**Experiment No - 08**

**Aim**: Generate 3-tuple intermediate code for given infix expression.

**Date:**

**Competency and Practical Skills:**

* Understanding of intermediate code generation and its role in compiler construction
* Ability to write intermediate code for given infix expression.

**Relevant CO:** CO2

**Objectives:**

By the end of this experiment, the students should be able to:

* + Understand the different intermediate code representations and its significance in compiler construction
  + Write intermediate code for given infix expression

**Software/Equipment:** C compiler

**Theory:**

**Three address code** is a type of intermediate code which is easy to generate and can be easily converted to machine code. It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler. The compiler decides the order of operation given by three address code.

# Three address code is used in compiler applications:

**Optimization:** Three address code is often used as an intermediate representation of code during optimization phases of the compilation process. The three address code allows the compiler to analyze the code and perform optimizations that can improve the performance of the generated code.

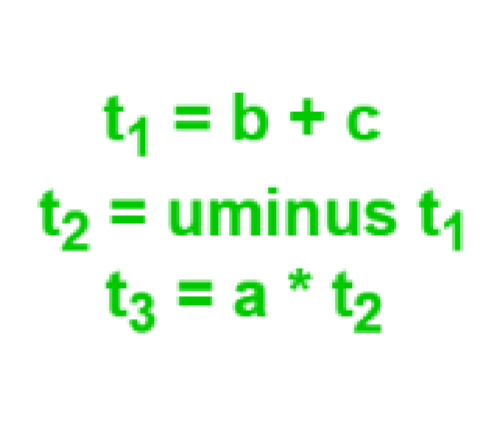
**Code generation:** Three address code can also be used as an intermediate representation of code during the code generation phase of the compilation process. The three address code allows the compiler to generate code that is specific to the target platform, while also ensuring that the generated code is correct and efficient.

**Debugging:** Three address code can be helpful in debugging the code generated by the compiler. Since three address code is a low-level language, it is often easier to read and understand than the final generated code. Developers can use the three address code to trace the execution of the program and identify errors or issues that may be present.

**Language translation:** Three address code can also be used to translate code from one programming language to another. By translating code to a common intermediate representation, it becomes easier to translate the code to multiple target languages. **General representation –** a = b op c

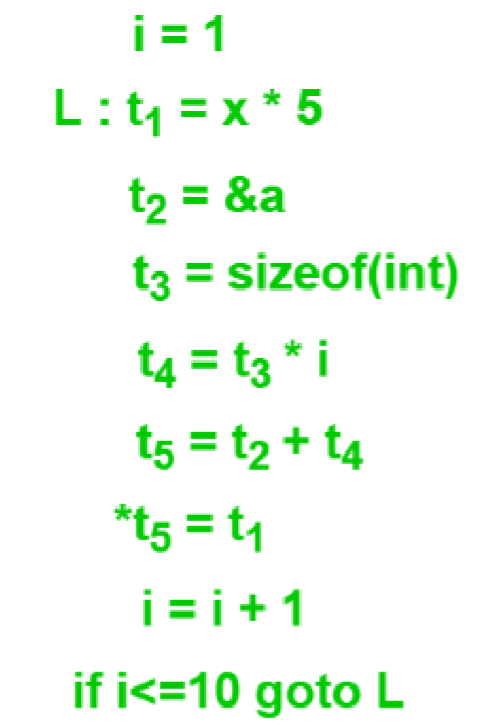
Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator

