بنام خدا

درس برنامه نویسی بزبان C

معرفی کتاب :

کتاب جامع برنامه نویسی سی اثر بایرون گاتفرید-

کتاب جامع برنامه نویسی بزبان سی اثر عین ا... جعفرنژاد قمی –

کتاب برنامه نویسی سی اثر دنیس ریچی

(بسیاری از مباحث داخل کلاس مثل برنامه ها یا الگوریتم ها بنا به بعضی ملاحظات مثل کمی وقت و یا الزام شما به شرکت در کلاس در این جزوه نخواهد بود و لذا از بابت آن عذر خواهی بنده را بپذیرید.)

مراحل برنامه سازی :

قبل از برنامه نویسی باید صورت مساله را بخوبی تحلیل نموده و شناخت خوبی از آن بدست آورد اینکار کمک می کند امکان سنجی های لازم از لحاظ زمان و هزینه مصرفی تعیین شده و مشخص گردد آیا پروژه مذکور از لحاظ اقتصادی یا بعضی جنبه های دیگر مقرون به صرفه خواهد بود یا نه؟و بدنبال آن یک استراتژی خاصی برای برنامه نویسی اتخاذکرد تحلیل و شناخت مساله و نیازهای آن و طراحی الگوریتم بهینه ای که تا حد امکان در حافظه صرفه جویی های لازم را کرده باشد و بنا به مقتضیات از سرعت خوبی برخور دار بوده و ساختمان داده های مناسبی را برای برنامه در نظر گرفته باشد از اهم مسایل است بعداز اینکه الگوریتم مناسب تهیه شد و روش حل مساله شناخته شد نوبت به کد نویسی برنامه و یا اصطلاحا پیاده سازی آن میرسد .

برنامه بهتر است در زبان سی از روش طراحی بالا به پایین نوشته شود یعنی ابتدا به روالهای کلی تر و در ادامه به روالهای جزیی تر پرداخته شود حسن اینکار سهولت در فهم مساله خوانایی و قدرت بیشتر برنامه و رفع اشکال راحتتر ان است یعنی با ماژولار کردن برنامه نویسی فهم برنامه بهتر انجام می شود.و بعدا اشکالزدایی (debugging) ان راحتتر انجام خواهد شدبعد از کد نویسی برنامه برنامه با یکسری داده ها و توسط افراد مختلف و در سطوح مختلف تست می شود تا اشکالات احتمالی رفع گردد بعد ازان مرحله نگهداری و تعمیم یا بهینه کردن و توسعه دادن کارایی کد برنامه مطرح می شود در اینکار مستند سازی(مدیریت پیکر بندی) قبلی برنامه نقش مهمی را دارد اعم از تهیه کتابچه راهنمای نرم افزار تهیه شده (user guide)و یا مستنداتی که در داخل فایل برنامه بین /*.....*/ قرار میگیرد.و کامنت نام دارد .

معیار های یک برنامه خوب :

یک برنامه در صورتی کارایی های لازم را خواهد داشت که برنامه نویس با شناخت کافی از مساله یک الگوریتم بهینه ای را برای آن انتخاب نموده و از ساختمان داده های مناسب برای آن بهره ببرد .یک برنامه خوب باید تا حد امکان انعطاف پذیر باشد و منابع کمتری از سیستم را در اشغال خود نگه دارد برای مثال حافظه کمتری را استفاده نماید و از سرعت اجرایی خوبی برخوردار باشد تا تمام نیازهای کاربران را در هر سطحی براورده نموده و از کارایی خوبی برخوردار باشد.البته بیشتر اوقات برنامه نویس مجبور می شود از بین حافظه کمتر سرعت بیشتر یکی را انتخاب نماید.

مراحل اجرای برنامه :

مثل تمام زبانهای برنامه نویسی مجبوریم کدهای برنامه را داخل یک ادیتور تایپ نماییم کد برنامه در داخل یک ادیتور مثل ادیتوری که محیط توربو سی یا بورلند سی ارائه کرده نوشته می شود فایل مربوطه تحت عنوان یک فایل برنامه نویسی شده با پسوند C. یا CPP. روی دیسک ذخیره می شود و چنانچه بخواهیم می توان انرا کامپایل نمود انجام اینکار باعث ایجاد یک فایل با پسوند OBJ. می شود که ترکیبی از ۰و۱ هاست بعد نوبت به لینک این فایلهای ابجکت رسیده این فایلها بهم لینک شده و برنامه اجرا میشود و یا اینکه فایلی با پسوند EXE. ساخته می شود. این فایل اجرایی برنامه را میتوان در سطح سیستم عامل و بدون نیاز به محیط سی و احیانا با دادن پارامترهای آن اجرا نمود که بعدا در مورد آن بیشتر خواهیم خواند.

