Compiler Design

Today's Class Topics

-Run time Environment
-Activation Record
-PYQ Discussion

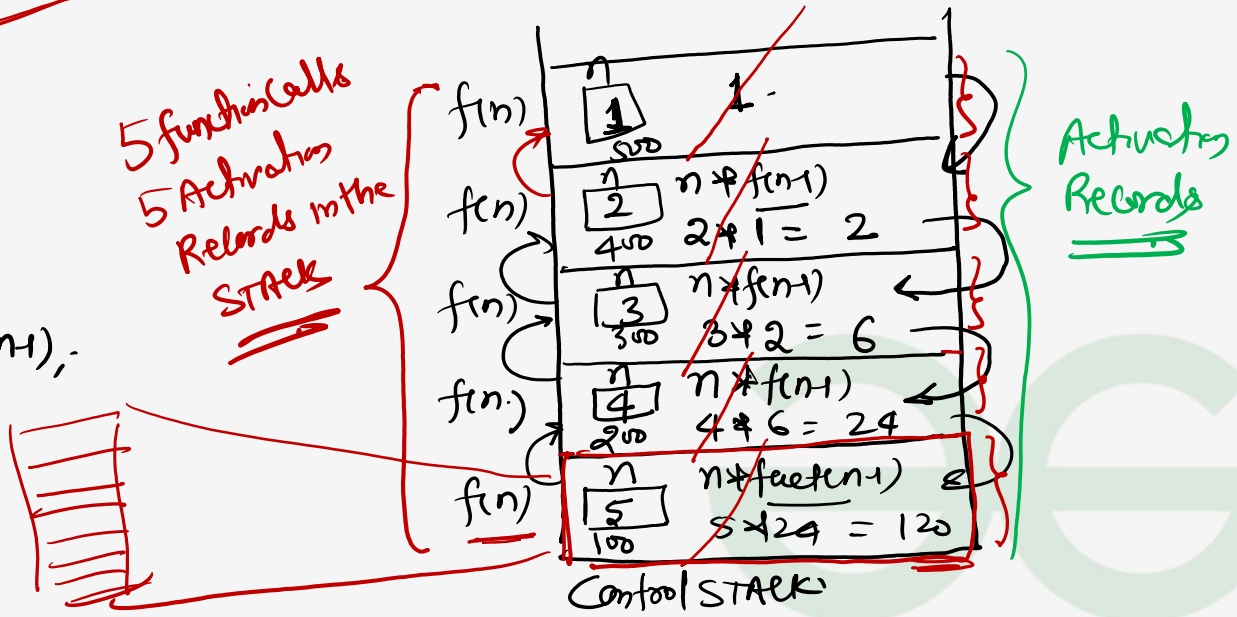
Compiler Design

Run time Environment :- SL → IR → m/c code → Runtime Environment

What ever a necessary support required by O.S to run a Program is called Runtime environment.

Ex:

```
fact(int n)
{ if (n ≤ 1)
    return 1;
  else
    return n * fact(n-1);
}
```



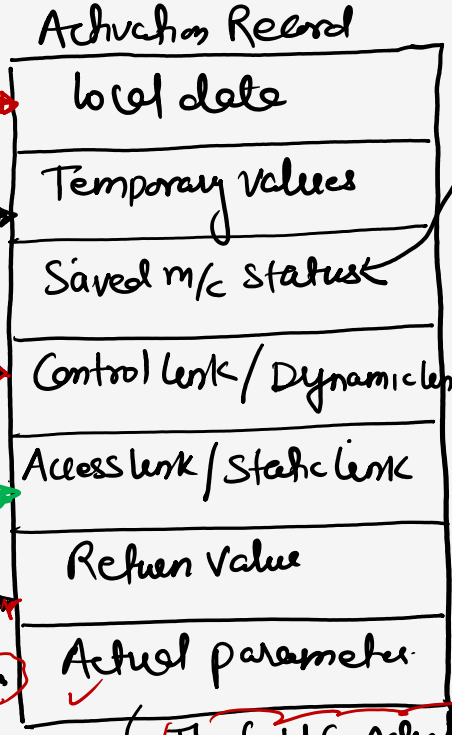
Compiler Design

Activation Records:-

Activation records Contain the necessary information to run the procedure, it has 7 fields

