Proyecto ATLyS: Backend + Informe

Integrante: Gazzaneo, Lautaro Nicolas 61484

Profesores:

Agustin Golmar (mgolmar@itba.edu.ar)
Agustin Benvenuto (abenvenuto@itba.edu.ar)
Rodrigo Ramele (rramele@itba.edu.ar)
Ana Maria Arias Roig

Tabla de Contenidos:

Introducción:

La idea de este proyecto es generar a partir de código una Character/Item/Monster/Merchant Sheet para D&D 5e. Puede completarse con datos que tienes o los necesarios para conseguir un “sheet” completo.

Esta hoja de personaje sería convertida a html donde aparecería un archivo de este tipo que al ponerlo en el navegador te diría toda la información como si fuera una hoja propia de Wizards of the Coast (Siendo estos los que controlan D&D). Se podrían elegir distintos modelos y podrías descargar el tuyo como imagen o pdf u otro tipo de archivo.

Estas hojas luego de haber sido convertidas a html quizá mediante una página web, estas podrían ser subidas a la nube, o compartidas con los demás usuarios.

También este proyecto, tiene el objetivo de juntar dos de mis pasiones, Juegos de Rol y Programación.

Consideraciones Adicionales:

No se previó que se tuviera que cambiar las producciones de un momento a otro pero se entendió que solo se estaba haciendo una skin a C y no realmente algo propio.
Otras consideraciones, fue la simplificación de algunos campos/nodos creados, ya que solo se dispusieron de 3 terminales que se usaron sin

muchas restricciones que pudieran haber sido puestas para mejorar el resultado final.
El mundo de los TTRPG's(TableTop Role Playing Games) es muy complejo y grande para hacerle debida justicia con este proyecto ya se solo centrándonos en D&D tenemos cambios masivos entre ediciones en las que cambia el combate, stats, clases y se renuevan, agregan o quitan distintos aspectos que cambian la dinámica del juego.

Desarrollo:

Bison: Primeramente, se creó una producción en bison bastante ambiciosa y compleja la cual se entregó en la parte de frontend hace meses, en la cual se podían crear funciones, imprimir en pantalla y chequear stats, requerimientos de ítems y equipamiento, crear una party entre Characters, hasta poder simular el uso de los dados de D&D; (Anexo 1)
Luego se vio que era demasiado complejo y ambicioso para una sola persona con el tiempo que quedaba así que, se cambiaron las reglas de producción a algo más conciso y sencillo. Se generó un nodo "padre" llamado Sheet, la cual podía ser de un Personaje (Character Sheet), de un Item (Item Sheet), de un Monstruo (Monster Sheet), o de un NPC (NPC Sheet), la más compleja y en la que se centró más fue en la Character. Esta tenía de todo tipo de datos desde nombre, clase, nivel, estadísticas, "inventario"(Items). Mientras que el resto era más básico, solo con algún que otro campo como Nombre, Descripción, Stats (para Monster),etc.

Flex: En Flex al comienzo se tenían muchas entradas ya que se tenían que utilizar para las funciones propias que iban a estar disponibles entre otras. Cuando se produjo el cambio estas entradas se eliminaron y se cambiaron por otras como los nombres al comienzo de cada campo que debe de poner para ingresar la información.
También sobrevivieron algunas reglas para usar como tipos de datos como:

```
digit           [0-9]
endline         [\n]
whitespace      [ \f\r\t\v]
alphaval        [\-\+]?[0-9A-Za-z]+
cadena          [^"']*
comillas        ["]
names           [a-zA-Z][a-zA-Z0-9_\-]*
dicedamage      [0-9]+[d][0-9]+([\+\-][0-9]+)?
```

Las cuales se usan de la siguiente manera:

```
{digit}+                                    { return
IntegerPatternAction(yytext, yyleng); }

{dicedamage}                                {return DiceDamage(yytext,
yyleng);}

{comillas}                                  {BEGIN(STRING);}
<STRING>{cadena}                            {return
StringPatternAction(yytext, yyleng);}
<STRING>{comillas}                          {BEGIN(INITIAL);}

{names}                                     { return Cadena(yytext); }

{alphaval}                                  { return
AlphavalPatternAction(yytext, yyleng); }

{whitespace}+                               {
IgnoredPatternAction(yytext, yyleng); }

{endline}                                   {return
EndlineAction(yytext); }
```

Backend: En lo respectivo al backend, hay dos principales temas el ASL y el GenerateCode, ya que la tabla de símbolos no se llegó a hacer.