لازم بذکر است کامپایلر های متنوعی برای زبان سی ساخته شده ولی مد نظر ما سی استاندارد ANSI می باشد که تمام کامپایلر ها از آن پشتیبانی مینمایند.

ویژگی های زبان برنامه نویسی سی:

زبان سی در سال ۱۹۷۰ توسط دنیس ریچی و بر پایه زبان BCPL که آنهم مشتق از زبان B بود بوجود آمد .زبان برنامه نویسی C یک زبان سطح میانی است از آنجهت که عناصر سطح بالا را با خصیصه تابعی زبان اسمبلی در هم آمیخته و لذا نباید نگران این بود که برنامه نویسی در این زبان به سختی زبانهایی مثل اسمبلی باشد و یا اینکه قدرت این زبان کمتر از قدرت زبانهایی مثل بیسیک یا پاسکال باشد.در تقسیم بندی زبانها باید گفت زبانهای سطح پایین به سخت افزار نزدیکتر بوده و برای مثال برای کار برروی رجیستر ها و آدرسهای حافظه امکاناتی را پیشنهاد کرده اند و بخاطر همین سرعت اجرای بالای دارند ولی در عوض برنامه نویسی بزبانهای سطح بالا راحتتر بوده چون به زبان محاوره ای ما نزدیک می باشد ولی زبان سی در واقع همه این معیار ها را در خود دارد هم دارای عملگر ها و امکاناتی است که بتوان رجیستر ها و آدرسهای حافظه را در اختیار گرفته تست یا مقداردهی کرد و هم از سرعت بالایی برخوردار است و قدرت زیادی را در اختیار برنامه نویس قرار می دهد برای طراحی نرم افزارهای پیچیده .....CAD- OS-COMPILER-INTERPRETER-EDITORS AND SO ON

۱- مثل سایر زبانهای میانی قابلیتهایی برای کار کردن (تست مقداردهی و شیفت)بیت - بایت و ادرسهای حافظه که از عناصر ابتدایی ماشین هستند را برای برنامه نویس ایجاد میکند.عملگرهایی برای اینکار دارد که فقط روی متغیرهایی از نوع int , char عمل کرده و میتوانند آدرسهای حافظه و بیتها را دچار تغییر نمایند.

۲- برنامه یا کدهای نوشته شده به این زبان براحتی قابلیت حمل از یک کامپیوتر سازگار با ibm به کامپیوتری از نوع apple را دارنداین ویژگی را portability گویند بخاطر داشتن عملگر هایی مثل (نوع متغیر) sizeof در واقع زبان سی یک زبان مستقل از ماشین است و این باعث صرفه جویی زیادی در وقت و هزینه های برنامه نویس است مخصوصا اگر بخواهد برنامه نوشته شده برای یک کامپیوتر IBM را برروی یک کامپیوتر از نوع دیگر اجرا نماید.

۳- این زبان یک زبان سیستمی با قدرت و قابلیتهای فوق العاده برای طراحی مفسرها کامپایلر ها و سیستم های عامل و ویراستارها و همینطور CAD می باشدو همین طور نرم افزارهای مدیریت بانکهای اطلاعاتی را براحتی میتوان با زبان سی کد نویسی و اجرا نمود.

۴- تعداد کلمات کلیدی رزرو شده بسیار کمی دارد و به حروف کوچک و بزرگ حساس است لذا این زبان int را بعنوان یک کلمه کلیدی می شناسد ولی INT را نه.تمام کلمات کلیدی این زبان بحروف کوچک نوشته می شوند.

۵- این زبان یک زبان ساختیافته است که میتوان مجموعه ای از دستورالعملها را در آن داخل { } نوشت و متغیر های تعریف شده در این بلاک برنامه فقط در این بلاک معنی داشته باشند یا اصطلاحا محلی این بلاک باشند.و نیزمثل فرترن نوشتن کدهای برنامه به سطر و ستون خاصی از فایل وابسته نباشد.ضمنا در این زبان کمتر سعی میشود از ساختار goto استفاده شود زیرا از

خوانایی برنامه کاسته و ایجاد اشکال(bug) می نماید . به جای ان از ساختار هایی مثل while,DO WHILE ,for,if else در این زبان استفاده می شود .