Compiler Design

Activation Records: has 7 - fields



Holds the data that is local to the execution of the function

Stores the values that arises in the execution of an expression

Store the Address of Activation record of the caller function

Store the information of data which is outside the local scope (i.e. data of calling function)

Used by the called procedure, to return a value to calling function

Stores m/c status such as register, program Counter etc. before the procedure is called Holds the information about status of machine just before the function call

Ex:

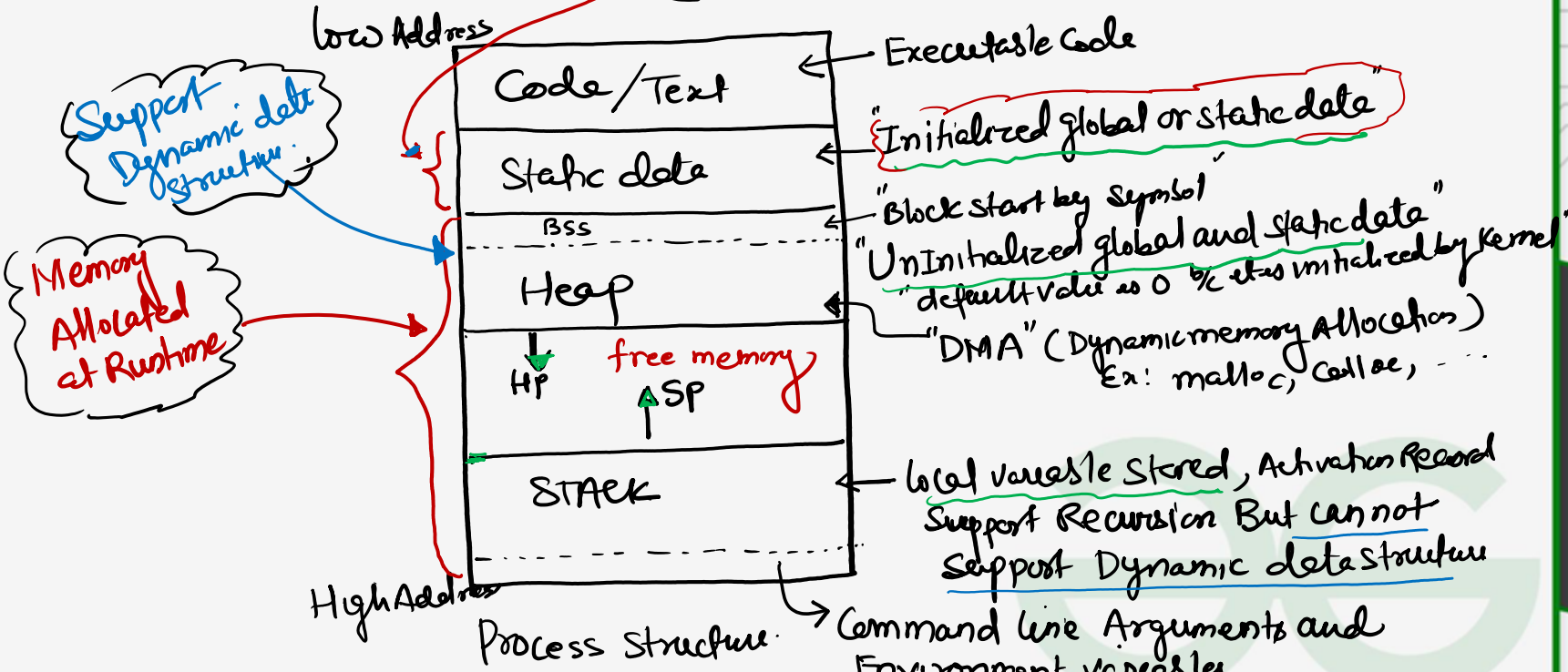
```
A()  /* Caller or Calling function */
{ int a, b;
  =
  B(); /* Called function */
  =
}
B()
{ int c;
  c = a + b
  =
}
```

The field for Actual parameters is used by the Calling Procedure to supply parameters to the called procedure

Compiler Design

Runtime Environment of C-program :