ASL: En el ASL, se utilizaron muchos nodos simples los cuales solo iban a terminales como:

```
typedef struct{
    objecttext * splname;
    objecttext * spldescript;
    DiceDmg * spldamage;
}Spell;
```

Aunque estos nodos simples iban a parar a nodos que juntaban estos pequeños nodos en uno más grande, los mejores ejemplos son las Sheets en especial la Character Sheet.

```
typedef struct{
    nameCharac * name;
    levelCharac * level;
    classCharac * classC;
    backgroundCharac * background;
    playername * playername;
    characrace * race;
    alignmentCharac * alignment;
    exppointsCharac * exppoints;
    restofbodyCharac * restofbody;
}Body;

typedef struct{
    statsSpread * stats;
    ArmorClass * armorclass;
    Initiative * initiative;
    Speed * speed;
    proficienciesCh * proficiencies;
    featuresCh * features;
    equipmentCh * equipment;
    itemsCh * items;
    CharacBackStory * backstory;
    Spellbook * spellbook;
}restofbodyCharac;
```

Por ultimo, tendriamos a los nodos que se llaman asi mismos, estos cumplen la funcion de campos que tendrian varios elementos del mismo tipo como: Store, Equipment, Inventory(Items), etc…

```
typedef enum{
    RECURSIVETYPE,
    NORECURTYPE
}IsRecursive;

struct equipmentCh{
    IsRecursive type;
    objecttext * equipmentName;
    objecttext * equipmentDescription;
    objecttext * partofbody;
    equipmentCh * profCh;
};
```

Flex Backend: Para conectar Flex con el backend se necesitaba que se cambiaran 3 funciones que reconocían a terminales para que pasaran el valor por yylval al bison y por ende al generator code.

String: “Cadena” con Cadena siendo cualquier caracter/es en cualquier combinación, siempre que esten entre comillas. Un error que paso yse resolvió gracias al profesor fue que debido a que no se le hacia calloc al aux3->text ocurría una segmentation fault.

```
token StringPatternAction(const char * lexeme, const int length) {
    LogDebug("StringPatternAction: '%s'", lexeme);
    objecttext * aux3 = malloc(sizeof(objecttext));
    aux3->text = calloc(length + 1, 1);
    strncpy(aux3->text, lexeme, length);
    aux3->len = length;
    yylval.text = aux3;
    //yylval.token = STR;
    return STR;
}
```

Integer: Cualquier número. Se guarda a la string en un parcial auxiliar al que se le hace calloc para luego que se le haga atoi() a ese string para que se convierta en Integer.

```
token IntegerPatternAction(const char * lexeme, const int length) {
```

```
    LogDebug("IntegerPatternAction: '%s' (length = %d).", lexeme,
length);
    // Reservar memoria para el lexema identificado y el \0 final:
    char * text = (char *) calloc(length + 1, sizeof(char));
    // Copiar el lexema y \0 para evitar segmentation-faults:
    strncpy(text, lexeme, length);
    Constant * aux = malloc(sizeof(Constant));
    // Convertir el lexema en un entero de verdad:
    aux->value = atoi(text);
    yylval.value = aux;
    // Liberar la memoria, ya que solo nos interesa el resultado de
atoi(.)
    // (no debería llamar a free(.), si "text" debe ser utilizado en
Bison):
    free(text);

    //yylval.token = INTEGER;
    return INTEGER;
```

