Parte 2

Ideia de modificação do programa

Originalmente, nós pensamos em simplesmente tratar os tipos criados (*type*) como tipos originais (e.g. integer), mas logo percebemos que isso inviabilizaria a comparação entre os níveis. Por exemplo, para entender que banana é compatível com integer, precisamos de um nível de informação a mais no tipo banana que nos dê esse conhecimento.

A solução foi criar uma estrutura que represente os tipos criados (*type)* e modificar a função de validação de tipos (*checaTipos*) para incluir essa verificação especial para tipos criados, que é um pouco mais complexa que validar tipos originais (mais informações abaixo).

Modificações realizadas no código do compilador

As principais modificações no código foram relacionadas à declaração e ao tratamento de tipos. As tabelas abaixo representam um recorte dos principais trechos que foram modificados no código do compilador, bem como uma breve descrição da solução.

Foi criada uma *struct* para representar um *type*, contendo valor, identificador do subtipo (0 para desconhecido, 1 para integer e 2 para boolean) e nome.

```
typedef struct tipos_t
{
      int valor;
      int ValorsubTipo;
      char nome[TAMANHO_IDENTIFICADOR];
} tipos;
```

No arquivo *compilador.y*, a regra `parte_declara_tipos_helper` foi adicionada ao bloco, quatro novas regras foram inseridas (*Tabela 1*) para tratar a sintaxe de declaração de tipos:

```
parte_declara_tipos_helper:
    | parte_declara_tipos
;

parte_declara_tipos:
    TYPE definicao_tipo_helper PONTO_E_VIRGULA
;

definicao_tipo_helper:
    definicao_tipo
```

```
    | definicao_tipo_helper PONTO_E_VIRGULA definicao_tipo
;

definicao_tipo:
    IDENT
    {
        strcpy(l_oper, token);
    }
    IGUAL
    IDENT
    {
        for (int i = 0; i < taMvetorDeTipos; ++i)
        {
            if (!strcmp(vetorDeTipos[i].nome, l_oper))
            {
                char str[DEFAULT_STRING_LENGTH];
                sprintf(str, "Tipo ja declarado: %s", l_oper);
                imprimeErro(str);
            }
        }
        int existe=0;
        for (int i = 0; i < taMvetorDeTipos; ++i)
        {
            if (!strcmp(vetorDeTipos[i].nome, token))
            {
                existe = 1;
                vetorDeTipos[taMvetorDeTipos].valor = taMvetorDeTipos;
                vetorDeTipos[taMvetorDeTipos].ValorsubTipo =
vetorDeTipos[i].valor;
                strcpy(vetorDeTipos[taMvetorDeTipos].nome, l_oper);
                strcpy(valorTipoPraString[taMvetorDeTipos], l_oper);
                controleSubtipo[taMvetorDeTipos] =
vetorDeTipos[taMvetorDeTipos].ValorsubTipo;
                taMvetorDeTipos++;
            }
        }
    }
;
```

A função *checaTipos* verifica se operandos envolvidos em uma operação possuem tipos compatíveis. Essa função foi alterada para incluir a verificação de tipos criados (*types*).

Basicamente, cada tipo criado (*type*) possui um subtipo do qual foi baseado (e.g. integer) e as únicas operações válidas são entre tipos iguais (e.g. banana e banana) ou subtipos compatíveis com tipos padrões (e.g. banana e integer):

```
void checaTipos()
{
    if (typesStack.topo &&
        typesStack.elementos[typesStack.topo] !=
typesStack.elementos[typesStack.topo-1] )
    {
        int tipo1, tipo2;
        tipo1 = typesStack.elementos[typesStack.topo];
        tipo2 = typesStack.elementos[typesStack.topo-1];
        // tests if subtype match type (or the other way around)
        if (controleSubtipo[tipo1] == tipo2 || controleSubtipo[tipo2] == tipo1)
        {

            removeElemento(&typesStack, 2);
            insereElemento(&typesStack, controleSubtipo[tipo1] );
            return;
        }
        char str[DEFAULT_STRING_LENGTH];
        sprintf(str, "Tipos incompativeis: %s/%s",
        tabelaTipos(typesStack.elementos[typesStack.topo]),
        tabelaTipos(typesStack.elementos[typesStack.topo-1]));
        imprimeErro(str);
    }
    removeElemento(&typesStack, 1);

}
```

Por fim, no *Main* foram incluídos trechos de inicialização dos vetores de controle. Nós usamos basicamente dois vetores: *vetorDeTipos* e *valorTipoParaString*. O *vetorDeTipos* tem a função de guardar os *types* criados, enquanto o *valorTipoParaString* apenas mapeia o valor do subtipo para a string correspondente, tornando possível apresentar mensagens de erros mais descritivas (e.g. “Erro na linha 13 - Tipos incompatíveis: laranja/banana”).

```
    vetorDeTipos[0].valor = 0;
    vetorDeTipos[0].ValorsubTipo = 0;
    strcpy(vetorDeTipos[0].nome, "desconhecido");
    vetorDeTipos[1].valor = 1;
    vetorDeTipos[1].ValorsubTipo = 1;
```

```
    strcpy(vetorDeTipos[1].nome, "integer");
    vetorDeTipos[2].valor = 2;
    vetorDeTipos[2].ValorsubTipo = 2;
    strcpy(vetorDeTipos[2].nome, "boolean");
    taMvetorDeTipos=3;
    controleSubtipo[0]=0;
    controleSubtipo[1]=1;
    controleSubtipo[2]=2;
    strcpy(valorTipoPraString[0], "desconhecido");
    strcpy(valorTipoPraString[1], "integer");
    strcpy(valorTipoPraString[2], "boolean");
```

Maiores problemas ao implementar Type

A parte que precisou de mais atenção foi na verificação dos tipos e subtipos para determinar se as operações são válidas (função *checaTipos*). Além de verificar se os tipos são iguais, como era feito até então, agora é preciso verificar a compatibilidade dos tipos originais com os tipos criados (e.g. banana e integer) e a incompatibilidade entre os tipos criados (e.g. banana e morango).