**Example-1:** Convert the expression a \* – (b + c) into three address code.



**Example-2:** Write three address code for following code for(i = 1; i<=10; i++)

{ a[i] = x \* 5; }



# Implementation of Three Address Code –

There are 3 representations of three address code namely

1. Quadruple
2. Triples
3. Indirect Triples

**1. Quadruple –** It is a structure which consists of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression.

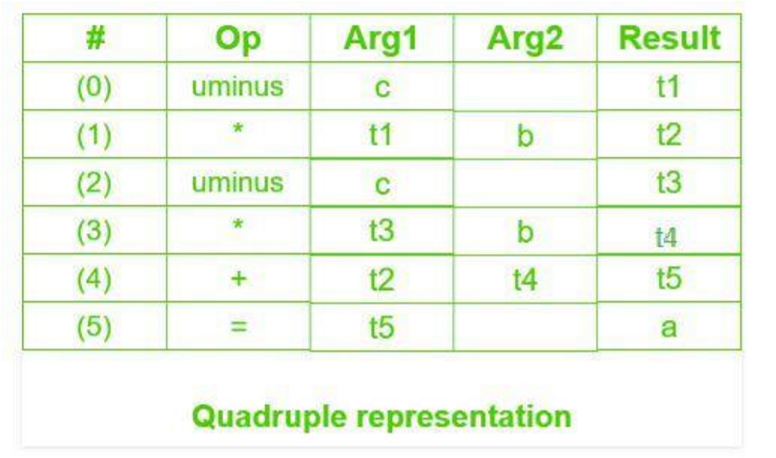
# Advantage –

* Easy to rearrange code for global optimization.
* One can quickly access value of temporary variables using symbol table.

# Disadvantage –

* Contain lot of temporaries.
* Temporary variable creation increases time and space complexity. **Example –** Consider expression a = b \* – c + b \* – c. The three address code is:

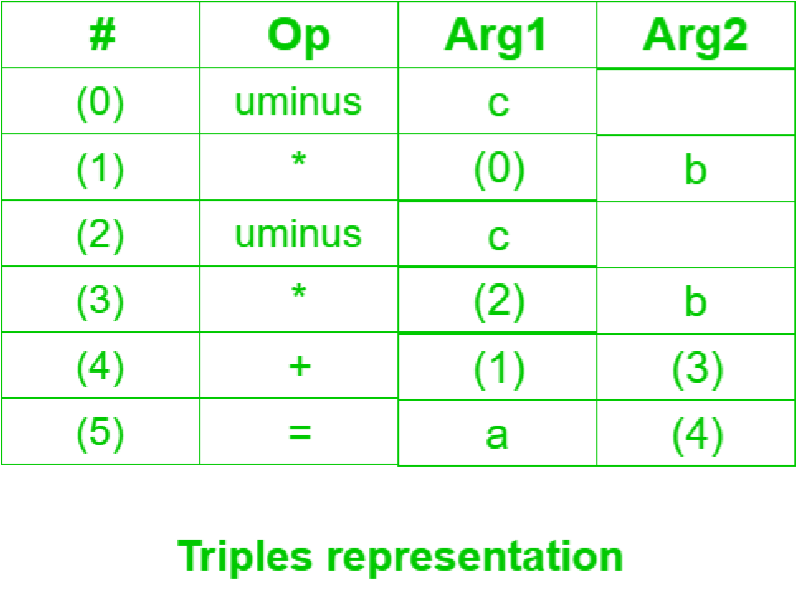
t1 = uminus c t2 = b \* t1 t3 = uminus c t4 = b \* t3 t5 = t2 + t4 a = t5



1. **Triples –** This representation doesn’t make use of extra temporary variable to represent a single operation instead when a reference to another triple’s value is needed, a pointer to that triple is used. So, it consist of only three fields namely op, arg1 and arg2.

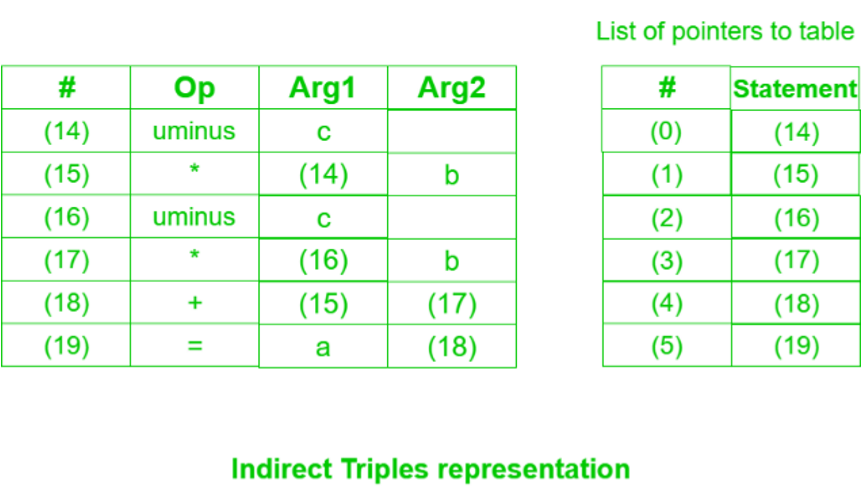
**Disadvantage –** • Temporaries are implicit and difficult to rearrange code.