Dice damage: El cual tiene 3 partes la cantidad de dados, el tipo de dado y un modificador al daño que es +- un número usualmente debajo de 10. Se usa strok para separar esa string por d y el +- dejando a cada pedazo para hacerse atoi(); para pasar a integer y ser guardado en el tipo de dato. Los while se usan para contar los dígitos de cada uno de los datos y además saber si el modificador es positivo o negativo.

```
token DiceDamage(const char * lexeme, const int length) {
    LogDebug("DiceDamage: '%s'", lexeme);
    // Reservar memoria para el lexema identificado y el \0 final:
    char * text = (char *) calloc(length + 1, sizeof(char));
    // Copiar el lexema y \0 para evitar segmentation-faults:
    strncpy(text, lexeme, length);

    int i = 0;
    int cambio = 1;
    while (text[i] != 'd'){
        i++;
    }
    int j = i + 1;
    while (text[j] != 0 || text[j] != '+' || text[j] != '-'){
        j++;
    }
    if (text[j] == '+' || text[j] != '-'){
```

```
        if(text[j] == '-')
            cambio = -1;
        int x = j + 1;
        while (text[x] != 0){
            x++;
        }
        x = x - j - 1;
    }
    j = j - i - 1;

    char delimitadores[] = "+-d ";
    char * tok;

    DiceDmg * aux2 = malloc(sizeof(DiceDmg));

    tok = strtok(text, delimitadores);
    if (tok != NULL)
        aux2->cantDice = atoi(tok);
    tok = strtok(NULL, delimitadores);
    if(tok != NULL)
        aux2->typeofdice = atoi(tok);
    tok = strtok(NULL, delimitadores);
    if(tok != NULL)
        aux2->modif = cambio * atoi(tok);

    yylval.damage = aux2;
    // Liberar la memoria, ya que solo nos interesa el resultado de
atoi(.)
    // (no debería llamar a free(.), si "text" debe ser utilizado en
Bison):
    free(text);
    //yylval.token = DICEDMG;
    return DICEDMG;
}
```

Bison Backend: En bison para unirlo con el backend se utilizan las funciones en las producciones que toman los valores que estan yylval y otros nodos "hijos" y generan una instancia de ellos mismos, se reservan su tamaño con malloc(); y guardan la informacion.
Estos incluyen algunos ejemplos:

```
proficienciesCh * proficienciesChFunction(IsRecursive type, objecttext
* arg1, objecttext * arg2, Constant * arg3, proficienciesCh *
proficienciesC){
    proficienciesCh * t = malloc(sizeof(proficienciesCh));
    t->profDescription = arg2;
    t->nameProf = arg1;
    t->valueProf = arg3;
    if(type == RECURSIVETYPE){
        t->type = RECURSIVETYPE;
        t->profCh = proficienciesC;
    }
    else{
        t->type = NORECURTYPE;
        t->profCh = NULL;
    }
    return t;
}
```

En este caso, al este nodo ser recursivo y tener varias reglas de producción tenemos que usar o un if else o switch/case para separar casos.

GenerateCode: Se creo un file html con fopen al cual se le escribe con fprintf, el cual toma el file y una linea a escribir y la pone reemplazando lo necesario. Ejemplos de partes de codigo son:

```
void generatesheet(Sheet * sh){
    fprintf(f,"<head>\n");
    fprintf(f, "<style>\n");
    fprintf(f, "p{ \n");
    fprintf(f, "color: navy;\n");
    fprintf(f, "text-align:center;\n");
    fprintf(f, "} \n");
    fprintf(f, "</style>\n");
    fprintf(f, "<title>");

    switch (sh->type){
        case CHARACTERSH:
            fprintf(f, "Your Character Sheet with id = %d </title>\n",
sh->id->value);
            fprintf(f, "</head>\n");
            fprintf(f, "<body>\n");
            generatechbody(sh->body);
            break;
```

```
        case MOSNTERSH:
            fprintf(f, "Your Monster Sheet with id = %d </title>\n",
sh->id->value);
            fprintf(f, "</head>\n");
            fprintf(f, "<body>\n");
            generatemonsbody(sh->mbody);
            break;

        case ITEMSH:
            fprintf(f, "Your Item Sheet with id = %d </title>\n",
sh->id->value);
            fprintf(f, "</head>\n");
            fprintf(f, "<body>\n");
            generateitembody(sh->itembody);
            break;

        case NPCSH:
            fprintf(f, "Your Merchant Sheet with id = %d </title>\n",
sh->id->value);
            fprintf(f, "</head>\n");
            fprintf(f, "<body>\n");
            generatemerchantbody(sh->npcbody);
            break;

        default:
            LogError("Error en el tipo de Sheet");
            break;
    }
}
```