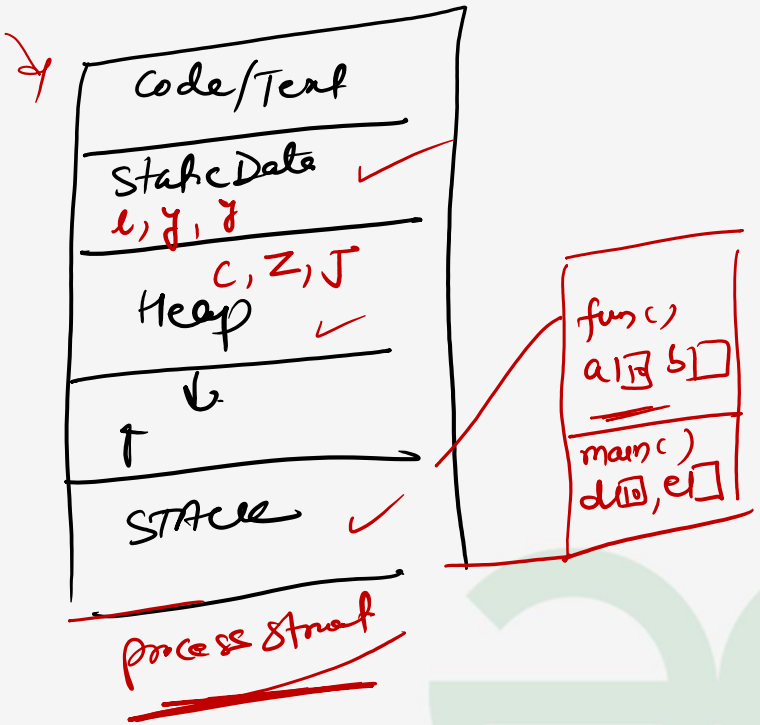
memory allocation at Compile time it is fixed, it can not support Dynamic data structure



Compiler Design

Ex-

```
int i=10, J;
fun (int x)
{ static int y=10, Z;
  int a=10, b;
}
main()
{ static int f=20, C;
  int d=10, e;
  fun()
```



Compiler Design

Static Allocation:

1) Memory Allocation is done at Compile time
2) Binding do not change at runtime
3) Recursion is not Supported.
4) Size of data objects must be Known at Compile time
5) Dynamic data structure not Supported

Heap Allocation:

1) Allocation and deallocation will be done at any time based on user requirement
2) Recursion Supported
3) Dynamic data structure supported

Stack Allocation:

1) Recursion Supported
2) local variable belongs to new Activation Record.
3) Dynamic data structure not supported
4) local variables can not be retained once Activation end.

Compiler Design

59 ⇒ ⇒

15 — 2011 — 14

Important Topics:

1) Phases of Compiler
2) Count No. of Tokens
3) Errors
4) Parsers
   - First & Follow, Left Recursion, Left factoring
   - LL(1) Parser
   - SLR(1) & LR(0) Parser
   - CLR(1) Parser
   - LALR Parser
   - S/R, R/R Conflict
5) SDT
6) SSA
7) DAG
8) Control flow Graph (CFG) & Liveness Analysis

Compiler Design

Compiler Design

Compiler Design

Compiler Design

Q-1: The number of tokens in the following C statement is
printf("i = %d, &i = %x", i, &i);
(A) 3
(B) 26
(C) 10
(D) 21

Q-2: In a compiler, keywords of a language are recognized during
(A) parsing of the program
(B) the code generation
(C) the lexical analysis of the program
(D) dataflow analysis

Compiler Design

Q-3: Which one of the following statements is FALSE?
(A) Context-free grammar can be used to specify both lexical and syntax rules.
(B) Type checking is done before parsing.
(C) High-level language programs can be translated to different Intermediate Representations.
(D) Arguments to a function can be passed using the program stack

Q-4: Incremental-Compiler is a compiler
(A) which is written in a language that is different from the source language
(B) compiles the whole source code to generate object code afresh
(C) compiles only those portion of source code that have been modified.
(D) that runs on one machine but produces object code for another machine

Compiler Design

Q-5: A lexical analyzer uses the following patterns to recognize three tokens T_1, T_2, and T_3 over the alphabet {a,b,c}.

T_1: a?(b|c)*a
T_2: b?(a|c)*b
T_3: c?(b|a)*c

Note that 'x?' means 0 or 1 occurrence of the symbol x. Note also that the analyzer outputs the token that matches the longest possible prefix.

If the string *bbaacabc* is processes by the analyzer, which one of the following is the sequence of tokens it outputs?

