Ans1) (a)

A **scanner,** also known as a lexical analyzer, is the first phase of a compiler. It converts the input program into a sequence of tokens. It uses a set of rules to determine whether each character is part of a token or not. If the scanner sees a sequence of letters that matches a keyword, it will create a token for that keyword. Once the scanner has created all of the tokens, it passes them to the parser, which is the next phase of the compiler.

A **token** is a sequence of characters that has a meaning in the programming language. For example, the token "int" is a keyword that represents an integer.

A **lexeme** is a sequence of characters that is recognized by the scanner as a single token. For example, the lexeme "int" is a keyword in C programming language.

(b)

There are several benefits to using machine independent intermediate forms. These include:

- **Portability:** A compiler that uses a machine independent intermediate form can be easily ported to new target machines. This is because the intermediate form is not specific to any particular machine, so the compiler does not need to be re-written for each new target machine.
- **Efficiency:** A machine independent intermediate form can be used to perform optimizations that would not be possible if the compiler generated code directly for the target machine. This is because the intermediate form is a more abstract representation of the program, so the compiler has more freedom to perform optimizations.
- **Flexibility:** A machine independent intermediate form can be used to generate code for different target machines. This allows the compiler to be used to generate code for a variety of different platforms, without having to be re-written for each platform.

(c)

Some of the primary structure preserving transformations on basic blocks are:

- **Common subexpression elimination:** This transformation removes common subexpressions from a basic block. For example, if a basic block contains the expressions `a + b` and `c + b`, then common subexpression elimination can be used to remove the expression `b` from both expressions. This can improve the performance of the generated code.
- **Dead code elimination:** This transformation removes dead code from a basic block. Dead code is code that is never executed. For example, if a basic block contains the statement `if (x == 0) { y = 1; }` and the value of `x` is always known to be non-zero, then dead code elimination can be used to remove the statement `y = 1;` from the basic block. This can improve the performance of the generated code.
- **Renaming of temporary variables:** This transformation renames temporary variables in a basic block. This can improve the readability of the generated code and can also help to prevent conflicts between names of temporary variables.
- **Interchange of two independent adjacent statements:** This transformation interchanges two independent adjacent statements in a basic block. This can improve the performance of the generated code if the two statements are independent of each other.

(d)

- First set: The first set of a non-terminal symbol is the set of all terminals that can appear at the beginning of a string produced by that non-terminal symbol.
- Follow set: The follow set of a non-terminal symbol is the set of all terminals that can appear immediately after that non-terminal symbol in a string produced by the grammar.

Example:

E -> E + T | T

T -> T * F | F

F -> (E) | id

| Non Terminal | First | Follow |
|---|---|---|
| E | { (, id } | { $, +, ) } |
| T | { (, id } | { $, +, ), * } |
| F | { (, id } | { $, +, ), * } |

(e)

https://www.youtube.com/watch?v=uzjk7UopdSk&list=PLEqLbPpJwryvcGCpfLTLghmDRDOyrIz72&index=43

(f)

A symbol table is a data structure used by compilers to store information about the identifiers used in a program.

The symbol table is used by the compiler to perform a variety of tasks, such as:

- **Name resolution:** The compiler uses the symbol table to resolve identifiers to their corresponding objects. This is necessary to ensure that the compiler can correctly interpret the program.
- **Type checking:** The compiler uses the symbol table to check the types of expressions to ensure that they are compatible. This is necessary to prevent errors at runtime.
- **Scope checking:** The compiler uses the symbol table to verify that identifiers are used in the correct scope. This is necessary to prevent errors at runtime.
- **Optimization:** The compiler can use the symbol table to perform optimizations, such as common subexpression elimination and dead code elimination.

*Structure:*

https://www.youtube.com/watch?v=O-iMkZ7FhKU&list=PLEqLbPpJwryvcGCpfLTLghmDRDOyrIz72&index=19

Ans3) **Bootstrapping** is the process of writing a compiler for a programming language using the language itself. Bootstrapping is used to produce a self-hosting compiler. **Self-hosting compiler** is a type of compiler that can compile its own source code.

There are two common approaches to bootstrapping a compiler:

1. **Top-down bootstrapping:** This approach involves writing a compiler in a different language, such as C or C++, and then using that compiler to compile the target language. This is the most common approach, as it is relatively easy to write a simple compiler in a language like C or C++.
2. **Bottom-up bootstrapping:** This approach involves writing a small subset of the target language first, and then using that subset to compile the rest of the language. This is a more difficult approach, but it can lead to a more efficient compiler.

Some of the advantages of bootstrapping are:

- **Portability:** A self-hosting compiler can be run on any platform that supports the language it is written in.
- **Efficiency:** A self-hosting compiler can be optimized for the target platform, resulting in faster execution times.
- **Flexibility:** A self-hosting compiler can be extended to support new features and languages.