Que lo que hace es separar las 4 Sheets a distintos generates con el switch/case.

```
void generateprof(proficienciesCh * pr){
        proficienciesCh * aux = malloc(sizeof(proficienciesCh));
        aux = pr;
        while (aux->type != NORECURTYPE){
            fprintf(f, "<p> %s: %s, %d </p>\n", aux->nameProf->text,
aux->profDescription->text, aux->valueProf->value);
            aux = aux->profCh;
        }
```

```
        fprintf(f, "<p> %s: %s, %d </p>\n", aux->nameProf->text,
aux->profDescription->text, aux->valueProf->value);
    }
```

En el caso de las funciones recursivas creó un auxiliar con el cual con un while voy imprimiendo cada una de las entradas y pasando a la siguiente hasta llegar a la última.

Dificultades:

Las mayores dificultades fueron principalmente el manejo del tiempo debido a que solo soy un integrante y también se estaba en la época de exámenes y entregas de otros TP 's.
Dificultades técnicas hubo también ya que cuando quería ver cuales eran los conflictos S/R que tenía mi sistema la terminal no me dejaba usar la flag correspondiente -Wcounterexamples y tiraba error.
Me tuve que cambiar a una VM o cosa parecida para arreglar bien el problema.
Otra dificultad para mi fue la de entender como yylval pasa los datos de Flex a Bison y como Bison pasa los datos de nodo en nodo, hasta que el profesor me pudo aclarar el tema que me llevo comiendo la cabeza bastante tiempo.

Futuras Modificaciones:

- Embellecer el html con CSS u otro tipo de lenguaje de estilo.
- Agregar Posibilidad de no solo ser para D&D e incluir otros TTRPG’s
- Incluir más Información/Nodos/Campos.
- Poner más escrutinio en el tipo de datos. Ejemplo: Que solo ciertas strings puedan pasar en alignment como “Chaotic Evil”, “True Neutral”, etc.
- Ver la posibilidad si se puede poner la información directamente en un pdf u otro tipo de dato.
- Crear un backend en la página html que permite que compartas tu Sheet con otras personas y viceversa.

