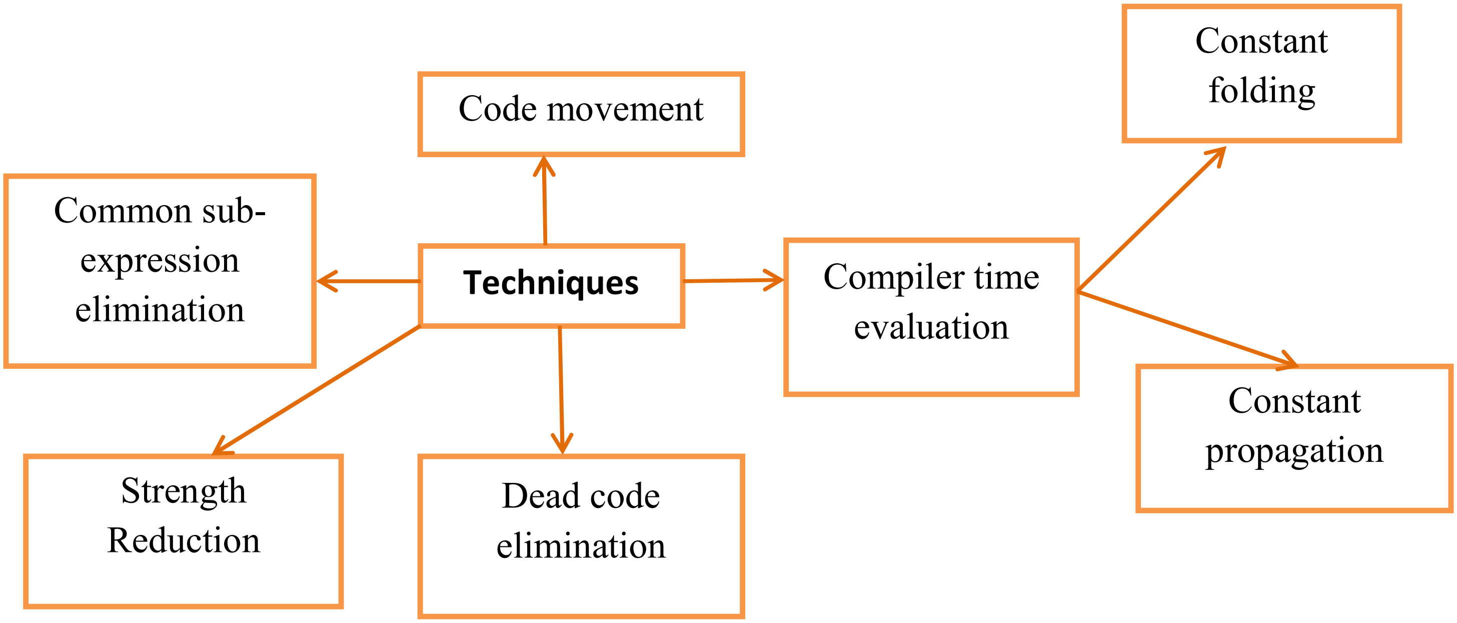
UNIT VI

Code Optimization and Error Detection and Recovery

Code optimization:

- Code optimization means code improvement.
- The code optimization in the synthesis phase is a program transformation technique, which tries to improve the intermediate code by making it consume fewer resources (i.e. CPU, Memory) so that faster running machine code will result.
- Compiler optimizes process should meet the following objectives:
  a) The optimization must be correct, it must not in any way, change the meaning of the program.
  b) Optimization should increase the speed and performance of the program.
  c) The optimization process should not delay the overall compiling process.
- Optimization of the code is performed at the end of the development stage since it reduces readability and adds code that is used to increase the performance.

Code optimizationtechniques:



a) Code movement:

If variables is used in a computation within a loop are not altered, the calculation performed outside of the loop and the results used within a loop.