• It is difficult to optimize because optimization involves moving intermediate code. When a triple is moved, any other triple referring to it must be updated also. With help of pointer one can directly access symbol table entry. **Example –** Consider expression a = b \* – c + b \* – c



1. **Indirect Triples –** This representation makes use of pointer to the listing of all references to computations which is made separately and stored. Its similar in utility as compared to quadruple representation but requires less space than it. Temporaries are implicit and easier to rearrange code.

**Example –** Consider expression a = b \* – c + b \* – c

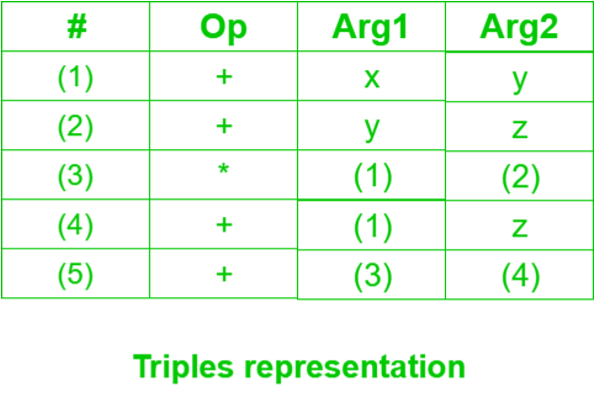
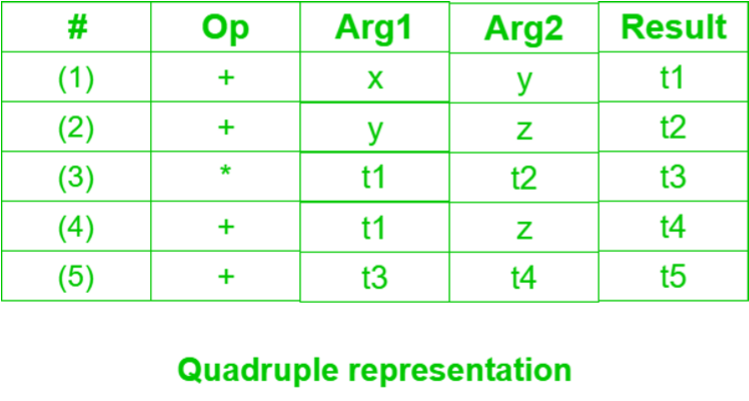


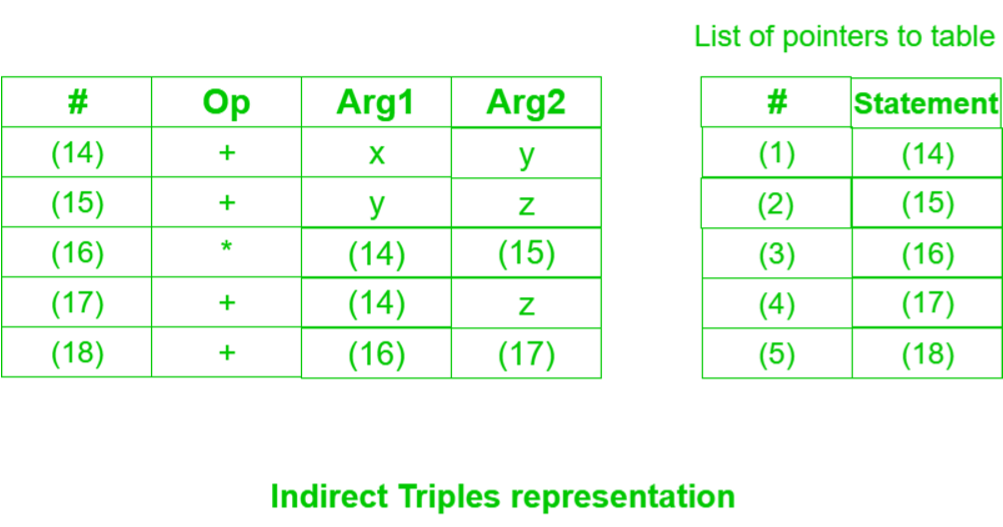
**Question –** Write quadruple, triples and indirect triples for following expression : (x + y) \* (y + z) + (x + y + z)

**Explanation –** The three address code is:

t1 = x + y t2 = y + z t3 = t1 \* t2 t4 = t1 + z

t5 = t3 + t4





**Program:**

void main()