Anexo 1: Producciones anteriores que no se reciclaron.

```
aux: startprograma                    {$$ = StartProgramAction($1);}
    | crearfunct freeendlines aux    {$$ =
CrearnuevafunctionAction($1);}
    ;

crearfunct: FUNCT truedata CADENA OPEN_PARENTHESIS
argumentosparadeclarar CLOSE_PARENTHESIS OPEN_LLAVES freeendlines
programa freeendlines CLOSE_LLAVES  {$$ = b("crearfunct     ");}
            ;

truedata: datatype                   {$$ = b("truedatadatatype     ");}
        | complexdatatype            {$$ = b("truedatacomplexdata
");}
        ;


complexdatatype : STAT               {$$ = b("STAT     ");}
                | STATS              {$$ = b("STATS     ");}
                | ITEM               {$$ = b("ITEM     ");}
                | MOSNTER            {$$ = b("MOSNTER     ");}
                | MODIF              {$$ = b("MODIF     ");}
                | CHARAC             {$$ = b("CHARAC     ");}
                | PARTY              {$$ = b("PARTY     ");}
                | NPC                {$$ = b("NPC     ");}
                | RAZGO              {$$ = b("RAZGO     ");}
                | SHEET              {$$ = b("SHEET     ");}
                ;

argumentosparadeclarar: truedata CADENA COMMA argumentosparadeclarar
{$$ = b("argumetosparadeclararrec  ");}
                        | truedata CADENA
{$$ = b("argumetoultimoparadeclarar   ");}
                        ;

startprograma: INTDT START OPEN_PARENTHESIS CLOSE_PARENTHESIS
OPEN_LLAVES freeendlines trueprogram RET OPEN_PARENTHESIS INTEGER
CLOSE_PARENTHESIS PUNTOCOMA freeendlines CLOSE_LLAVES       {$$ =
b("startPrograma   ");}
                | INTDT START OPEN_PARENTHESIS CLOSE_PARENTHESIS
OPEN_LLAVES freeendlines RET OPEN_PARENTHESIS INTEGER CLOSE_PARENTHESIS
```

```
PUNTOCOMA freeendlines CLOSE_LLAVES                {$$ =
b("startProgramavacio   ");}
               ;

programa: trueprogram RET OPEN_PARENTHESIS CADENA CLOSE_PARENTHESIS
PUNTOCOMA       {$$ = b("programaretornocadena    ");}
       | trueprogram RET OPEN_PARENTHESIS valorrel CLOSE_PARENTHESIS
PUNTOCOMA     {$$ = b("programaretornovalor    ");}
       | RET OPEN_PARENTHESIS valorrel CLOSE_PARENTHESIS PUNTOCOMA
{$$ = b("programasintrueprogram    ");}
       ;

programacond: trueprogram
{$$ = b("programacondsinret    ");}
           | trueprogram RET OPEN_PARENTHESIS CADENA CLOSE_PARENTHESIS
PUNTOCOMA freeendlines      {$$ = b("programacondconretcadea    ");}
           | trueprogram RET OPEN_PARENTHESIS valorrel
CLOSE_PARENTHESIS PUNTOCOMA freeendlines                {$$ =
b("programacondconretvalor    ");}
           | RET OPEN_PARENTHESIS valorrel CLOSE_PARENTHESIS PUNTOCOMA
freeendlines                                {$$ =
b("programacondconsoloret    ");}
           ;

trueprogram: variableoper freeendlines                                {$$
=  b("variableoper  ");}
           |variableoper freeendlines trueprogram                     {$$
= b("variableoper truep  ");}
           | returnfunction PUNTOCOMA freeendlines trueprogram        {$$
= b("returnfunction truep  ");}
           | aconditional freeendlines trueprogram                    {$$
= b("aconditional truep  ");}
           | returnfunction PUNTOCOMA freeendlines                    {$$
= b("returnfunction  ");}
           | aconditional freeendlines                                {$$
= b("aconditional  ");}
           ;

variableoper: avariable         {$$ =  b("variableopertrue  ");}
           ;

avariable: declararvar              {$$ =  b("declararvar  ");}
       | complexvar                 {$$ = b("complexvar  ");}
