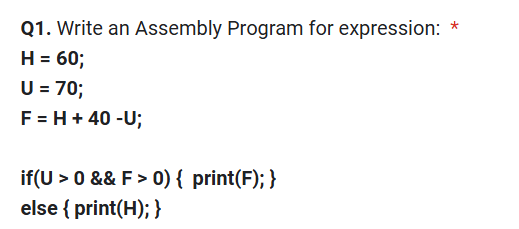
**Roll No: 1903001**

**Lab Performance Evaluation [02]**

**Lab Task Q1**

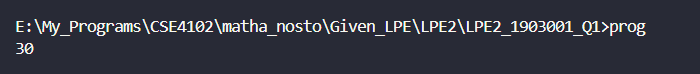
**Question:**



**Solution (Bold your own written code):**

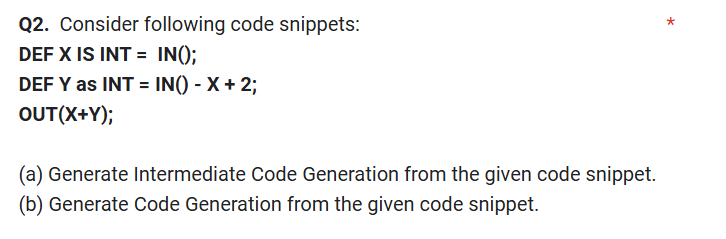
|  |
| --- |
| ;start -1  .686  .model flat, c  include E:\masm32\include\msvcrt.inc  includelib E:\masm32\lib\msvcrt.lib  .stack 100h  printf PROTO arg1:**Ptr** *Byte*, printlist:VARARG  scanf PROTO arg2:**Ptr** *Byte*, inputlist:VARARG  .data  output\_integer\_msg\_format *byte* "%d", 0Ah, 0  inp\_msg\_format *byte* "%s", 0  output\_msg\_format *byte* "%s", 0Ah, 0  input\_integer\_format *byte* "%d",0  number *sdword* ?  .code  main proc  push ebp  mov ebp, esp  sub ebp, 100  mov *dword ptr* [ebp-0], 60 ;H  mov *dword ptr* [ebp-4], 70;U  mov eax, [ebp-0]  mov ebx, [ebp-4]  add eax, 40  sub eax, ebx  mov *dword ptr* [ebp-8], eax ; F  mov eax, [ebp-4]  cmp eax, 0  jng ELSE\_  mov eax, [ebp-8]  cmp eax, 0  jng ELSE\_  push [ebp-8]  push [ebp-4]  push [ebp-0]  push ebp  INVOKE printf, ADDR output\_integer\_msg\_format, eax  pop ebp  pop [ebp-0]  pop [ebp-4]  pop [ebp-8]  jmp EXIT\_  ELSE\_:  push [ebp-8]  push [ebp-4]  push [ebp-0]  push ebp  mov eax, [ebp]  INVOKE printf, ADDR output\_integer\_msg\_format, eax  pop ebp  pop [ebp-0]  pop [ebp-4]  pop [ebp-8]  EXIT\_:      ret  main endp  end |

**Output (Screen/SnapShot):**



**Lab Task Q2**

**Question:**



**Solution (Bold your own written code):**

**Lexer.l**

|  |
| --- |
| %option *noyywrap*  %{      #define INT\_TYPE 1      #include <stdio.h>      #include <stdlib.h>      #include <string.h>      #include "parser.tab.h"        int lineno = 1; // initialize to 1      void yyerror();  %}  letter [a-zA-Z]  digit [0-9]  ID ({letter})({letter}|{digit})\*  ICONST {digit}+  %%  "IS" {return(IS);}  "IN" {return(SCAN);}  "INT" {return(INT);}  "OUT" {return(PRINT);}  "DEF" {return(DEF);}  "as" {return(AS);}  ";" {return(SEMI);}  {ID} {strcpy(yylval.str\_val, yytext); return(ID);}  "-" {return(MINUS);}  "+" {return(PLUS);}  {ICONST} {yylval.int\_val=atoi(yytext); return(ICONST);}  ")" {return(RP);}  "(" {return(LP);}  "=" {return(ASSIGN);}  "\n"        { lineno += 1; }  [ \t\r\f]+  .       { yyerror("Unrecognized character"); } |

