Compiler Design Lab

CSS651

Group – 2

Members

Saharsh Ananta Jaiswal – 18CS8061

Pudi Pavan Kumar – 18CS8062

Ayesha Uzma – 18CS8063

Rithik Sureka – 18CS8064

Ravupalli Harsha Vardhan – 18CS8065

Assignment: 6

Implement one Optimizer which would be able to perform : i)Folding, ii)Constant Propagation

iii) elimination of common sub expression

Software Used: Code::Blocks(IDE)

Theory:

Constant Folding: If operation are known at compile time perform the operation statically.

Ex: int x = (2+3) – 8;

After optimization becomes – int x = -3;

Constant Propagation: If the value of a variable is known to be a constant, replace the use of the variable by that constant. Value of the variable must be propagated forward from the point of assignment. This is a substitution operation.

- Example:

```
int x = 5;
int y = x * 2;
int z = a[y];
```
➔
```
int y = 5 * 2;
int z = a[y];
```
➔
```
int y = 10;
int z = a[y];
```
➔
```
int z = a[10];
```

Common sub-expression Elimination: Replace an expression with previously stored evaluations of that expression. Almost always improves performance. But sometimes, it might be less expensive to re-compute an expression, rather than to allocate another register to hold its value (or to store it in memory and later reload it).

- Example:

```
[a + i*4] = [a + i*4] + 1
```

- Common subexpression elimination removes the redundant add and multiply:

```
t = a + i*4; [t] = [t] + 1
```

Code:

Start here | question6.c

```c
#include<stdio.h>
#include<conio.h>
#include<string.h>

struct expression
{
    char l;
    char r[20];
}
op[10],pr[10];

//function to check if the string argument consist of only numbers
int digits_only(const char *s)
{
    while (*s) {
        if (isdigit(*s++) == 0) return 0;
    }

    return 1;
}

// A utility function to check if a given character is operand
int isOperand(char c) { return (c >= '0' && c <= '9'); }

// utility function to find value of and operand
int value(char c) {  return (c - '0'); }

// This function evaluates simple expressions. It returns -1 if the
// given expression is invalid.
int evaluate(char *exp)
{
    // Base Case: Given expression is empty
    if (*exp == '\0')  return -1;

    // The first character must be an operand, find its value
    int res = value(exp[0]);

    // Traverse the remaining characters in pairs
    for (int i = 1; exp[i]; i += 2)
    {
        // The next character must be an operator, and
        // next to next an operand
        char opr = exp[i], opd = exp[i+1];

        // If next to next character is not an operand
        if (!isOperand(opd))  return -1;

        // Update result according to the operator
        if (opr == '+')       res += value(opd);
        else if (opr == '-')  res -= value(opd);
        else if (opr == '*')  res *= value(opd);
        else if (opr == '/')  res /= value(opd);

        // If not a valid operator
        else                  return -1;
    }
    return res;
}


void main()
{
```

```
    int a,i,k,j,n,z=0,m,q,len,len2,v;
    char *p,*l,*exp,*new_exp;
    char temp,t;
    char *tem;
    printf("Enter the Number of Values:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("left: ");
        op[i].l=getche();
        printf("\tright: ");
        scanf("%s",op[i].r);
    }

    printf("Intermediate Code\n") ;
    for(i=0;i<n;i++)
    {
        printf("%c=",op[i].l);
        printf("%s\n",op[i].r);
    }

    //constant propagation optimization if any
    for(i=0;i<n-1;i++)
    {
        temp = op[i].l;
        int constant = digits_only(op[i].r);
        if(constant)
        {
            for(j=i+1;j<n;j++)
            {
              exp = op[j].r;
              p = strchr(exp,temp);
              if(!p)
                  break;
              len = strlen(exp);
              m=0;
              for(k=0;k<len;k++)
              {
                  if(exp[k]==temp)
                  {
                      len2 = strlen(op[i].r);
                      for(v=0;v<strlen(op[i].r);v++)
                      {
                          new_exp[m] = op[i].r[v];
                          m++;
                      }
                  }
                  else
                  {
                      new_exp[m] = exp[k];
                      m++;
                  }
              }
              new_exp[m] = '\0';
              strcpy(op[j].r,new_exp);
           }
       }
   }
   m=0;
   printf("After Constant Propagation Optimization\n");
   for(i=0;i<n;i++)
```

