Tshwane University of Technology
We empower people

INSTRUCTIONS TO CANDIDATES

1. All exam rules stated by the Tshwane University of Technology apply.
2. **Ensure a single final version of your source code is handed in as requested.**
3. If needed, state all necessary assumptions clearly in code commentary.

MARKS: 100%

PAGES: 15 (incl. cover)

EXAMINER:

Mr A.J. Smith

Prof J.A. Jordaan

MODERATOR:

Mr D Engelbrecht

TIME:

120 Minutes

FACULTY OF ENGINEERING AND THE BUILT ENVIRONMENT

DEPARTMENT OF ELECTRICAL ENGINEERING

ES216BB
ENGINEERING SOFTWARE DESIGN B

EVALUATION 2

OCTOBER 2024

EVALUATION INSTRUCTIONS

1. **Plagiarism:** Submit only original work. We will use similarity software to verify the authenticity of all submissions.

2. **Permitted Tools:** You are allowed to use only **CodeBlocks,** and **Google Chrome** to access the evaluation, view the evaluation PDF and upload submission for this evaluation. Access to emails, other online resources, and memory sticks is strictly prohibited. Please be aware that computer activity will be remotely monitored. Breaches of TUT's official examination and module rules will result in a minimum penalty of zero for this evaluation, with the potential for further disciplinary action.

3. **File Submission:** Your source code file must be named according to this format: **"<student number>.cpp" (e.g. 21011022.cpp).** Do not add any other text (name, surname, etc.) to the file name (ONLY YOUR STUDENT NUMBER).

4. **Uploading Instructions:** Submit your "**.cpp**" file via the designated upload link. While multiple uploads are allowed, only the most recent submission will be retained on the system. If you make an error in your initial upload, simply re-upload your file, and the previous version will be overridden. Download your submission to confirm that the correct file has been uploaded.

5. **Evaluation Scope:** This assessment encompasses basic content from ES216AB and specifically ES216BB content defined in **Unit1 to Unit4**

6. **Programming Language:** Construct your program in **C++** and adhere to structured programming principles.

7. **Editing and Requirements:** Your program must meet all specified requirements. Refer to the attached appendices for additional details.

8. **Evaluation Requirements:**

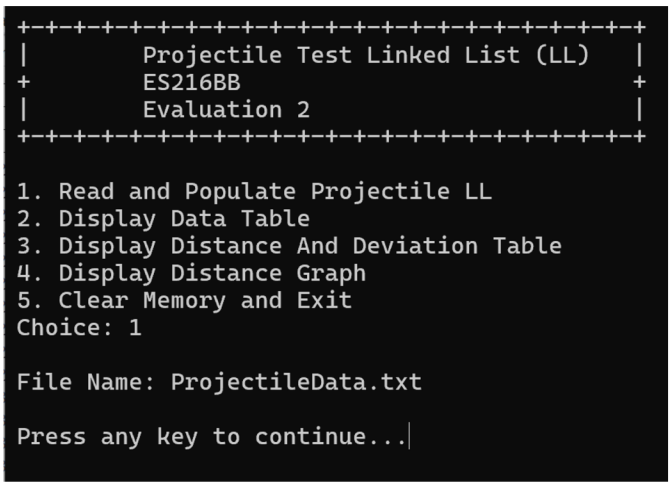
    a. Remember to save your work on the PC "D: Drive" and save regularly throughout the evaluation.
    b. Do not modify the given code in the "**.cpp**" file except for implementing the requested functions as required.
    c. Use the exact function names and parameters as used in the in the "**.cpp**" file function prototypes and main function.
    d. Complete the C++ functions below the main function in each comment block as shown.

C++ FILE CODE EXPLANATION

You will be provided with a **C++** file, which contains a partially completed program that manages a linked list structure to store projectile test data. Your task is to implement the missing functions as described below.

The main function sets up a menu system that allows the user to perform the following tasks:

- Read data from a text file (**ProjectileData.txt**) and populate the linked list.
- Display the linked list data in a tabular format.
- Display the projectile distances along with deviations from the average distance.
- Display a simple distance graph using stars (*), where each star represents 10 meters.
- Delete all nodes in the linked list and exit the program.



The provided code contains the **pNode** structure to store projectile data, including **test name**, **weight**, **speed**, and **offset**. The linked list is dynamically managed, and memory is allocated or deallocated as needed. You will be required to implement the missing functions as detailed below.