Parser.y

%{

    #include <stdio.h>

    #include <stdlib.h>

    #include <string.h>

    #include "symtab.c"

    #include "codeGen.c"

    void yyerror();

    extern int lineno;

    extern int yylex();

%}

%union

{

    char str\_val[100];

    int int\_val;

}

%token SCAN PRINT MINUS PLUS RP LP ASSIGN IS DEF AS SEMI

%token<str\_val> ID

%token<int\_val> ICONST INT

%left PLUS MINUS

%type<int\_val> type

%start code

%%

code: {gen\_code(START, -1);}  statements  {gen\_code(HALT, -1);};

statements: statements statement | ;

statement:  printf

            | assignment;

printf: PRINT LP pexp RP SEMI

    {

        char\* name = "\_\_TEMP\_\_";

        int addr = idcheck(name);

        if(addr==-1) {

            insert(name, INT\_TYPE);

            addr = idcheck(name);

        }

        gen\_code(PRINT\_INT\_VALUE, addr);

    };

assignment: DEF ID IS type ASSIGN exp SEMI

        {

            int addr = idcheck($2);

            if(addr==-1) {

                insert($2, $4);

                addr = idcheck($2);

            }

            gen\_code(STORE, addr);

        }

        | DEF ID AS type ASSIGN exp SEMI

        {

            {

            int addr = idcheck($2);

            if(addr==-1) {

                insert($2, $4);

                addr = idcheck($2);

            }

            gen\_code(STORE, addr);

        }

        }

        ;

pexp: pexp PLUS T

    {

        gen\_code(ADD, -1);

        char\* name = "\_\_TEMP\_\_";

        int addr = idcheck(name);

        if(addr==-1) {insert(name, INT\_TYPE);

        addr = idcheck(name);}

        gen\_code(STORE, addr);

    }

    | T;

exp: exp PLUS T

    {

        gen\_code(ADD, -1);

    }

    | exp MINUS T

    {

        gen\_code(SUB, -1);

    }

    | T

    ;

T: ID

    {

        int addr = idcheck($1);

        if(addr!=-1) gen\_code(LD\_VAR, addr);

        else exit(0);

    }

   | ICONST

   {

        gen\_code(LD\_INT, $1);

   }

   | scanf

   ;

scanf: SCAN LP RP

    {

        char\* name = "\_\_TEMP\_\_";

        int addr = idcheck(name);

        if(addr==-1) {

            insert(name, INT\_TYPE);

            addr = idcheck(name);

        }

        gen\_code(SCAN\_INT\_VALUE, addr);

        gen\_code(LD\_VAR, addr);

    };

type: INT{$$=INT\_TYPE;};

%%

void yyerror ()

{

    printf("Syntax error at line %d\n", lineno);

    exit(1);

}

int main (int *argc*, char \**argv*[])

{

    yyparse();

    printf("Parsing finished!\n");

    printf("============= INTERMEDIATE CODE===============\n");

    print\_code();

    printf("============= ASM CODE===============\n");

    print\_assembly();

    return 0;