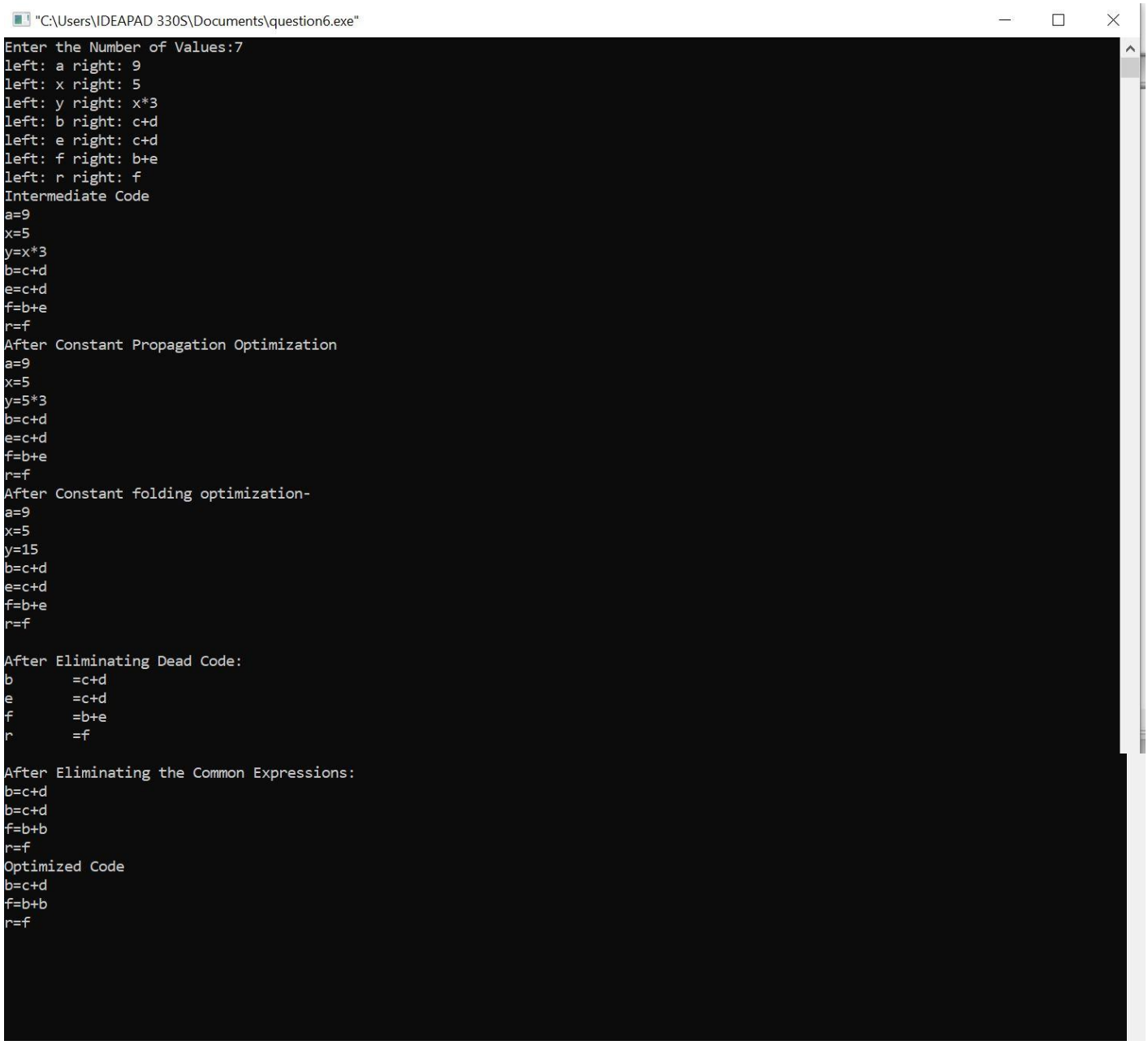
```
    {
        printf("%c=",op[i].l);
        printf("%s\n",op[i].r);
    }

    //eliminates constant folding
    for(i=0;i<n-1;i++)
    {
        exp = op[i].r;
        len = strlen(exp);
        int flag=0;
        int result = evaluate(exp);

        if(result!=-1)
        {
            //convert the integer to string
            char exp2[20];
            sprintf(exp2, "%d", result);
            //printf("%s\n",exp2);
            strcpy(op[i].r,exp2);
        }
    }
    printf("After Constant folding optimization-\n");
    for(i=0;i<n;i++)
    {
        printf("%c=",op[i].l);
        printf("%s\n",op[i].r);
    }
    //eliminates dead code
    for(i=0;i<n-1;i++)
    {
        temp=op[i].l;
        for(j=i+1;j<n;j++)
        {
            p=strchr(op[j].r,temp);
            if(p)
            {
                pr[z].l=op[i].l;
                strcpy(pr[z].r,op[i].r);
                z++;
            }
        }
    }
    pr[z].l=op[n-1].l;
    strcpy(pr[z].r,op[n-1].r);
    z++;
    printf("\nAfter Eliminating Dead Code: \n");
    for(k=0;k<z;k++)
    {
        printf("%c\t=",pr[k].l);
        printf("%s\n",pr[k].r);
    }
    //sub-expression elimination
    for(m=0;m<z;m++)
    {
        tem=pr[m].r;
        for(j=m+1;j<z;j++)
        {
            p=strstr(tem,pr[j].r);
            if(p)
            {
                t=pr[j].l;
```

```
                pr[j].l=pr[m].l;
                for(i=0;i<z;i++)
                {