FUNCTIONS

The required functions for the C++ program should be implemented in the appropriate comment blocks as given in the **.cpp** file. The function declarations and descriptions are as follows:

1. Populate Node Function

void pNode::PopulateNode (string pT, float pW, float pS, float pO);

The **PopulateNode** function is a member of the **pNode** structure. It is used to initialise the **pTest**, **pWeight**, **pSpeed**, and **pOffset** members of the **pNode** structure with the provided parameters. The function receives four parameters: **pT**, **pW**, **pS**, and **pO**, which are used to set the **pTest**, **pWeight**, **pSpeed**, and **pOffset** members of the **pNode** structure, respectively. This function populates a node structure element with projectile test data.

- **pT**: The test name (**string**) represents the name of the projectile test.
- **pW**: The projectile weight (**float**) in kilograms.
- **pS**: The initial speed of the projectile (**float**) in meters per second.
- **pO**: The offset value (**float**) which compensates for specific design characteristics of the projectile.

This function allows the node to store all relevant information regarding a projectile test, which will be used later in calculations and data display.

2. Projectile Distance Function

float pNode::ProjectileDistance (void);

The **ProjectileDistance** function is a member of the **pNode** structure. It calculates the distance that a projectile will travel based on the projectile's weight, initial speed, and offset while also accounting for an approximation of the Earth's curvature. The function does not take any parameters, as it uses the **pWeight**, **pSpeed**, and **pOffset** member variables of the **pNode** structure to perform the calculation. It returns a **float** value representing the calculated distance.

Function Details:

- The first step in the function is to calculate the curvature compensation factor, which adjusts the projectile distance based on the projectile's weight and the Earth's curvature.

$$CurvatureCompensation = 1.0 + \left(\frac{pWeight}{EARTH\_RADIUS \times 100.0}\right)$$

  This formula introduces a slight adjustment to the distance based on the weight of the projectile, which accounts for the Earth's curvature in a simplified manner. The Earth's radius is a constant, 6,371,000 meters, and the weight of the projectile is divided by the Earth's radius multiplied by 100.0 to scale the compensation.

- The formula used to calculate the distance is derived from basic kinematic equations for projectile motion. Assuming a 45-degree launch angle (which maximises the range) and no air resistance, the formula simplifies to:

$$Range = \left(\frac{pSpeed \times pSpeed}{GRAVITY}\right) \times CurvatureCompensation$$

- After calculating the range, the function applies an offset to the result, which compensates for specific design characteristics of the projectile. This is done by multiplying the calculated range by the pOffset value.

Return Value:

- The function returns a float that represents the final calculated distance of the projectile, factoring in the initial speed, weight, Earth's curvature, and offset.

3. Read File and Populate Function

void ReadFileAndPopulate (string FileName, pNode **sPtr);

The **ReadFileAndPopulate** function is responsible for reading projectile test data from a text file and populating a linked list of **pNode** structures. It receives two parameters: a **string FileName**, which represents the name of the file to be read, and a double pointer **sPtr**, which points to the start of the linked list. The function ensures that the linked list is populated with projectile test data from the file, where each test is stored as a node in the list.

Function Details:

- The function begins by opening the file specified by FileName using the **open()** function of the input file stream object (**ifstream**).
- Before populating the list, the function checks if there is existing data in the linked list. If the list is not empty (***sPtr != nullptr** or ***sPtr != NULL**), it calls **DeleteAllNodes(sPtr)** to clear the existing nodes from memory and avoid any potential memory leaks.
- After clearing the memory, the function opens the file and reads its contents in a loop, checking until the end of the file (**!File.eof()**).
- For each line in the file, the function reads the test name, weight, speed, and offset into corresponding local variables (Test, Weight, Speed, Offset).
- Once the data is extracted from the file, the function calls **InsertNode**(**sPtr**, **Test**, **Weight**, **Speed**, **Offset**) to create a new node and insert it at the end of the linked list.
- After all the data has been processed and inserted into the linked list, the file is closed using **close()** function of the input file stream object (**ifstream**).

Text File Content (ProjectileData.txt):

```
A01_Test        155.3   43.1    1.123
B29_Test        101.0   59.8    1.432
A02_Test        125.9   47.2    1.111
B30_Test        111.1   51.3    1.025
B31_Test        132.6   32.7    1.212
A03_Test        123.4   44.3    1.101
```

Key Operations:

1. **Memory Management:** The function ensures that any existing linked list is cleared before new data is read and added.
2. **File Reading:** The function reads data sequentially from the file, with each line containing information about a projectile test.
3. **Node Insertion:** The function calls InsertNode() to create and insert each new node into the linked list.