{ printf("Enter the expression:"); scanf("%s",j);

printf("\tThe Intermediate code is:\n"); small(); }

if(j[i]=='\*') printf("\tt%d=%s\*%s\n",c,a,b);

if(j[i]=='/')

printf("\tt%d=%s/%s\n",c,a,b);

if(j[i]=='+') printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')

printf("\tt%d=%s-%s\n",c,a,b);

if(j[i]=='=') printf("\t%c=t%d",j[i-1],--c); sprintf(ch,"%d",c); j[i]=ch[0]; c++; small(); } void small()

{

pi=0;l=0;

for(i=0;i<strlen(j);i++)

{

for(m=0;m<5;m++) if(j[i]==sw[m])

if(pi<=p[m])

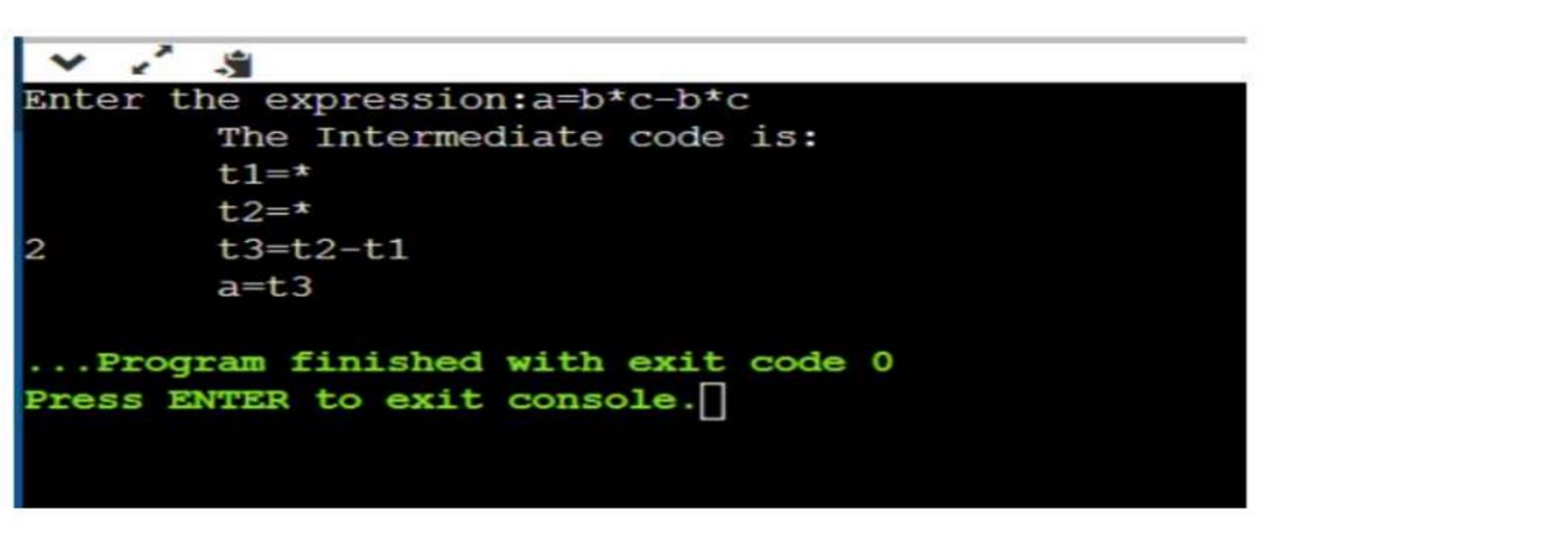
{ pi=p[m]; l=1; k=i;

}

} if(l==1) dove(k); else exit(0);

}

**Observations and Conclusion:**



In the above example the user is asked to write an infix expression and the output is generated intermediate code(3- address code).

**Quiz:**

1. What are the different implementation methods for three-address code?
2. What do you mean by quadruple?
3. Define triple and indirect triple.
4. What is abstract syntax tree?
5. Differentiate parse tree and abstract syntax tree.

**Suggested Reference:**

1. Introduction to Automata Theory, Languages and Computation by John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman.
2. <https://www.geeksforgeeks.org/introduction-to-intermediate-representationir/>
3. <https://cs.lmu.edu/~ray/notes/ir/>

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Knowledge**  **(2)** | | **Problem**  **Recognition**  **(2)** | | **Logic Building (2)** | | **Completeness and accuracy**  **(2)** | | **Ethics (2)** | | **Total** |
| **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** |
| **Marks** |  |  |  |  |  |  |  |  |  |  |  |