(A) $T_1T_2T_3$
(B) $T_1T_1T_3$
(C) $T_2T_1T_3$
(D) T_3T_3

Compiler Design

Q-6: Consider the following two statements:
P: Every regular grammar is LL(1)
Q: Every regular set has a LR(1) grammar
Which of the following is TRUE?
(A) Both P and Q are true
(B) P is true and Q is false
(C) P is false and Q is true
(D) Both P and Q are false

Q-7: A bottom-up parser generates:
(A) Left-most derivation in reverse
(B) Right-most derivation in reverse
(C) Left-most derivation
(D) Right-most derivation

Compiler Design

Q-8: Consider the following two sets of LR(1) items of an LR(1) grammar.

X ->c.X, c/d
X -> .cX, c/d
X -> .d, c/d

X ->c.X, $
X -> .cX, $
X -> .d, $

Which of the following statements related to merging of the two sets in the corresponding LALR parser is/are FALSE?

1. Cannot be merged since look aheads are different.
2. Can be merged but will result in S-R conflict.
3. Can be merged but will result in R-R conflict.
4. Cannot be merged since goto on c will lead to two different sets.

(A) 1 only
(B) 2 only
(C) 1 and 4 only
(D) 1, 2, 3, and 4

Compiler Design

Q-9: The grammar $S \rightarrow aSa \mid bS \mid c$ is

(A) LL(1) but not LR(1)

(B) LR(1)but not LR(1)

(C) Both LL(1)and LR(1)

(D) Neither LL(1)nor LR(1)

Q-10: Match all items in Group 1 with correct options from those given in Group 2.

Group 1	Group 2
P. Regular expression	1. Syntax analysis
Q. Pushdown automata	2. Code generation
R. Dataflow analysis	3. Lexical analysis
S. Register allocation	4. Code optimization

(A) P-4. Q-1, R-2, S-3

(B) P-3, Q-1, R-4, S-2

(C) P-3, Q-4, R-1, S-2

(D) P-2, Q-1, R-4, S-3

Compiler Design

Q-11: An LALR(1) parser for a grammar G can have shift-reduce (S-R) conflicts if and only if

(A) the SLR(1) parser for G has S-R conflicts

(B) the LR(1) parser for G has S-R conflicts

(C) the LR(0) parser for G has S-R conflicts

(D) the LALR(1) parser for G has reduce-reduce conflicts

Q-12: Which one of the following is a top-down parser?

(A) Recursive descent parser.

(B) Operator precedence parser.

(C) An LR(k) parser.

(D) An LALR(k) parser

Compiler Design

Q-12: Consider the grammar with non-terminals N = {S,C,S1 },terminals T={a,b,i,t,e}, with S as the start symbol, and the following set of rules:

$S \rightarrow iCtSS_1 | a$

$S_1 \rightarrow eS | \epsilon$

$C \rightarrow b$

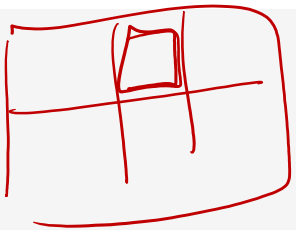
The grammar is NOT LL(1) because:

(A) it is left recursive

(B) it is right recursive

(C) it is ambiguous

(D) It is not context-free.



Compiler Design

Q-13: A canonical set of items is given below

S --> L. > R

Q --> R.

On input symbol < the set has

(A) a shift-reduce conflict and a reduce-reduce conflict.

(B) a shift-reduce conflict but not a reduce-reduce conflict.

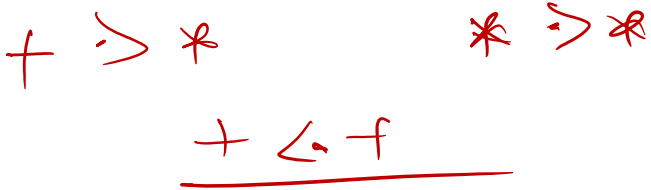
(C) a reduce-reduce conflict but not a shift-reduce conflict.

(D) neither a shift-reduce nor a reduce-reduce conflict.

Compiler Design

Q-14: Consider the grammar defined by the following production rules, with two operators * and +

S --> T * P
T --> U | T * U
P --> Q + P | Q
Q --> Id
U --> Id



Which one of the following is TRUE?

(A) + is left associative, while * is right associative
(B) + is right associative, while * is left associative
(C) Both + and * are right associative
(D) Both + and * are left associative