۶- هر خط برنامه ممکن است شامل چند دستورالعمل باشد که با : خاتمه می یابند و بالعکس هر دستور ممکن است در چند خط جابگیرد.

۷- بهتر است برنامه نویس برای صرفه جویی های بهتر و بیشتر کردن خوانایی برنامه از comment در برنامه خود استفاده نماید

۸- این زبان خطای boundary checking یا تجاوز از حدود تعیین شده را برای مثلا آرایه ها را نادیده میگیرد وکامپایلر اعلام error نمیکند. ضمن اینکه در گونه های قدیمی سی اگر نوع و تعداد آرگومانهای توابع و پارامتر های تعریف توابع با هم یکی نبودند اعلام خطا نمیشد که در نسخه های جدید با معرفی الگوی توابع (function prototyping) این معضل حل شده و باید حتما ارگومان توابع و پارامترهای دسترسی به تابع باید از یک نوع و به یک تعداد باشند.

۹- زبان برنامه نویسی سی مثل بقیه زبانها دارای تعدادی نوع داده DATA TYPE می باشد که بازه خاصی را شامل شده و عملیات خاصی بر مقادیر این بازه قابل انجام است در سی ۵ نوع داده اصلی داریم که شامل CHAR برای شناسه های از نوع کاراکتری و با طول یک بایت . INT برای شناسه های از نوع عددی صحیح و با طول ۲ بایت- FLOAT برای شناسه های عددی اعشاری با طول ۴ بایت – DOUBLE برای شناسه های از نوع عددی اعشاری با طول ۸ بایت و VOID برای توابع .بجز VOID نوع داده های دیگر با SIGNED-UNSIGNED – LONG – SHORT ترکیب شده و انواع جدیدی را می سازند برای نمونه LONG INT برای شناسه هایی از نوع اعداد صحیح و با طول ۴ بایت بکار میرود.البته بازه پوشش داده شده توسط نوع داده به نوع پردازشگر و نحوه پیاده سازی کامپایلر بستگی زیادی دارد.

۱۰- در زبانهای برنامه سازی دیگر مثل پاسکال و.... چند نوع زیر برنامه داریم یکی زیر برنامه تابع و دیگری زیر برنامه زیر روال –اما در زبان سی آنچه هست زیر برنامه های از نوع تابع است و دیگر هیچ – توابع داخل همدیگر نمی توانند تعریف شوند ولی فراخوانی یک یا چند تابع داخل تابع دیگر امکان پذیر است .در سی تابع اصلی MAIN نام دارد که توابع دیگر باید در آن تعریف شده و فراخوانی شوند در هنگام تعریف توابع لازم است نوع و تعداد آرگومانهای تابع با نوع و تعداد پارامترهای فراخوانی تابع یکسان باشند وگر نه کامپایلر با خطای مناسبی از ادامه اجرای برنامه جلوگیری خواهد کرد.

انواع خطاهای برنامه نویسی :

۱- خطاهای نحوی یا SYNTAX ERROR که مربوط به املای لغات دستورات زبان بوده و در زمان کامپایل به برنامه نویس گزارش می شود.

۲- خطاهای زمان اجرا که در واقع هنگام کامپایل نادیده گرفته می شود و در زمان اجرا جلوی ادامه اجرای برنامه گرفته شده و با پیام مناسب به کاربر گزارش می شود مثل تقسیم بر صفر یا OVERFLOW یا جذر اعداد منفی و....

۳- خطاهای منطقی که مربوط به منطق برنامه بوده و به الگوریتم استفاده شده توسط برنامه نویس بستگی دارد این خطاها نه در زمان کامپایل و نه در زمان اجرا مشخص نمی شوند .ولی عدم کارایی درست برنامه برنامه نویس را به اشتباه بودن منطق برنامه هدایت می کند.

متغیر :

متغیر نامی است برای آدرسی از حافظه که قرار است در طول اجرای برنامه محتویاتی داشته باشد متغیرها مقدار اولیه ندارند و تا تعریف نشوند قابل استفاده در برنامه نخواهند بود می توان با استفاده از کاما چند متغیر را با هم تعریف نمود ضمن اینکه در هنگام تعریف میتوان به متغیر ها مقدار اولیه نیز داد.میتوان با دستور CONST متغیر ها را در برنامه طوری تعریف نمود که در تمام طول برنامه مقدار ثابتی داشته باشند.