```

```
        | complexch                 {$$ = b("complexch  ");}
        | declarearray              {$$ = b("declarearray  ");}
        | declarearraycmpx          {$$ = b("declarearraycmpx  ");}
        | asignsimplearr            {$$ = b("assignsimplearray  ");}
        | asigncpxarr               {$$ = b("assigncomplexarray  ");}
        | complexdeclar             {$$ = b("complexdeclar  ");}
        ;

declararvar: datatype CADENA IGUAL valorvar PUNTOCOMA   {$$ =
b("declararvartotal  ");}
           | datatype CADENA PUNTOCOMA                  {$$ =
b("declararvarsimple  ");}
           ;

valorvar: subvalorvar                                   {$$ =
b("subvalorvar  ");}
        | simplevalues operation valorvar               {$$ =
b("opsimplevalues  ");}
        | returnfunction operation valorvar             {$$ =
b("retfunoper  ");}
        | OPEN_PARENTHESIS valorvar CLOSE_PARENTHESIS   {$$ =
b("valorvarparentesis  ");}
        | CADENA operation valorvar                     {$$ =
b("cadenaoperacionvalorvar  ");}
        ;

subvalorvar: simplevalues       {$$ = b("simplevaluessubvalorar  ");}
           | returnfunction     {$$ = b("returnfunctionsub  ");}
           | CADENA             {$$ = b("cadenasub  ");}
           | DICEDMG            {$$ = b("dicedmgsub  ");}
           | ALPHAVAL           {$$ = b("alphavalsub  ");}
           ;

simplevalues: INTEGER               {$$ = b("integer  ");}
           | STR                    {$$ = b("string  ");}
           ;

arrayvalues: INTEGER                {$$ = b("integerforarrays  ");}
           | CADENA                 {$$ = b("cadenaforarrays  ");}
           ;

datatype: INTDT                     {$$ = b("IntegerType  ");}
        | STRDT                     {$$ = b("StringType  ");}
```

```
        ;
*/

/*
operation: ADD              {$$ = b("suma  ");}
        | SUB               {$$ = b("resta  ");}
        | MUL               {$$ = b("multiplicacion   ");}
        | DIV               {$$ = b("division   ");}
        ;

asignarvar: CADENA IGUAL valorvar PUNTOCOMA     {$$ =
b("aignarvarcadenavalor  ");}
        ;

checkearvar: valorrel operrel valorrel      {$$ = b("checkarvar   ");}
           | CADENA operrel CADENA          {$$ = b("checkarvar2   ");}
           | CADENA operrel valorrel        {$$ = b("checkarvar3   ");}
           ;

valorrel: simplevalues                                      {$$ =
b("simplevaluesvalorrel  ");}
        | returnfunction                                    {$$ =
b("returnfuncvalorrel  ");}
        | OPEN_PARENTHESIS asignarvar CLOSE_PARENTHESIS     {$$ =
b("parentesisvalorrel  ");}
        ;

operrel: GREATER        {$$ = b("greater  ");}
        | LESSER        {$$ = b("lesser  ");}
        | GREATorEQ     {$$ = b("greateroreq  ");}
        | LESSorEQ      {$$ = b("lesseroreq   ");}
        | EQUA          {$$ = b("equals  ");}
        | NOTEQ         {$$ = b("noteq  ");}
        ;

complexvar: complexdatatype CADENA IGUAL NEW complexdatatype
OPEN_PARENTHESIS argumentos CLOSE_PARENTHESIS PUNTOCOMA    {$$ =
b("complexvariniti  ");}
        | complexdatatype CADENA IGUAL NEW complexdatatype
OPEN_PARENTHESIS CLOSE_PARENTHESIS PUNTOCOMA                {$$ =
b("complexvarinitisinarg  ");}
        | complexdatatype CADENA PUNTOCOMA
{$$ = b("complexvarsininit  ");}
```

```
        ;

complexdeclar: CADENA IGUAL NEW complexdatatype OPEN_PARENTHESIS
argumentos CLOSE_PARENTHESIS PUNTOCOMA     {$$ = b("complexvariniti
");}
            | CADENA IGUAL NEW complexdatatype OPEN_PARENTHESIS
CLOSE_PARENTHESIS PUNTOCOMA                 {$$ =
