* Assignment - 2 *

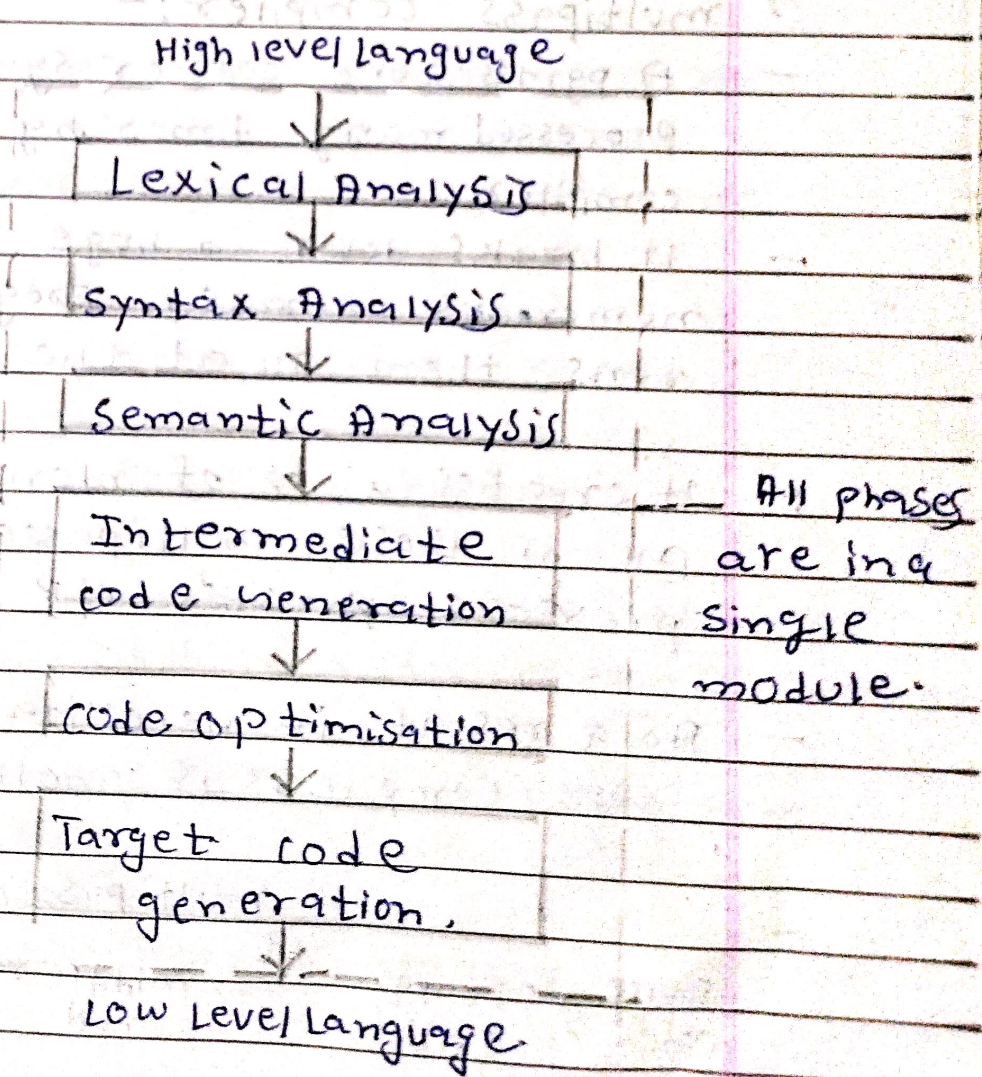
PAGE NO.
DATE : / /

(1) Write down diff. types of compiler in details and also draw a labelled diagram where need.

→ Single pass compiler :-

When we merge all the phases of compiler design in a single module, then it is called a single pass compiler. In the case of a single pass compiler, the source code converts into machine code.

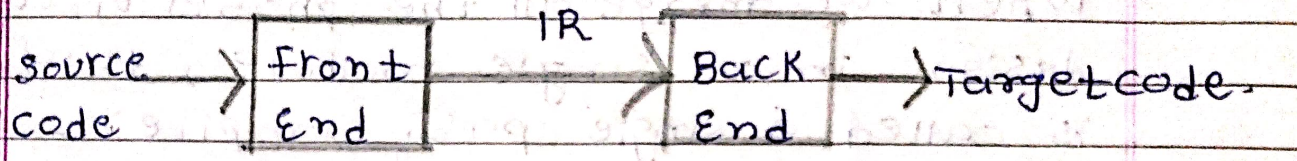
→



→ two pass compiler:-

– A processor that runs through the program to be translated twice is considered a two-pass compiler.

