.
  + hlavní problém: indent, dedent
    - Pro zpracování *INDENT* a *DEDENT* (odsazení) byla vytvořena pomocná struktura *iStack* (indent stack), do kterého se při odsazování ukládá aktuální úroveň odsazení a počet mezer v odsazení, zároveň byla použita statická proměnná, která slouží k poznačení si, zda je aktuálně zpracovávaný token prvním na řádku, či nikoliv, tyto informace jsou následně využity pro odsazování a vracení se z odsazení.
  + Pro ESCAPE sekvence je použita statická proměnná, díky který lze zkontrolovat, zda byl předchozí znak „\“ či nikoliv.
* **Syntaktická analýza**
  + pro většinu programů (bez výrazů) syntaktická analýza shoda dolů
    - vytvořili jsme si LL(1) gramatiku
    - neterminál = jedna funkce programu, potom na základě tokenů volíme, které pravidlo LL gramatiky zvolit – neočekávaný token = syntaktická chyba
  + Syntaktická analýza je založená na LL gramatice, která je simulována pomocí rekurzivního sestupu
  + při tokenu ID – není snadné rozhodnout, zda se jedná o výraz, přiřazení výrazu nebo přiřazení funkce
    - řešení: přednačtení tokenu
  + pro každý příkaz voláme příslušné funkce generátoru kódu
  + Zpracování výrazů:
    - syntaktická analýza zdola nahoru
    - precendenční tabulka + funkce, která se opakovaně volá, dokud se výraz nezredukoval na výsledek
    - implementace zásobníku pro ukládání operátorů a operandů
    - při každé redukci syntaktické + sémantické kontroly podvýrazů – ok = volání funkcí na generování kódu
    - jinak chyba
* **Sémantická analýza**
  + Informace o proměnných a jejich typech jsou ukládány do tabulky symbolů. Stejně tak informace o funkcích, zda byly definovány, počet parametrů atp. Tyto informace jsou následně využívány ke kontrolám definic a redefinic.
  + při kontrolu definicí funkcí kontrolujeme, zda volaná funkce je již funkce definována dříve, navíc kontrolujeme i funkce, na kterých je tato závislá
  + pro globální proměnné ve funkci máme poznamenáno, zda byly využity, pokud se následně definuje proměnná stejného jména program skončí se sémantickou chybou
* **Generování kódu**
  + jednosměrně vázaný seznam – ukládání všech instrukcí před jejich výpisem
  + možnost vkládání i před existující položky v seznamu – využití při tvorbě definic funkcí před hlavním tělem programu, při vytváření těla cyklu while či tvorbě konstrukce if-else
  + pro ostatní moduly překladače – pomocné funkce pro generování konstrukcí
  + hlavní výzva: typové kontroly – pro vyřešení této výzvy jsme použili funkce na generování if-else, přičemž bylo potřeba ošetřit všechny možné kombinace operandů pro instrukce a v případě špatné instrukce ukončit interpret s chybovým návratovým kódem
  + generování vestavěných funkcí, které se mohli následně využít v dalším kódu