4. Insert Node Function – insert at the back of the linked list

void InsertNode(pNode **sPtr, string Test, float Weight, float Speed, float Offset);

The **InsertNode** function is used to create a new node (**pNode**) and **insert it at the back of a linked list**. It receives four parameters: **Test**, **Weight**, **Speed**, and **Offset**, which represent the projectile test details. The function dynamically allocates memory for the new node, populates the node with the given data, and links it to the existing linked list.

Parameters:

- **sPtr**: A pointer to the pointer of the start node of the linked list (pNode**). This parameter allows the function to modify the head of the list if necessary.
- **Test**: A string representing the test name of the projectile.
- **Weight**: A float representing the weight of the projectile in kilograms.
- **Speed**: A float representing the initial speed of the projectile in meters per second.
- **Offset**: A float representing the design-specific offset for the projectile.

Function Details:

1. The function starts by dynamically allocating memory for a new node using new pNode.
    - If memory allocation fails (i.e., newNode is nullptr/NULL), the function simply returns without making any changes to the linked list.
2. If memory allocation is successful, the function populates the new node using the **PopulateNode()** structure member function. This sets the new node's pTest, pWeight, pSpeed, and pOffset values with the corresponding parameters.
3. The new node's nextPtr is set to nullptr/NULL, as it will be inserted at the end of the list and will have no subsequent nodes.
4. If the linked list is currently empty (*sPtr is nullptr/NULL), the new node becomes the first node in the list, and *sPtr is updated to point to the new node.
5. If the linked list is not empty, the function traverses the list using a pointer (currentPtr) to find the last node. It loops through the list until it finds the node where nextPtr is nullptr/NULL.
6. Once the end of the list is reached, the new node is inserted by setting the last node's nextPtr to point to the new node.

5. Display Data Table Function

void DisplayDataTable(pNode *sPtr);

The **DisplayDataTable** function is responsible for displaying the data stored in each node of the linked list in a well-formatted table. The function receives one parameter, **sPtr**, which is a pointer to the start of the linked list (**pNode ***). It iterates through the linked list, printing the test name, projectile weight, initial speed, and offset value for each node in a neatly formatted table.

Function Details:

- The function begins by creating a pointer variable currentPtr that is initialised to point to the head of the linked list (sPtr). This pointer will be used to traverse the linked list.
- The function then outputs a header row to label the columns of the table. These columns represent the projectile test data, including Test, Weight, Speed, and Offset.
- Using the setw() manipulator from the <iomanip> library, each value is printed with a width of 20 characters to ensure proper alignment of the columns. The left manipulator ensures that the text is left-aligned within each column.
- Inside a while loop, the function traverses through the linked list starting from currentPtr and continues until the end of the list (i.e., when currentPtr becomes nullptr/NULL).
- For each node in the list, the function prints the pTest, pWeight, pSpeed, and pOffset values, followed by a new line. After printing the values of the current node, currentPtr is updated to point to the next node in the list (currentPtr = currentPtr->nextPtr).
- Once all nodes have been printed, the function outputs a footer line to close the table.

Expected Output:

```
------------------------------------------------------------------
Test                Weight              Speed               Offset
------------------------------------------------------------------
A01_Test            155.3               43.1                1.123
B29_Test            101                 59.8                1.432
A02_Test            125.9               47.2                1.111
B30_Test            111.1               51.3                1.025
B31_Test            132.6               32.7                1.212
A03_Test            123.4               44.3                1.101
------------------------------------------------------------------
```

Key Points:

- The function iterates through the entire linked list, printing out the values stored in each node in a structured and readable format.
- The use of manipulators like setw() and left ensures that the output is well-formatted and aligned in columns, making the data easy to read.

6. Display Distance and Deviation Table Function

void DisplayDistanceDeviationTable(pNode *sPtr);

The **DisplayDistanceDeviationTable** function calculates and displays each projectile's distance and its deviation from the average distance. It traverses the linked list to compute the average distance and then prints the results in a formatted table.