Some of the disadvantages of bootstrapping are:

- **Complexity:** Bootstrapping can be a complex and time-consuming process.
- **Risk:** There is a risk that the bootstrapping process will fail, resulting in a compiler that cannot be used to compile itself.
- **Maintenance:** Self-hosting compilers can be more difficult to maintain than compilers that are written in a different language.

Ans7)

**Runtime Environment**

The runtime environment is a set of data structures and routines that are used to manage the execution of a program. It provides the program with access to memory, files, and other resources, and it handles tasks such as garbage collection, exception handling, and thread synchronization.

There are three main types of runtime environments:

- **Static runtime environment:** A static runtime environment is created when the program is compiled. It is stored in the program's executable file and is used to load the program into memory, initialize its data structures, and start its execution.
- **Stack-based runtime environment:** A stack-based runtime environment is created when a function is called. It is stored on the stack and is used to store the function's local variables, return address, and other data.
- **Dynamic runtime environment:** A dynamic runtime environment is created when a program is executed. It is stored in memory and is used to store the program's global variables, heap data, and other data.

**Activation Record**

The activation record is created when the function is called and is destroyed when the function returns. It is stored on the stack, and its size is determined by the number of local variables that the function declares.

The different fields of an activation record are as follows:

- **Local variables:** The local variables of the function are stored in the activation record.
- **Return address:** The return address is the address of the instruction that called the function. It is stored in the activation record so that the function can return to the caller when it finishes executing.
- **Other data:** The activation record may also contain other data that the function needs to access, such as the function's arguments, its environment, or its stack pointer.

Example

```
def foo(a, b):
  c = a + b
  return c
```

| Field | Value |
|---|---|
| Local variables | `a`, `b`, `c` |
| Return address | Address of the instruction that called `foo` |
| Other data | None |

Ans9)

(a)

Peephole optimization is a local optimization technique, meaning that it only considers small sequences of instructions. This makes it a relatively simple optimization technique to implement, but it can also be very effective in improving the performance of code.

Some examples of peephole optimization are:

- **Redundant instructions:** Peephole optimization can remove redundant instructions. For example, if the same value is loaded into a register twice, the second load can be removed.
- **Combining instructions:** Peephole optimization can combine two instructions into a single instruction. For example, the addition and assignment instructions `a = b + c` can be combined into a single instruction `a += c`.
- **Constant folding:** Peephole optimization can fold constants. For example, the multiplication instruction `a = b * c` can be folded into a constant if `b` and `c` are both constants.

Some of the benefits of peephole optimization:

- **Improved performance:** Peephole optimization can improve the performance of code by removing redundant instructions, combining instructions, and folding constants.
- **Reduced code size:** Peephole optimization can reduce the code size of a program by removing redundant instructions and combining instructions.
- **Simplified code:** Peephole optimization can simplify the code of a program by removing redundant instructions and combining instructions.

Some of the drawbacks of peephole optimization:

- **Can be missed:** Peephole optimization can miss some opportunities for optimization, such as when the compiler cannot see the entire program.
- **Can introduce bugs:** Peephole optimization can introduce bugs if the compiler does not correctly analyse the code.
- **Can be time-consuming:** Peephole optimization can be time-consuming to implement and to perform.

(b)

**Bison** is a parser generator that uses a formal grammar to generate a parser. A formal grammar is a set of rules that define the syntax of a programming language. Bison can generate parsers for a wide variety of programming languages, including C, C++, Java, and Python.

**Flex** is a lexical scanner generator that uses a regular expression to generate a lexical scanner. A lexical scanner is a program that analyzes the input stream of a programming language and converts it into a sequence of tokens. Flex can generate lexical scanners for a wide variety of programming languages, including C, C++, Java, and Python.

(c)

Sure.

**IN and OUT** sets are used in global dataflow analysis to track the flow of data through a program. Global dataflow analysis is a type of dataflow analysis that is used to analyze the flow of data through an entire program.

The **IN** set of a statement contains all of the variables that are defined before the statement, and the **OUT** set of a statement contains all of the variables that are defined by the statement.

Some of the benefits of using IN and OUT sets in global dataflow analysis are:

- **Improved performance:** Global dataflow analysis can improve the performance of a program by eliminating dead code, folding constants, and eliminating common subexpressions.
- **Reduced code size:** Global dataflow analysis can reduce the code size of a program by eliminating dead code and folding constants.

- **Simplified code:** Global dataflow analysis can simplify the code of a program by eliminating dead code and folding constants.

Some of the drawbacks of using IN and OUT sets in global dataflow analysis are:

- **Can be time-consuming:** Global dataflow analysis can be time-consuming to perform, especially for large programs.
- **Can be complex:** Global dataflow analysis can be complex to implement, especially for languages with complex control flow.
- **Can be inaccurate:** Global dataflow analysis can be inaccurate if the program contains side effects.