Eg.

```
 Before optimization                               after optimization
for (i=0;i<=0;i++)                                x=y+x;
{                                                 for(i=0;i<=0;i++)
x=y+z;                                            {
a[i]=2*i;                                          a[i]=2*i;
}                                                 }
```

b) Strength Reduction:

Replaces less efficient instructions with more efficient ones.
Eg. In array subscripting, an add instruction replaces a multiply instruction.

c) Common subexpression elimination:

In common expressions, the same value is recalculating in a subsequent expression. The duplicate expression can be eliminated by using previous values.

Eg.

```
Before                   after
t1=t2*1;                 t1=t2*1;
t2=t3*2;                 t2=t3*2;
t3=t4*3;                 t3=t4*3;
t1=t2*1;
t2=t3*2;
```

d) Compile time evaluation:

i. Constant folding:

It is the process of recognizing and evaluating constant expressions

Eg.

```
    Length =22/7*breath;
```

ii. Constant propagation:

- Constants used in an expressions are combined and new ones are generated
- Some implicit conversions between integers and floating point types are done.

Eg.

```
#define π=3.14
#define r=0.5

void main()
{
……………
……………
a=π*r*;
}
```

e) Dead code elimination:

- Eliminates stores when the value stored is never reference again.
- Eg. If two stores to the same location have no intervening load , the first store is unnecessary and is removed.

Before optimization

```
i=0;
if(i==1)
{
statements
}
```

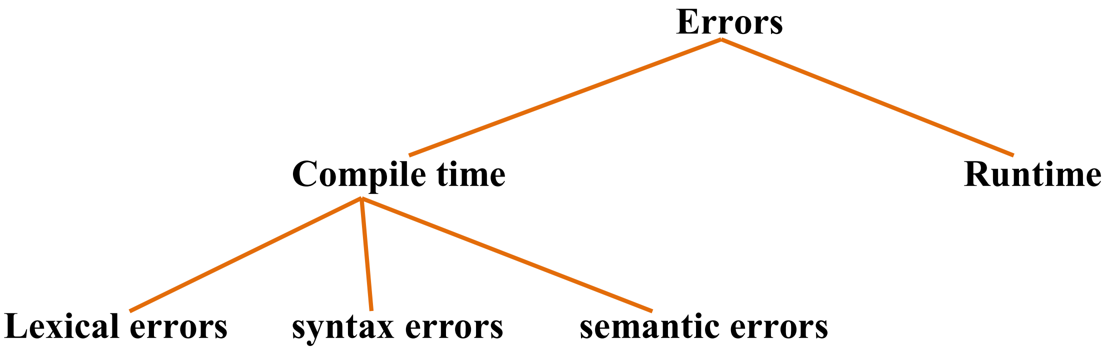
Dead code

After optimization

```
i=0;
```

#**Errors:**

The reaction of compiler towards mistakes in source program is called as errors. There are two types of errors.



a) Lexical phase errors:

- During the lexical analysis phase this type of error can be detected.
- Lexical error is a sequence of characters that does not match of any tokens.
- Lexical phase error is found during the execution of the program.
- The function of lexical analyzer is to carve the stream of characters.
- It means conversion the stream of characters into number of tokens.
- During the conversion the analyzer will detect errors

Eg. Suppose a FORTRAN programmer write a statement

Lexical errors can be

i. Spelling error i.e. misplaced keyword
ii. Exceeding length of identifier or numeric constants.
iii. To remove the character that should be present.
iv. Apperances of illegal character

Eg. int a,b,&;

v. To replace a character with an incorrect character.
vi. Transposition of two characters.

Eg.

```
void main()
{
int x=10,y=20;
char *a;
a=&x;
x=1*ab;
}
```

In this code, 1*ab is neither a number nor an identifier. So this code will show the lexical error.

Syntax errors:

- Most of the error detection and error recovery is centered around syntax analysis.
- Reason is high degree usage of context free grammar.
- During the syntax analysis phase, this type of error appears.
- Syntax error is found during the execution of the program.