Function Details:

- The function initialises a float variable to store the sum of all projectile distances and an int variable to track the number of nodes.
- A pointer, currentPtr, is used to traverse the linked list.
- In the first loop, the function:
  - Iterates through each node, calling **ProjectileDistance()** to calculate and accumulate the total distance.
  - Calculates the average distance by dividing the float variable (sum) by the int variable (count).
- The function prints a table header, and then, in the second loop, it:
  - Iterates again through the list, printing the test name, the projectile distance, and the deviation (distance minus average) for each node.

Expected Output:

```
------------------------------------------------------------------------
Test                Distance            Deviation
------------------------------------------------------------------------
A01_Test            212.65              -56.4
B29_Test            522.007             252.957
A02_Test            252.307             -16.7431
B30_Test            274.973             5.92276
B31_Test            132.108             -136.942
A03_Test            220.255             -48.7949
------------------------------------------------------------------------
```

Key Points:

- It calculates and displays the average distance and deviation for each projectile.
- The table is formatted using setw() for readability.
- The function is critical for showing how each projectile's distance compares to the average.

7. Display Node Distance Graph Function

void DisplayNodeDistanceGraph(pNode *sPtr);

The **DisplayNodeDistanceGraph** function visually represents the projectile distances stored in the linked list using stars (*), where each star represents 10 meters. It traverses the linked list and prints each projectile test's name alongside a star-based graph depicting the calculated distance.

Function Details:

- The function uses currentPtr to traverse the linked list, starting from sPtr.
- A table header is printed with two columns:
  - Test: The name of the projectile test.
  - 10 meter per star: A visual representation where each star corresponds to 10 meters of distance.
- In the loop:
  - The ProjectileDistance() function is called to compute the distance for each projectile.
  - The distance is divided by 10 to determine how many stars to print: pDist = (int) currentPtr->ProjectileDistance() / 10. Typecast to integer value is used to remove any comma values.
  - The function prints the test name and the corresponding number of stars (*) to visualise the distance.
- The loop continues until all nodes are processed, and a footer line is printed to close the graph.

Expected Output:

```
------------------------------------------------------------------
Test                | 10 meter per star
------------------------------------------------------------------
A01_Test            | ********************
B29_Test            | **************************************************
A02_Test            | ************************
B30_Test            | ***************************
B31_Test            | *************
A03_Test            | **********************
------------------------------------------------------------------
```

Key Points:

- The function visualises projectile distances using a simple star-based graph, with each star representing 10 meters.
- It provides a clear and easy-to-understand representation of how far each projectile travelled relative to the others.
- The graph is formatted for clarity using setw() to align the test names and stars.

8. Delete All Nodes Function

void DeleteAllNodes (pNode **sPtr);

The **DeleteAllNodes** function deallocates memory for all the nodes in the linked list, ensuring there are no memory leaks. It also resets the head pointer to nullptr/NULL to indicate that the list is empty.

Function Details:

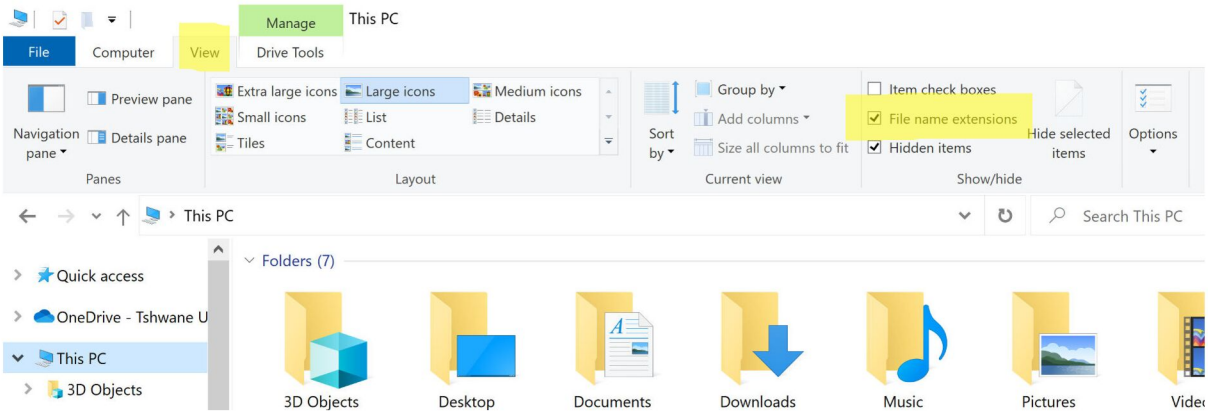
- The function takes a double pointer, sPtr, which points to the head of the linked list. This allows the function to modify the list and reset its start pointer.
- The pointer currentPtr is initialised to point to the first node (*sPtr). The pointer deletePtr is used to store the node that will be deleted.
- The function enters a while loop that iterates through the linked list:
    - deletePtr temporarily holds the current node: currentPtr
    - The pointer currentPtr is moved to the next node: currentPtr = currentPtr->nextPtr.
    - The node stored in deletePtr is then deallocated using the delete operator.
- After all nodes are deleted, the head pointer *sPtr is set to nullptr/NULL, indicating that the list is empty.

