تعریف متغیر :

در مورد بدنه اصلی برنامه *Delphi* گفتیم. در آن بخش در مورد تعریف متغیرها توضیحی داده نشد حال به این مساله و مساله تعریف ثوابت می پردازیم.

بخش *var* شکل جزیی زیر را دارد:

$$var < name1 >:< data-type >;$$

$$[< name2 >:< data-type >;]$$

در مورد نام متغیر در بخشهای قبل صحبت کردیم و همان قواعد در انتخاب $< name >$ ها لازم و کافیند. بخش *type* *data* یا نوع داده هم در واقع اسامی انواع داده ای است که یا شما در بخش *type* تعریف کرده اید و یا در خود *Delphi* استاندارد است. به عنوان مثال *Interger* اسم نوع داده " عدد صحیح " است که در خود *Delphi* وجود دارد. نحوه ایجاد انواع داده در بخش *type* در فصل پنجم به تفصیل مورد بحث قرارخواهد گرفت.

در حال حاضر به انواع داده موجود می پردازیم

عدد صحیح **2** بایتی : *longInt=Integer,SmallInt* و عدد صحیح **4** بایتی : *Integer* عدد حسابی **4** بایتی : *Cardinal* و عدد صحیح **1** بایتی : *shortInt* عدد حقیقی : *Real* و عدد حسابی **1** بایتی : *byte* و عدد حسابی **2** بایتی : *word* رشته حروف : *string* و حرف *char* و عدد حقیقی توسعه یافته : *Extended*

اعداد صحیح : چنانکه قبلاً چند بار ذکر شده داده ها یا همان متغیرها بخشهایی از حافظه اند. لذا خالی از لطف نیست که چگونگی برخورد بصورت عدد صحیح با حافظه را بررسی کنیم!

ساده ترین روش برخورد با حافظه برخورد بصورت عدد حسابی است. در این حالت به سادگی با در نظر گرفتن هر بیت حافظه *Bit* $\overset{D}{=}$ *Binary Digit* رقم دودویی) بصورت رقم مبنای دوی یک عدد بخش حافظه

را عدد حسابی گرفت. مثلاً وقتی ترتیبی از صفر و یکها بصورت زیر در حافظه است:

$$(01101001)_{\text{حافظه}} \overset{D}{=} 105$$

در این روش برخورد همه اعداد مثبت و یا صفرند لذا مجموعه حسابی بدستی می آید.

برای آنکه عدد های منفی را نیز داشته باشیم باید بخشی از فضای متغیر عدد را به مثبت یا منفی بودن آن نسبت دهیم. یک روش ساده در نظر گرفتن بیت علامت است مثلاً آخرین بیت سمت چپ را علامت بگیریم.

عدد علامت

$$(01101001)_{\text{حافظه}} = +105, \quad (11101001)_{\text{حافظه}} = -105$$

حال برای آنکه خود را هم آزموده باشید بدون خواندن پاراگراف بعدی به سئوال پاسخ دهید:

اینگونه در نظر گرفتن منفی و مثبت چه ایرادی است (راهنمایی : چگونه استفاده ناصحیح از حافظه می کند)؟

در این نوع برخورد عدد صفر دو نماد پیدا می کند یکی (**10000000**) و یکی (**00000000**) و این خود عدم استفاده صحیح از حافظه است. اما مساله بعدی جمع و تفریق، دو عدد صحیح به عنوان ساده ترین اعمال روی اعداد است. در جمع اعداد حسابی مشکلی وجود ندارد مانند جمع کردن انسانها رقم به رقم عددها را با هم جمع می کنیم و **2** بر **1** های لازم را هم در نظر می گیریم (به این **2** بر **1** ها که مشابه **10** بر **1** در جمع مبنای ده است رقم *Carry* می گویند.) تنها اشکال ممکن عدم وجود فضای کافی برای ذخیره نتیجه جمع است. مثلاً

به این حالت *Overflow* یا سرریز می گویند زیرا رقم دودویی از ظرف حافظه سرریز کرده است!

	+	**10010000**
		10100100
	1	**00110100**

برای رقم حافظه نداریم

حال برای جمع دو عدد صحیح چه کنیم؟! با روش بالا این کار بسیار مشکل است لذا روش " مکمل دو(*two's complement*)" برای نمایش اعداد صحیح استفاده می شود. ابتدا به پایه تئوری این روش می پردازیم، گفته شد که حافظه ذخیره هر عدد محدود است مثلاً فرض کنید $1\ Byte$ **(۸ بیت) است، در این حالت بیت فرضی نهم را در نظر می گیریم که تاثیری در مقدار حافظه ندارد لذا اگر** a **و** b **دو عدد باشند:**

$$2^8 + a - b \cong a - b \rightarrow a - b \cong a + \left(2^8 - b\right)$$

از طرفی $2^8 - b$ **اگر** b **عدد 8 بیتی باشد حتماً 8 بیتی است یعنی بیت نهم آن صفر است (چرا؟) لذا عمل جمع " با مکمل دو " ی یک عدد معادل تفریق است. این انگیزه باعث می شود که** $-b$ **را در حافظه بصورت** $2^8 - b$ **ذخیره کنیم. ممکن است سئوال پیش بیاید محاسبه** $2^8 - b$ **چگونه است؟ این کار به سهولت و با** not **منطقی کردن عدد و جمع آن با 1 صورت می پذیرد مثلاً**

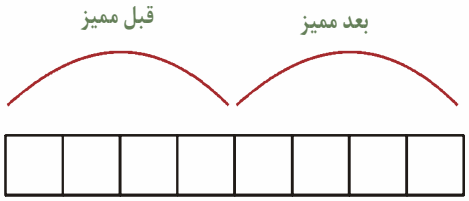
$$2^8 - \overline{(01010011)}_2 = \overline{(10101100)}_2 + 1 = (10101101)_2$$

با این روش علاوه بر آنکه هر عدد حتی نمایش یکتا دارد عمل تفریق هم به سهولت ممکن می شود.