**Output (Screen/SnapShot):**

**Output.txt**

**In line no 1, ID \_\_TEMP\_\_ is not declared.**

**In line no 1, Inserting \_\_TEMP\_\_ with type INT\_TYPE in symbol table.**

**In line no 1, ID X is not declared.**

**In line no 1, Inserting X with type INT\_TYPE in symbol table.**

**In line no 2, ID Y is not declared.**

**In line no 2, Inserting Y with type INT\_TYPE in symbol table.**

**Parsing finished!**

**============= INTERMEDIATE CODE===============**

**0: start -1**

**1: scan\_int\_value 0**

**2: ld\_var 0**

**3: store 1**

**4: scan\_int\_value 0**

**5: ld\_var 0**

**6: ld\_var 1**

**7: sub -1**

**8: ld\_int 2**

**9: add -1**

**10: store 2**

**11: ld\_var 1**

**12: ld\_var 2**

**13: add -1**

**14: store 0**

**15: print\_int\_value 0**

**16: halt -1**

**============= ASM CODE===============**

**;start -1**

**.686**

**.model flat, c**

**include E:\masm32\include\msvcrt.inc**

**includelib E:\masm32\lib\msvcrt.lib**

**.stack 100h**

**printf PROTO arg1:Ptr Byte, printlist:VARARG**

**scanf PROTO arg2:Ptr Byte, inputlist:VARARG**

**.data**

**output\_integer\_msg\_format byte "%d", 0Ah, 0**

**output\_string\_msg\_format byte "%s", 0Ah, 0**

**input\_integer\_format byte "%d",0**

**number sdword ?**

**.code**

**main proc**

**push ebp**

**mov ebp, esp**

**sub ebp, 100**

**mov ebx, ebp**

**add ebx, 4**

**;scan\_int\_value 0**

**push eax**

**push ebx**

**push ecx**

**push edx**

**push [ebp-8]**

**push [ebp-4]**

**push [ebp-0]**

**push ebp**

**INVOKE scanf, ADDR input\_integer\_format, ADDR number**

**pop ebp**

**pop [ebp-0]**

**pop [ebp-4]**

**pop [ebp-8]**

**mov eax, number**

**mov dword ptr [ebp-0], eax**

**pop edx**

**pop ecx**

**pop ebx**

**pop eax**

**;ld\_var 0**

**mov eax, [ebp-0]**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;store 1**

**mov dword ptr [ebp-4], eax**

**;scan\_int\_value 0**

**push eax**

**push ebx**

**push ecx**

**push edx**

**push [ebp-8]**

**push [ebp-4]**

**push [ebp-0]**

**push [ebp+4]**

**push ebp**

**INVOKE scanf, ADDR input\_integer\_format, ADDR number**

**pop ebp**

**pop [ebp+4]**

**pop [ebp-0]**

**pop [ebp-4]**

**pop [ebp-8]**

**mov eax, number**

**mov dword ptr [ebp-0], eax**

**pop edx**

**pop ecx**

**pop ebx**

**pop eax**

**;ld\_var 0**

**mov eax, [ebp-0]**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;ld\_var 1**

**mov eax, [ebp-4]**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;sub -1**

**sub ebx, 4**

**mov eax, [ebx]**

**sub ebx, 4**

**mov edx, [ebx]**

**sub edx, eax**

**mov dword ptr [ebx], edx**

**add ebx, 4**

**mov eax, edx**

**;ld\_int 2**

**mov eax, 2**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;add -1**

**sub ebx, 4**

**mov eax, [ebx]**

**sub ebx, 4**

**mov edx, [ebx]**

**add eax, edx**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;store 2**

**mov dword ptr [ebp-8], eax**

**;ld\_var 1**

**mov eax, [ebp-4]**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;ld\_var 2**

**mov eax, [ebp-8]**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;add -1**

**sub ebx, 4**

**mov eax, [ebx]**

**sub ebx, 4**

**mov edx, [ebx]**

**add eax, edx**

**mov dword ptr [ebx], eax**

**add ebx, 4**

**;store 0**

**mov dword ptr [ebp-0], eax**

**;print\_int\_value 0**

**push eax**

**push ebx**

**push ecx**

**push edx**

**push [ebp-8]**

**push [ebp-4]**

**push [ebp-0]**

**push [ebp+4]**

**push [ebp+8]**

**push [ebp+12]**

**push [ebp+16]**

**push ebp**

**mov eax, [ebp-0]**

**INVOKE printf, ADDR output\_integer\_msg\_format, eax**

**pop ebp**

**pop [ebp+16]**

**pop [ebp+12]**

**pop [ebp+8]**

**pop [ebp+4]**

**pop [ebp-0]**

**pop [ebp-4]**

**pop [ebp-8]**

**pop edx**

**pop ecx**

**pop ebx**

**pop eax**

**;halt -1**

**add ebp, 100**

**mov esp, ebp**

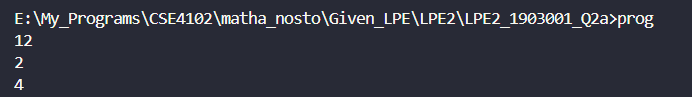
**pop ebp**

**ret**

**main endp**

**end**

**ScreenShot:**

****