b("complexvardeclarsinarg  ");}
                ;

complexch: CADENA POINT complexch
{$$ = b("complexch2  ");}
        | asignarvar
{$$ = b("complexch3  ");}
        ;

declarearray: datatype CADENA OPEN_CORCHETES CLOSE_CORCHETES IGUAL NEW
datatype OPEN_CORCHETES INTEGER CLOSE_CORCHETES PUNTOCOMA        {$$ =
b("declarearray  ");}
                ;

declarearraycmpx: complexdatatype CADENA OPEN_CORCHETES CLOSE_CORCHETES
IGUAL NEW complexdatatype OPEN_CORCHETES INTEGER CLOSE_CORCHETES
PUNTOCOMA      {$$ = b("declarearraycmpx  ");}
                ;

asignsimplearr: CADENA OPEN_CORCHETES arrayvalues CLOSE_CORCHETES IGUAL
valorvar PUNTOCOMA      {$$ = b("assignsimplarrvalor  ");}
            ;

asigncpxarr: CADENA OPEN_CORCHETES arrayvalues CLOSE_CORCHETES IGUAL
NEW complexdatatype OPEN_PARENTHESIS argumentos CLOSE_PARENTHESIS
PUNTOCOMA                {$$ = b("assigncmparr1  ");}
            | CADENA OPEN_CORCHETES arrayvalues CLOSE_CORCHETES IGUAL
NEW complexdatatype OPEN_PARENTHESIS CLOSE_PARENTHESIS PUNTOCOMA
{$$ = b("assigncmparrsinarg  ");}
            | CADENA OPEN_CORCHETES arrayvalues CLOSE_CORCHETES POINT
complexch
{$$ = b("assigncmparr3  ");}
            | CADENA OPEN_CORCHETES arrayvalues CLOSE_CORCHETES POINT
complexdeclar
{$$ = b("assigncmparr2  ");}
            ;
```

```
returnfunction: CADENA OPEN_PARENTHESIS argumentos CLOSE_PARENTHESIS
{$$ = b("returnfunct3  ");}
            | functionnames OPEN_PARENTHESIS argumentos
CLOSE_PARENTHESIS                   {$$ = b("retunrfunct4  ");}
            ;

argumentos: valorvar                          {$$ = b("argvalor1  ");}
        | valorvar COMMA argumentos     {$$ = b("argvalor2omas  ");}
        ;

functionnames: PRINT            {$$ = b("f1  ");}
            | CCHAR             {$$ = b("f2  ");}
            | CMONS             {$$ = b("f3   ");}
            | CRACE             {$$ = b("f4   ");}
            | CCLASS            {$$ = b("f5   ");}
            | CITEM             {$$ = b("f6   ");}
            | CNPC              {$$ = b("f7   ");}
            | CFEAT             {$$ = b("f8    ");}
            | CPARTY            {$$ = b("f9    ");}
            | ASTAT             {$$ = b("f10    ");}
            | ACINFO            {$$ = b("f11    ");}
            | ACBACK            {$$ = b("f12    ");}
            | ASPBK             {$$ = b("f4    ");}
            | ASTSPBK           {$$ = b("f4    ");}
            | ARMOD             {$$ = b("f4    ");}
            | ACMOD             {$$ = b("f4    ");}
            | ANPCINF           {$$ = b("f4    ");}
            | AMINF             {$$ = b("f4    ");}
            | AITDES            {$$ = b("f4    ");}
            | AITINF            {$$ = b("f4    ");}
            | APMEM             {$$ = b("f4    ");}
            | RMPMEM            {$$ = b("f4    ");}
            | CHEXP             {$$ = b("f4    ");}
            | CHCLEV            {$$ = b("f4    ");}
            | CHCLASS           {$$ = b("f4    ");}
            | CHLEV             {$$ = b("f4    ");}
            | CHRACE            {$$ = b("f4    ");}
            | BHCMULCL          {$$ = b("f4    ");}
            | CHITCLASS         {$$ = b("f4    ");}
            | CHITRAR           {$$ = b("f4    ");}
            | CHITREQ           {$$ = b("f4    ");}
            | CHNPCCLASS        {$$ = b("f4    ");}
```

```
            | EQITEM            {$$ = b("f4    ");}
            | UEQIT             {$$ = b("f4    ");}
            | CHKCLASS          {$$ = b("f4     ");}
            | CHKLEVL           {$$ = b("f4     ");}