اعداد حقیقی:

برای ذخیره اعداد حقیقی چه روشی را پیشنهاد می کنید؟ باز هم از ساده ترین روش شروع کنیم، یک عدد 1 بایتی حقیقی داریم (چنین نوع داده ای وجود ندارد!) می توان چهار بیت آن را بعد ممیز و چهار

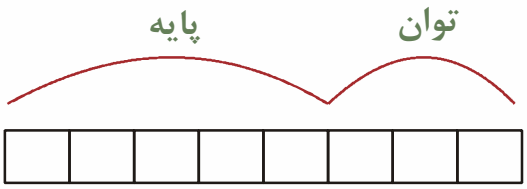
بیت آن را قبل ممیز در نظر گرفت.



$01100111 \cong 6.7$ یا $10100100 \cong 10.2$

باز هم برای آزمودن خود محدودیت های روش را بررسی کنید.

راه دیگر استفاده از روش نماد علمی است. در این روش چه می کردید؟ مثلاً بـرای نمـایش **232/456** روش نمایش بصورت $2/32456\times10^{2}$ است. با این ایده می توان توان ده و مقدار پایه عدد را ذخیره کرد.



$$01100111 \cong 1.2\times10^{7} \qquad 10100100 \cong 2.0\times10^{4}$$
$$\underset{20}{} \quad \underset{7}{} \qquad \underset{20}{} \quad \underset{4}{}$$

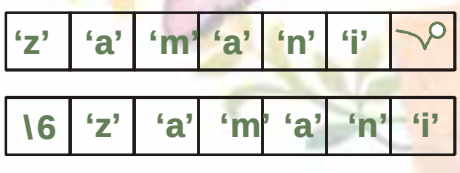
با این روش بازه وسیعتری از اعداد حقیقی قابـل نمایـشند. بـه روش اول *fixed point* و روش دوم *floating point* گویند هر روش مزایا و محدودیت های خاص خود را دارد. روش اول برای محاسبات جمع و تفریق و روش دوم برای ضرب و تقسیم بهترند.

روش استاندارد کار با اعداد حقیقی روش *floating point* است و لذا هر دو نوع *Real Extended* براین مبنا هستند. برای تمرین مشکلات روش *floating point* را در کار با اعـداد بررسـی کنیـد البتـه در استاندارد *floating point* روش نمایش اندکی با نماد علمی فرق دارد و عدد پایه بجای آنکه بین **0**و**10** باشد بین **0**و**1** است یعنی $232/456 = 0/232456\times10^{3}$

رشته ها و حروف :

حروف در دنیای رایانه همان اعداد هستند. در دنیای قدیمی *Dos* تعداد حروف **256** عدد بود. از کد **0** الی کد **255** یعنی **1** بایت، لذا در نمایش حروف که هر حرف را قرار می دادیم و شکل آن نمایش داده می شد مثلاً شکل کد **65** حرف 'A' و شکل کد **48** حرف 'q' بود. در نسخه های قدیمی *Windows* یعنی **3/1**و**95**و**98** هم روش تقریباً بر همین منوال است اما در نسخه های بعدی $(XP,2000,NT)$ هر حرف کد **2** بایتی دارد به کدهای حروف در *Dos* کدهای *ASCII* و به کدهای حروف در *XP* کدهای *Unicode* می گویند. مسلماً در روش *Unicode* تعداد بسیار بیشتری حرف و نماد وجود دارد.

پس از این توضیح باید بگوییم که روش ذخیره حروف در *Delphi* یک کد **1** بایتی است و لذا کاملاً با روش *ASCII* است در مورد حروف $(characters)$ مساله دیگری وجود ندارد اما در مورد رشته ها. طول یک رشته حرفی چقدر است؟ یعنی یک رشته حرفی تا کجای حافظه ادامه دارد؟ برای معین کردن این مساله باید به نحوه ذخیره سازی رشته ها پرداخت. یک روش، روش $ASCIIZ \stackrel{\Delta}{=} ASCII + Zero$ است. یعنی انتهای یک رشته حرفی کد(**0**) است. روش دیگر که روش *Delphi* است ذخیره طول و مشخصات رشته حرفی در ابتدای رشته حرفی است. یعنی ابتدا در حافظه طول رشته را می نویسیم و سپس خود رشته را هر جا درون' ' بوده نماد حرف و هر جا \ قرار دارد کد را نوشته ایم.



برای نسبت دادن مقادیر به متغیرها از دستور $<name>:=<value>$ استفاده می شود بجای *name*

مثال.

```
var
  X  : Integer ;
 Y  : Re al ;
  N  : String ;
  C : Char ;
M
begin
M
  X  :=12;
 Y  :=145.5;
 C :=' B '
  N  :=" Zamani ";
M
end .
```

سه نوع دیگر که نام نبردیم *Pointer* **(اشاره گر) است که در فصل پنجم مورد بررسی قرار می گیرد.**

همچنین نوع داده *Boolean* **(منطقی) که برای ذخیره نتایج منطقی عبارات استفاده می شود.**