Cairo University

Compilers Project Document

Team :16

Name	Section	B.N
Aly Ramzy Hassan	1	35
Mohamed Ahmed Ibrahim	2	9
Nour Ahmed	2	29
Hager Ahmed	2	32
Ali Khaled	1	34

Supported data types

1. int : to represent integer numbers.
2. float : to represent decimal numbers, or numbers with exponent part.
3. char : to represent a single character.
4. bool : true/false.
5. void : for functions as a return type.

We support type conversion

Syntax

1. Variables and constants declaration:

   $To\ define\ a\ constant$:

   $const\ dataType\ identifier\ =\ value;$

   $Example$: $const\ int\ x\ =\ 5$;

   $To\ define\ a\ variable$:

   $dataType\ identifier\ =\ value;$

   $Example$: $int\ \ x\ =\ 5$;

2. Mathematical and logical expressions.

   Supported operations :

   - Addition , Subtraction , Multiplication , Division , Modulus .
   - & , | , ^ , || , && .
   - < , > , >= , <= .
   - ~ , ! .
   - Any level of pranthess/complexity .

3. Assignment statement

   $variable\ =\ expression$;

   $Example$: $x\ =\ 5$;

4. If-then-else statement

Supports conditional statements like C language.

```
if(condition){
    code
}
else{
    if(condition)
        statement;
}
```

5. Switch statement

C-like.

```
switch(expression){
    case value1:
        code
    case value2:
        code
    default:
        code
}
```

6. Loops

- While/for loops are C-like.

```
while(expression){
    code
}
```

```
for(headers; expression; expression){
    code
}
```

- Repeat-until has the following syntax:

```
repeat{
    code
}until(expression);
```

7. Block structures

Each block represents a scope.
Variables within a scope are not visible to other scopes, excluding some semantic cases that would be determined later.

```
{
    code
    {
        scope 1
    }
    {
        scope 2
    }
}
```

8. Functions

We don’t support default parameter’s values.
Functions have the following syntax:

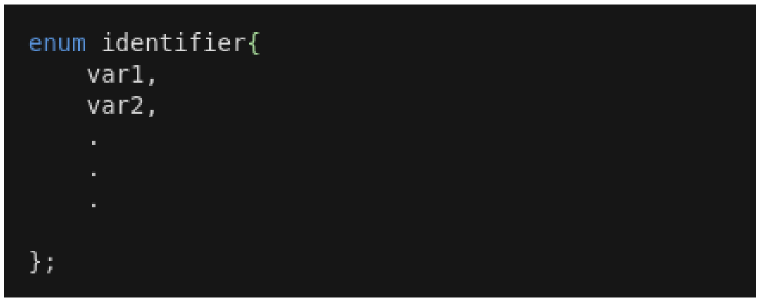
```
return_type function_name(parameters){
    code
}
```

For example:

```
int max(int x, int y){
    if(x > y)
        return x;
    else
        return y;
}
```

9. Enums

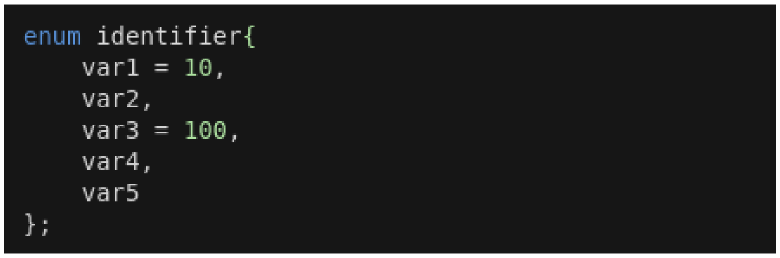
Enums have the following syntax



```
enum identifier{
    var1,
    var2,
    .
    .
    .

};
```

Enums' name could be empty.
Variables could have a value.
For example:



```
enum identifier{
    var1 = 10,
    var2,
    var3 = 100,
    var4,
    var5
};
```

Quadruples

Quadruple	Description
ADD_<type>	Take the last two values in the stack(S1, S2), then push S2 + S1
SUB_<type>	Take the last two values in the stack(S1, S2), then push S2 - S1
MUL_<type>	Take the last two values in the stack(S1, S2), then push S2 * S1
DIV_<type>	Take the last two values in the stack(S1, S2), then push S2 / S1
MOD_<type>	Take the last two values in the stack(S1, S2), then push S2 % S1
NEG_<type>	Take the last value in the stack(S1), then push -S1

AND_<type>	Take the last two values in the stack(S1, S2), then push S2 & S1
OR_<type>	Take the last two values in the stack(S1, S2), then push S2 | S1
XOR_<type>	Take the last two values in the stack(S1, S2), then push S2 ^ S1
NOT_<type>	Take the last value in the stack(S1), then push !S1
GT_<type>	Take the last two values in the stack(S1, S2), then push ”true” if S2 > S1, “false” otherwise
GTE_<type>	Take the last two values in the stack(S1, S2), then push ”true” if S2 >= S1, “false” otherwise
LT_<type>	Take the last two values in the stack(S1, S2), then push ”true” if S2 < S1, “false” otherwise
LTE_<type>	Take the last two values in the stack(S1, S2), then push ”true” if S2 <= S1, “false” otherwise
EQ_<type>	Take the last two values in the stack(S1, S2), then push ”true” if S2 = S1, “false” otherwise
NEQ_<type>	Take the last two values in the stack(S1, S2), then push ”true” if S2 != S1, “false” otherwise
<type>_TO_<type>	Take the last value in the stack(S1), then push it after conversion from left type to right type
PUSH_<type> <value>	Push “value” to the stack
POP_<type> <dst>	Pop the top of the stack into “dst”

JMP <label>	Unconditional jump to “label”
JNZ_<type> <label>	Take the last value in the stack(S1), then jump if S1 != 0
JZ_<type> <label>	Take the last value in the stack(S1), then jump if S1 = 0
PROC <identifier>	Start of procedure with name “identifier”
CALL <ident>	Call of procedure with name “identifier”
RET	Return from procedure

Semantic errors

1. Variable declaration conflicts in the same scope.
2. Variables used before being initialized and unused variables.
3. The addition of type conversion quadruples to coupe with operators’ semantic requirements.

Bonus Semantic errors

1. Changing the value of a constant.
2. Multiple “default” labels in switch scope.
3. Multiple case-labels with the same constant expression in switch scope.
4. Return type mismatch in functions.
5. Calling a non-function type i.e. variable.
6. Errors in function calls i.e. more/less arguments, non-matching parameter’s type.

We implemented a fancy GUI with server side with the ability to upload code files.

Work load

Aly Ramzy	● “Variables” functionalities. ● “Functions” functionalities. ● GUI.
Mohamed Ahmed Ibrahim	● Conditional statements. ● Loops. ● Expressions.
Nour Ahmed	● “Variables” functionalities. ● “Functions” functionalities. ● GUI.
Hager Ahmed	● Conditional statements. ● Loops. ● Expressions.
Ali Khaled	● Enums