                    l=strchr(pr[i].r,t) ;
                    if(l)
                    {
                        a=l-pr[i].r;
                        //printf("pos: %d",a);
                        pr[i].r[a]=pr[m].l;
                    }
                }
            }
        }
    }
    printf("\nAfter Eliminating the Common Expressions: \n");
    for(i=0;i<z;i++)
    {
        printf("%c=",pr[i].l);
        printf("%s\n",pr[i].r);
    }
    //duplicate production elimination
    for(i=0;i<z;i++)
    {
        for(j=i+1;j<z;j++)
        {
            q=strcmp(pr[i].r,pr[j].r);
            if((pr[i].l==pr[j].l)&& !q)
            {
                pr[i].l='\0';
                strcpy(pr[i].r,"\0");
            }
        }
    }
    printf("Optimized Code\n");
    for(i=0;i<z;i++)
    {
        if(pr[i].l!='\0')
        {
            printf("%c=",pr[i].l);
            printf("%s\n",pr[i].r);
        }
    }
    getch();
}
```

Output:


"C:\Users\IDEAPAD 330S\Documents\question6.exe"
Enter the Number of Values:7
left: a right: 9
left: x right: 5
left: y right: x*3
left: b right: c+d
left: e right: c+d
left: f right: b+e
left: r right: f
Intermediate Code
a=9
x=5
y=x*3
b=c+d
e=c+d
f=b+e
r=f
After Constant Propagation Optimization
a=9
x=5
y=5*3
b=c+d
e=c+d
f=b+e
r=f
After Constant folding optimization-
a=9
x=5
y=15
b=c+d
e=c+d
f=b+e
r=f
After Eliminating Dead Code:
b =c+d
e =c+d
f =b+e
r =f
After Eliminating the Common Expressions:
b=c+d
b=c+d
f=b+b
r=f
Optimized Code
b=c+d
f=b+b
r=f