گروه ACM دانشگاه آزاد اسلامی واحد پرند

داده های عددی

قسمت اول

تهیه کننده: محسن صفری

تاریخ: ۱۳۹۲/۰۳/۰۲

در این مقاله که در دو قسمت آماده شده است، به نحوه ذخیره سازی و نمایش انواع داده های عددی پرداخته و در ادامه برخی از برداشت ها رایج در مورد داده های عددی بررسی می گردد. آشنایی با این مفاهیم ضروری بوده و کمک شایانی به فهم و درک صحیح از کد نوشته شده خواهد کرد. متن حاضر قسمت اول این مقاله بوده و قسمت دوم آن به زودی ارائه خواهد گردید.

داده های عددی (قسمت اول)

انتخاب نوع داده ی[1] مناسب گاه می تواند تنها تفاوت میان یک راه حل درست و یک راه حل نادرست باشد. به خصوص هنگام حل مسائل هندسی و مشکلات مربوط به گرد کردن و نمایش اعداد اعشاری، ممکن است باعث رد شدن راه حلی درست گردد. برای اجتناب از این گونه مشکلات ابتدا باید به نحوه ی ذخیره سازی و نمایش انواع داده ای در کامپیوتر پی برد. در این نوشتار سعی بر آن است تا علاوه بر روشن ساختن برخی حقایق در این رابطه، چند نمونه برداشت نادرست از نحوه ذخیره سازی و نمایش انواع داده ای معرفی شده و مورد بحث و تحلیل قرار گیرند.

شایان ذکر است این نوشتار به هیچ وجه نمی تواند به عنوان مرجعی کامل در این زمینه تلقی گردد و هدف از نوشتن و انتشار آن عنوان نمودن برخی مطالب ابتدایی و در عین حال کاربردی به زبان بسیار ساده و قابل درک برای کسانیست که در ابتدای راه ای.سی.ام هستند. مطالبی که در ادامه ذکر خواهد شد متمرکز بر ماشین های با معماری X۸۶، که قالب دستگاه های خانگی و شخصی و سرورها را شامل می شود، است. برای مثال فرض بر این است که در ماشین ما یک بایت یا کلمه ۸ بیت است و سیستم از رجیسترهای ۳۲ بیتی برای ذخیره اعداد صحیح استفاده می کند. با وجود آنکه بیشتر مطالبی که عنوان خواهد شد به صورت کلی بوده و قابل تعمیم به اکثر زبان های برنامه سازی است، تمرکز اصلی آن بر روی زبان C++ بوده و در برخی موارد خاص تفاوت های کامپایلرهای g++ و MSVC++ تشریح خواهد شد.

برای شروع جدول داده های عددی صحیح در جدول ۱ نمایش داده شده است. در ادامه ذکر خواهد شد که برای حل مسائل، دانستن میزان حافظه انواع داده های عددی و همچنین آشنایی با بازه ی اعدادی که شامل می شود، حیاتی است.

[1] Data Type

جدول ۱ انواع داده ی عددی صحیح

Name	Size in bits	Representable range
char	۸	$-۲^{۷}$ to $۲^{۷}-۱$
unsigned char	۸	۰ to $۲^{۸}-۱$
short	۱۶	$-۲^{۱۵}$ to $۲^{۱۵}-۱$
unsigned short	۱۶	۰ to $۲^{۱۶}-۱$
int	۳۲	$-۲^{۳۱}$ to $۲^{۳۱}-۱$
unsigned int	۳۲	۰ to $۲^{۳۲}-۱$
long	۳۲	$-۲^{۳۱}$ to $۲^{۳۱}-۱$
unsigned long	۳۲	۰ to $۲^{۳۲}-۱$
long long	۶۴	$-۲^{۶۳}$ to $۲^{۶۳}-۱$
unsigned long long	۶۴	۰ to $۲^{۶۴}-۱$

توجه!!

- نوع داده ی `char` نیز یک نوع **عددی** است و می توان از آن برای ذخیره سازی اعداد استفاده کرد. تنها تفاوت آن با انواع دیگر منحصر به اندازه آن می باشد. با این حال توابع ورودی/خروجی در هنگام نمایش و ذخیره سازی کاراکترها از این نوع استفاده کرده و کد اسکی[۲] حرف مورد نظر را در این نوع ذخیره کرده و

[۲] Ascii Code

نمایش می دهند. عددی بودن `char` امکاناتی در مواجه با حروف به ما می دهد که در برخی موارد (مانند فهمیدن این که حرف مورد نظر چندمین حرف از حروف الفبای انگلیسی است) کاربردهای خاص خود را دارد.

- میزان حافظه برای ذخیره سازی یک `int` و یک `unsigned int` تا حد زیادی بستگی به دستگاه، سیستم عامل و کامپایلر دارد. برای مثال در ماشین هایی که از رجیسترهای ۶۴ بیتی برای ذخیره اعداد استفاده می کنند، در g++، `int` دارای ۶۴ بیت می باشد. با این حال در ماشین های حاضر که اکثر سیستم های فعلی و همچنین ماشین داور را شامل می شود، می توان حداقل ۳۲ بیت برای `int` را تضمین نمود و برای حل مسائل نیز فرض را بر ۳۲ بیتی بودن `int` قرار می دهیم.
- در کامپایلر MSVC++ می توان از `__int64` به جای `long long` استفاده کرد.