- Some syntax errors can be:
  i. Errors in structure
  ii. Missing operator
  iii. Missing semicolon
  iv. Placed colon instead of semicolon
  v. Unbalanced parentheses

- When an invalid calculation enters into a calculator then a syntax error can also occurs. This can be caused by entering several decimal points in one number or by opening brackets without closing them.

Eg.

Using = when "==" is needed
i.e. if (no==200)

Eg.

missing semicolon
int a=5

Eg.

Errors in expressions

```
X=( 3+5 ;          //missing closing parenthesis
Y=3+*5;            //missing arguments between '+' and '*'
```

b) Semantic errors:

- During the semantic analysis phase, this type of errors appears.
- These types of errors can be detected at compile time.
- Most of the compile time errors can arises wrong operator or doing operation in wrong order.
- Most of the compile time errors are scope and declaration errors.

Eg.

Undeclared or multiple declared identifiers.

Type mismatched is another compile time error.

- The undeclared variable and type incompatability are the semantic errors.
- The recovery from the undeclared variable is state forward.
- When we determine undeclared name we make entry for that name in symbol table.
- The symbol table must store entry with attributes.
- The flag in symbol table is set to indicate that entry create semantic errors.
- If semantic error is determine the error message is printed. If error doesn't determine then it will continue with process.

Some semantic errors can be:

- In compatible types of operands
- Undeclared variables
- Not matching of actual argument with formal argument.

Eg.

Use of non- initialized variable

```
int i;
void f(int m)
{
m=t;
}
```

In this code, t is undeclared that's why it shows the semantic error.

Eg.

Type incompability

```
int a=”hello”;        //the types string and int are not incompatible
```

Eg.

Error in expression:

```
String s = “  ”;
int a=5-s;            //operator does not support arguments
```

# loop optimization

- The loop optimization is extended version of code optimization.

i. **Basic block:**

- The first step in loop optimization is to break a source code into basic blocks.
- The basic block is sequence of consecutive statements.
- When the statements are enter into basic blocks they are

executed in sequence of statements into basic blocks is as follows:

#Algorithm name partition into basic blocks

#input the sequence of statements

#output list of basic blocks.

#Method:

a) We determine the statements of basic blocks. The rules are as follows:
   i. First statement is leader
   ii. Any statement which is target is a leader.
b) For each leader construct the basic block.

ii. Flow graph:

- The basic block and relationship is represented using directed graph called as flow graph.
- The nodes of flow graph are the basic block and root node is the relation between two basic blocks.

iii. Code motion:

- The running time of program may be improve we decrease the length of loop.

Eg.

```
While char= " ";
Tochar =getchar();
```

Here getchar() return the next character.

Eg.

```
While (i<100)
{
A=sin(x)/cos(x)+i;
i++;
}
```

Initial code

Optimized code:

```
t=sin(x)/cos(x);
while(i<100)
{
        a=t+i;
        i++;
}
```

iv. Loop unrolling:

- Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program.
- It removes or reduces iterations.
- Loop unrolling increases program’s speed by eliminating loop control instruction and loop test instructions.

I

Initial code:

```
for(int i=0;i<5;i++)

printf(“cocsit \n”);
```

optimized code:

```
printf(“cocsit \n”);

printf(“cocsit \n”);

printf(“cocsit \n”);

printf(“cocsit \n”);

printf(“cocsit \n”);
```

v. Loop jamming:

- Loop jamming is the combining the two or more loops inside a single loop.
- It reduces the time taken to compile the many number of loops.

Eg.

Initial code:

```
For(int i=0;i<5;i++)
{
a=i+5;
        for(int i=0;i<5;i++)
        b=i+10;
}
```

Optimized code:

```
For(int i=0;i<5;i++)
{
a=i+5;
b=i+10;
}
```

UNIT-V

Explain syntax directed definition ?

Explain evaluation of postfix notation ?

Describe parse tree with one example ?

What is intermediate code ?

Explain the implementation of SDT ?

Discuss the various types of three address code ?

UNIT-VI

Explain various loop optimizations ?

Discuss the symantic error ?

Explain the lexical error ?

Explain code optimization technique ?

Explain syntactic phase error ?