Two pass compiler:



→ multipass compiler:-

– A pgm's source code or syntax tree is processed many times by the multipass compiler.

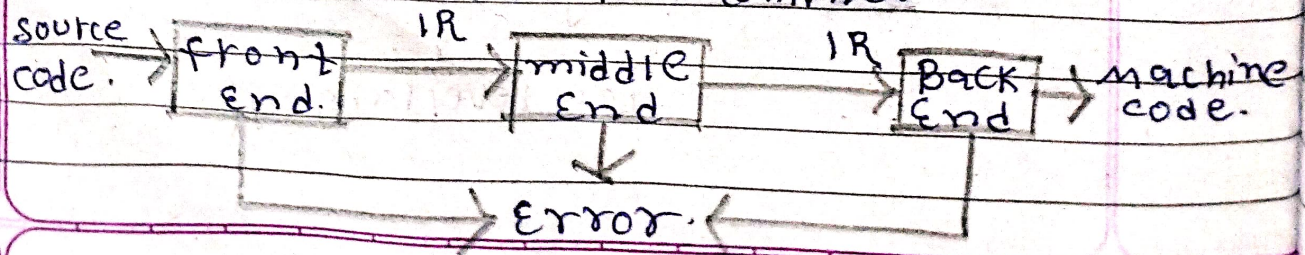
– It breaks down a huge programe into numerous smaller programmes and runs them all at the same time.

– It creates a no. of intermediate codes. All of these multipasses use the previous phase's o/p as an i/p.

– As a result, it necessitates less memory. 'wide compiler' is another name for it.

–

multipass compiler.



(2) Explain about re-allocatable machine code in difference to loader & Linker.

→ Re-allocatable machine code :-

- Re-allocatable ~~at different~~ machine code is a type of M-C that can be loaded and executed at different memory locations.

- This flexibility is useful becaus it allows the same program to run in different memory areas without modification.

- This code contains special instructions or placeholders that can be adjusted based on where in memory the code is loaded.

→ Loader :- is a part of the os that loads pgms into memory & prepares them for execution.

(1) Load pgm :- It reads the pgm's mc from the disk into the computer memory.

(2) Adjusts memory Addresses :- If the pgm is re-allocatable, the loader adjusts the memory addresses so the pgm can run correctly from the loaded location.

PAGE NO. :
DATE : / /

(3) Starts execution :- finally, the loader transfers control to the program so it can start running.

→ Linker :- A Linker is a tool that combines diff. pieces of code & data into a single executable pgm.

(1) combines obj. files :- It takes various object files & combines them into one pgm.

(2) Resolves References :- If one part of the code calls a function or uses a variable from another part, the linker makes sure these references are correctly connected.

(3) produces Executable :- The linker produces the final executable file that the loader can load & run.

PAGE NO.:
DATE: / /

(3) What are the advantages & disadvantage of compiler design Define.

→ Advantages of CD:-

(1) optimization :- compilers can make the code run faster and use less memory by optimizing it during the translation from high-level language to machine code.

(2) Error checking :- compilers catch many types of errors before the pgm runs, such as syntax error or type mismatches.

(3) Machine independence :- high-level code can be written once & compiled for different machine architectures without changing the source code.

(4) Speed of execution :- once a pgm. is compiled, it can be executed quickly by the machine.

(5) Security :- compiled code is harder to reverse-engineer than interpreted code.

→ Disadvantages of CD :-

(1) Compilation Time :- The process of converting high-level code to machine code can take a significant amount of time.

(2) Memory usage :- compilers can use a lot of memory during the compilation process.

(3) Debugging Difficulty :- debugging compiled code can be harder because the source code is translated into machine code.

(4) Lack of flexibility :- one compiled, the m-c is fixed & can't be easily changed.

(5) Complexity :- compiler design involves understanding complex algorithms and data-structures.

Define.

(4) (i) Why code optimizer is optimal in compiler design.

→ A code optimizer is a crucial part of CD bcz it makes the generated machine code more efficient.

(1) faster execution :-
optimized code runs faster because the optimizer removes unnecessary instructions and improves the overall flow of the program.

(2) Reduced memory usage :-
The optimizer can minimize the amount of memory a prg. uses by eliminating redundant data & instructions.

(3) Efficient Resource utilization :-
optimize code makes better use of the computer's resources (cpu, memory, etc.)

(4) Improved Battery life :-
for mobile devices, optimized code can lead to less cpu usage and, therefore, lower power consumption.