Key Points:

- The function deallocates memory for each node to prevent memory leaks.
- Once all nodes are deleted, the start pointer is reset to ensure the list is properly cleared.
- It is essential for managing dynamic memory and ensuring the linked list is safely cleaned up when no longer needed.

HOW TO RUN THE SHOWCASE FILE

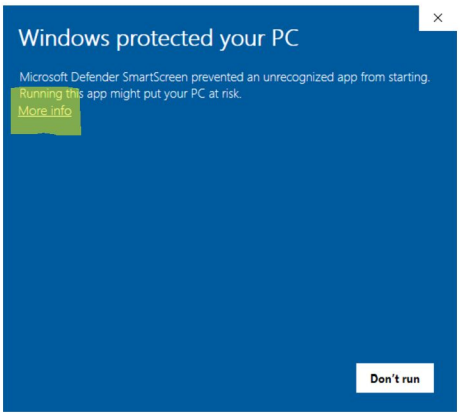
1. Enable file extensions (see highlighted in yellow)



2. Change the name from **"Showcase.old"** to **"Showcase.exe"**

3. Run the **"ShowcaseEV.exe"** by double-clicking on the icon.

4. The following may be shown by Windows. Click on **"More info"**



5. Click on **"Run anyway"**

ANNEXURE A – MARK ALLOCATION

Note: Score range is 0 - 4 which is: 0-none, 1-poor, 2-average, 3-good, 4-excellent

TEST RUBRIC	SCORE [0-4]	WEIGHT [%]
C++ CODE EVALUATION		**64**
1. Populate Node Structure Member Function (Initialize structure members with default parameters)		**5**
2.Calculate Projectile Structure Member Function (Calculate projectile range with given formulas)		**5**
3. Read File And Populate Function (Read text file and populate Linked List with InsertNode)		**6**
4. Insert Node Function (Insert new node at the back of Linked List)		**8**
5. Display Data Table Function (Display projectile nodes data table)		**8**
6. Display Distance and Deviation Table Function (Dispay projectile distance and deviation table)		**8**
7. Display Node Distance Graph Function (Display Distance Graph)		**8**
8. Delete All Nodes Function (Delete all the nodes in the Linked List)		**6**
9. Overall Impression (Neatness, Readability, Spacing, and Indentation)		**5**
10. No Compile or Runtime errors		**5**
TOTAL		**64**
STUDENT NUMBER		

Graduate Attribute	GA Number	GA Score [0-5]
Application of scientific and engineering knowledge	GA2	2,6
Engineering methods, skills, tools, including information technology	GA5	1,3,4,8
Impact of Engineering Activity	GA7	5,6,7
Engineering Professionalism	GA10	9,10

ANNEXURE B – INFORMATION SHEET

Data types: void, char, short, int, float, double

Data Type modifiers: const, auto, static, unsigned, signed

Arithmetic operators: * / % + -

Relational operators: < <= > >= == !=

Assignment operator: = += -= *= /= %= &= ^= |= <<= >>=

Logic operators: && || !

Bitwise logic operators: & | ^ ~ << >>

Pointer operators: Derefernce: * Address: &

Control Structures:

IF Selection:	if (condition) { … };
IF ELSE Selection:	if (condition) { … } else { … };
WHILE Loop:	while (condition) { … };
DO WHILE loop:	do { … } while (condition);
FOR Loop:	for (initial value of control variable; loop condition; increment of control variable) { … }
SWITCH Selection:	switch (control variable){ case ‘value’: … ; break; default: … ; break; }

Functions: return_data_type function_name (parameters) { … };

Common Library Functions: printf() , scanf() , rand() , srand() , time() , isalpha() , isdigit() , getchar() , getch(), strcpy()

Arrays:

One dimensional:	data_type variable_name[size];
Two dimensional:	data_type variable_name [x_size][y_size];

ANNEXURE C – ASCII TABLE

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com