در ادامه به برخی برداشت های درست و نادرست در مورد انواع داده ی عددی اشاره خواهد شد.

برداشت: اعداد صحیح با علامت به صورت تک بیت علامت و بیت های ارقام ذخیره می شوند.

درستی: تنها بخشی از این گفته صحیح می باشد.

بیشتر دستگاه های امروزی به ویژه دستگاه های مد نظر ما اعداد را به صورت **مکمل ۲**[۳] ذخیره می کنند. اگر چه در این روش سمت چپ ترین یا پر ارزش ترین بیت، اعداد غیر منفی ۰ و اعداد منفی ۱ می باشد، ولی این برداشت که این بیت تنها یک بیت علامت است صحیح نیست و به عنوان مثال نمی توان در این روش صفر منفی تولید کرد. در روش مکمل ۲ اعداد مثبت به صورت دودویی ذخیره می شوند و سمت چپ ترین بیت مقداری برابر ۰ دارد. ولی طریقه ذخیره سازی

[۳] Two's Complement

داده های عددی (قسمت اول)

اعداد منفی متفاوت است. به عنوان مثال عدد منفی n– به صورت نقیض (یعنی صفرها به یک و یک ها به صفر تبدیل شوند) عدد مثبت n-۱ ذخیره می شود. در جدول ۲ نمایش مکمل ۲ برخی از اعداد که در متغیر از نوع char ذخیره شده اند را مشاهده می کنید.

جدول ۲ نمایش مکمل ۲ برای چند عدد

Value	Two's Complement Form
۰	۰۰۰۰۰۰۰۰
۱	۰۰۰۰۰۰۰۱
۲	۰۰۰۰۰۰۱۰
۴۶	۰۰۱۰۱۱۱۰
۴۷	۰۰۱۰۱۱۱۱
۱۲۷	۰۱۱۱۱۱۱۱
-۱	۱۱۱۱۱۱۱۱
-۲	۱۱۱۱۱۱۱۰
-۳	۱۱۱۱۱۱۰۱
-۴۷	۱۱۰۱۰۰۰۱
-۱۲۷	۱۰۰۰۰۰۰۱
-۱۲۸	۱۰۰۰۰۰۰۰

داده های عددی (قسمت اول)

باید توجه داشت که با توجه به نحوه ی ذخیره سازی اعداد منفی مجموعه اعداد قابل نمایش در یک نوع عددی به صورت قرینه حول صفر نخواهد بود و بزرگترین عدد مثبت در نوع b بیتی برابر با $۲^{b-۱}$-۱ و کوچکترین عدد منفی برابر با $-۲^{b-۱}$ است.

یک راه برای محاسبه عددی منفی که به صورت مکمل ۲ ذخیره شده است این است که به جز بزرگترین توان ۲ که سمت چپ ترین بیت می باشد، باقی توان ها را جمع کرده و از بزرگترین توان کم کنیم. به عنوان مثال عدد منفی ۸ بیتی ۱۱۰۱۰۰۰۱ برابر است با

$$۱ \times (-۲^{۷}) + ۱ \times ۲^{۶} + ۰ \times ۲^{۵} + ۱ \times ۲^{۴} + ۰ \times ۲^{۳} + ۰ \times ۲^{۲} + ۰ \times ۲^{۱} + ۱ \times ۲^{۰} = -۱۲۸ + ۸۱ = -۴۷$$

برداشت: اعداد صحیح بدون علامت به صورت دودویی ذخیره می شوند.

درستی: صحیح

الگوی اعداد بدون علامت کاملاً دودویی است برای مثال اگر عدد ۱۱۰۱۰۰۰۱ که در مثال قبل ارزش عددی آن را دیدید را به صورت بدون علامت فرض کنیم برابر خواهد بود با

$$۱ \times ۲^{۷} + ۱ \times ۲^{۶} + ۰ \times ۲^{۵} + ۱ \times ۲^{۴} + ۰ \times ۲^{۳} + ۰ \times ۲^{۲} + ۰ \times ۲^{۱} + ۱ \times ۲^{۰} = ۲۰۹$$

بنابراین در اعداد صحیح بدون علامت b بیتی کوچکترین عدد ۰ و بزرگترین عدد ۲^{b}-۱ خواهد بود.

برداشت: در C++ تکه کد زیر عدد x را در تمام خانه های آرایه A ذخیره می کند.

```
int A[1000];

memset(A, x, sizeof(A));
```

درستی: غلط

تابع `memset()` قسمتی از حافظه را به صورت بایت به بایت پر می کند. به ازای بیشتر مقادیر x به نتایجی دور از انتظار دست خواهید یافت؛ مگر اینکه به جای `int` از `char` استفاده کنید. با این حال این توابع به ازای ۰ و ۱- به درستی عمل می کند، زیرا در ۰ تمام بیت ها صفر و در ۱- تمام بیت ها یک است.

توجه داشته باشید که اکثر پردازنده ها، عملگر هایی برای پر کردن قسمتی از حافظه با مقداری خاص را دارند؛ ولی `memset()` بسیار سریع تر از پر کردن حافظه در یک سیکل است.

همنگامی که بدانید چطور از این تابع استفاده کنید، می توانید از آن برای کاربردهای دیگر استفاده کنید. مثلا به وسیله آن می توان حافظه را با مقادیر بسیار کوچک یا بسیار بزرگ پر کرد.