(5) Reduced cost :-
Efficient programs can reduce the need for expensive hardware upgrades.

PAGE NO. :
DATE : / /

(ii) (8) Explain ways of code generator in CD by any code running example.

→ A code generator in compiler design takes the IR of the source code & translates it into Machine code or assembly code that the computer can execute.

→ code ex:- Simple Arithmetic.

- consider the following simple code in a high-level language (e.g., C):

```
- int main()
  {
      int a=5;
      int b=10;
      int c=a+b;
      return c;
  }
```

→ steps of code generation

① (IR)
② Instruction selection
③ Register Allocation
④ instruction scheduling.

→ (IR) :- first, the compiler translates the h-l code into an intermediate representation (IR).

DATE: / /

Ex- IR (using a hypothetical simple language).

```
a=5
b=10
c=a+b
return c.
```

→ (Instruction selection):- The code generator translates the IR into sequence of machine instructions or assembly code.

Ex- Assembly code.

```
mov eax, 5 ; move the value 5 into register eax (a=5)
mov ebx, 10 ; move the value 10 into register ebx (b=10)
add eax, ebx ; Add the values in eax & ebx,
               result in eax (c=a+b)
mov ecx, eax ; move the result into register
               ecx (store c).
mov ecx, eax ; move the result into register
               eax (return c)
ret ; Return the value in eax.
```

→ (Register Allocation):-
The compiler must decide which variables will be kept in the CPU registers, which are faster to access than memory. The above ex already demonstrates this, as it uses registers 'eax', 'ebx', & 'ecx'.

→ (Instruction scheduling):-
This step involves arranging the instructions to avoid pipeline stalls and improve performance.

(iii) (8) What are various operation of compiler.

→ Its break down the source pgm into smaller parts.

→ It enables the creation of symbol tables & IR.

→ It helps in compilation and error detection.

→ It saves all codes & variables.

→ It analysis the full pgm. & translate it into machine code.

→ convert source pgm to obj. code and obj code into machine code.

- source code — means user readable code or other words we can say natural language code in which the pgmer gives it in to as input.

→ various operation of compiler :-

→ ① Lexical Analysis. (Scanning) :-

- The source code is read & converted in to tokens.
- Tokens are the basic building blocks like keywords, identifiers, operators, & literals.
- for code int a=5 token would be 'int', 'a', '=', '5', & ';'.

② Syntax Analysis (parsing) :-

- The seq. of tokens is analyzed against the grammatical rules of the pgm language.
- The o/p is a parse tree or syntax tree representing the syntactic structure of the source code.
- ex- int a=5; the syntax tree might show that 'a' is a var. of type 'int' and is assigned the value '5'.

DATE: / /

③ Semantic Analysis :-

- The compiler checks for semantic errors, ensuring the meaning of code is correct.
the

- It involves type checking, variable declaration checks, and scope resolution.

- ex- int a = 5; float b = a;
, the compiler verifies that assigning an 'int' to a 'float' is valid.

④ IR code generation :-

→ The compiler generates an intermediate representation (IR) of the source code.

- The IR is a low-level code that is easier to optimize & translate into machine code.

ex - 'a = b + c;, the IR might be something like 't1 = b+c'. followed by 'a = t1;

DATE : / /

(5) optimization :-

- The compiler optimizes the intermediate code to improve performance and efficiency.

- optimization can be done at various level : local (with in basic block), global (across basic blocks), & loop optimization.

ex - 'a = b * 2 + b * 2; optimization might simplify it to 'a = 2 * (b * 2);

(6) code generation :-

- additional optimization are performed on the machine code to further improve performance.

- This can include instruction scheduling, register allocation, & peephole optimization.

ex - Rearranging instruction to avoid pipeline stalls or combining multiple instructions into a single efficient instruction.

DATE : / /

(5) write a program of implementation of recursive decendent parser without backtracking.

→
```
#include <stdio.h>
#include <ctype.h>

const char *input;
int expr();

int factor(){
    int result=0;
    if (*input == '('){
        input++;
        result = expr();
        input++;
    }
    else{
        while (isdigit(*input)){
            result = result*10 + (*input++ - '0');
            input++;
        }
    }
    return result;
}
```

```
int term() {
    int result = factor();
    while (*input == '*') {
        input ++;
        result *= factor();
    }
    return result;
}

int expr() {
    int result = term();
    while (*input == '+') {
        input ++;
        result += term();
    }
}

int main() {
    input = "2+3*(4+5)";
    printf("Result: %d\n", expr());
}
```

Output :- Result: 29.

-X-X-X-X-