            | CHKEXP            {$$ = b("f4     ");}
            | CHKSPLS           {$$ = b("f4     ");}
            | CHKITEM           {$$ = b("f4     ");}
            | CHKRACE           {$$ = b("f4     ");}
            | CHKITREST         {$$ = b("f4     ");}
            | CHKFEAT           {$$ = b("f4     ");}
            | CHKMONSINF        {$$ = b("f4    ");}
            | CHKPARTY          {$$ = b("f4    ");}
            | EXPSH             {$$ = b("f4   ");}
            | GETSH             {$$ = b("f4   ");}
            | ACTSH             {$$ = b("f4   ");}
            | CHTPSH            {$$ = b("f4   ");}
            | CHKSTATS          {$$ = b("f4   ");}
            | CHKSTAT           {$$ = b("f4   ");}
            ;

aconditional: ifelse        {$$ = b("ifelse  ");}
            | dowhile       {$$ = b("dowhile  ");}
            | foriter       {$$ = b("for  ");}
            ;

ifelse: IFCOND OPEN_PARENTHESIS condition CLOSE_PARENTHESIS OPEN_LLAVES
freeendlines programacond CLOSE_LLAVES elseiter     {$$ =
b("ifelsepadreconhijo  ");}
        |IFCOND OPEN_PARENTHESIS condition CLOSE_PARENTHESIS
OPEN_LLAVES freeendlines programacond CLOSE_LLAVES                {$$
= b("ifelsepadre  ");}
        ;

elseiter: ELSECOND OPEN_LLAVES freeendlines programacond CLOSE_LLAVES
{$$ = b("elseitersincond  ");}
        | ELSIFCOND OPEN_PARENTHESIS condition CLOSE_PARENTHESIS
OPEN_LLAVES freeendlines programacond CLOSE_LLAVES elseiter
{$$ = b("elseiterconcond  ");}
        ;

condition: checkearvar AND condition                              {$$
= b("condition and  ");}
```

```
        | checkearvar OR condition                                    {$$
= b("condition or  ");}
        | checkearvar                                                 {$$
= b("condition check  ");}
        | OPEN_PARENTHESIS condition CLOSE_PARENTHESIS                {$$
= b("condition parentesis  ");}
        ;

dowhile: DOCOND OPEN_LLAVES freeendlines programacond  CLOSE_LLAVES
WHILECOND OPEN_PARENTHESIS condition CLOSE_PARENTHESIS PUNTOCOMA
{$$ = b("dowhile1  ");}
        | WHILECOND OPEN_PARENTHESIS condition CLOSE_PARENTHESIS
OPEN_LLAVES freeendlines programacond  CLOSE_LLAVES
{$$ = b("while  ");}
        ;

foriter: FORCOND OPEN_PARENTHESIS argfor1 PUNTOCOMA condition PUNTOCOMA
argfor3 CLOSE_PARENTHESIS OPEN_LLAVES freeendlines programacond
CLOSE_LLAVES    {$$ = b("for1  ");}
        | FORCOND OPEN_PARENTHESIS PUNTOCOMA condition PUNTOCOMA
CLOSE_PARENTHESIS OPEN_LLAVES freeendlines programacond CLOSE_LLAVES
{$$ = b("for2  ");}
        | FORCOND OPEN_PARENTHESIS argfor1 PUNTOCOMA condition
PUNTOCOMA CLOSE_PARENTHESIS OPEN_LLAVES freeendlines programacond
CLOSE_LLAVES           {$$ = b("for3  ");}
        | FORCOND OPEN_PARENTHESIS PUNTOCOMA condition PUNTOCOMA
argfor3 CLOSE_PARENTHESIS OPEN_LLAVES freeendlines programacond
CLOSE_LLAVES           {$$ = b("for4  ");}
        ;

argfor1: asignarvar                                        {$$ =
b("argfor1asignvar  ");}
        | datatype CADENA IGUAL valorvar                   {$$ =
b("argfor1valor  ");}
        ;

argfor3: asignarvar                                                {$$ =
b("argfor3asignavar  ");}
        | simplevalues operation valorvar                          {$$ =
b("argfor3simplevalues  ");}
        | returnfunction operation valorvar                        {$$ =
b("argfor3returnfunction  ");}
```

```
        | OPEN_PARENTHESIS valorvar CLOSE_PARENTHESIS           {$$ =
b("argfor3parentesis      ");}
        | CADENA IGUAL valorvar                                {$$ =
b("argfor3default  ");}
        ;
```