عملگر ها در زبان سی:نماد هایی برای انجام اعمالی خاص.

۱- عملگر های محاسباتی مثل -- ++ % - / * + لازم به توضیح است عملگر ++ یک واحد به عملوند خود اضافه میکند.همین طور عملگر -- یک واحد از عملوند خود کم می کند.ولی ++xبا x++ تفاوت دارد x++ ابتدا یکی به عملوند خود اضافه نموده وبعد مقدار جدید عملوند را مورد استفاده قرار می دهد ولی ++x ابتدا مقدار فعلی عملوند را استفاده نموده و بعد به آن یکی اضافه می کند.

۲- عملگر های رابطه ای که رابطه بین دو یا چند عملوند را نشان می دهند مثل =! == => > =< < >

۳- عملگر های منطقی ! || && لازم به توضیح است ارزش درستی در زبان سی با مقداری غیر صفر و ارزش نادرستی با مقدار صفر نمایش داده می شود.

۴- عملگر های دستکاری بیتها >> << ~ ^ | &

۵- عملگرهای * &

۶- عملگر کاما

۷- عملگر sizeof

۸- عملگر ?

تبدیل انواع

وقتی که متغیرهای با نوع های مختلف در یک عبارت با یکدیگر ترکیب می شوند باید تبدیل نوع صورت گیرد . قاعده کلی این است که نوع های با طول کوچک تر به نوع هایی با طول بزرگتر تبدیل می شوند . مثلا" اگر دو متغیر از نوع کاراکتری و عددی صحیح با یکدیگر ترکیب شوند ، نوع کاراکتری به عددی صحیح تبدیل می شود .

مثال ۱: char ch;

```
int i;
float f;
double d;
result =( ch/i )+( f*d-f+i)
```

همان طور که از مثال ۱ پیداست ، نوع نتیجه double خواهد بود .

علاوه بر تبدیل انواع در عبارات ، در احکام انتساب نیز ممکن است تبدیل انواع صورت گیرد . در تبدیل انواع اطلاعاتی از بین می روند که باید در نتیجه حاصل از احکام انتساب دقت کافی به خرج داد. در ذیل تبدیل انواع در احکام انتساب مشاهده می شود

```
 int x;
char ch;
float f;
ch= x;
x= f;
f= ch;
f= x;
```

کلیه احکام انتساب فوق قابل انجام بوده و از طرف کامپایلر زبان C هیچ گونه خطایی گزارش نمی شود . اطلاعاتی که ممکن است در تبدیل انواع از بین بروند . . وقتی یک متغیر int به یک متغیر char انتساب داده می شود بایت کم ارزش متغیر از نوع int به متغیر char منتقل شده ولی با ارزش متغیر int در متغیر char جایی میگوییم در انتساب متغیرintبه char ، 8، بیت با ارزش از دست میرود .

موضوع :ساختارهای تکرار و تصمیم ---توابع و روشهای فراخوانی------کلاسهای حافظه

در حالت پیش فرض تمام دستورات برنامه نوشته شده از بالا به پایین و بترتیب اجرا خواهند شداگر لازم باشد بنا به صلاحدید و بر طبق الگوریتم برنامه میتوانیم تحت شرایط خاصی یک یا چند دستور را اجرا کنیم یا چند دستور دیگر را برای چند بار اجرا نماییم در این صورت از ساختارهای تکرار و تصمیم که در همه زبانهای برنامه نویسی وجود ارد استفاده خواهیم کرد.ساختارهایی مثل

```
For( مقدار اولیه اندیس حلقه;شرط حلقه;گام حرکت)
{
دستور یا بلاکی از دستورات لازم برای اجرا
}
```

گام حرکت و اندیس حلقه میتوانند مثبت منفی یا اعشاری باشندضمن اینکه ساختار حلقه مربوطه برای مواقعی در برنامه مناسب است که تعداد دفعات تکرار حلقه را بدانیم.اگر خواسته باشیم حلقه ما تا بینهایت ادامه داشته باشد ..

```
For(;;)
{
دستور یا بلاکی از دستورات لازم برای اجرا
}
```

ساختار بعدی ساختارwhile می باشد که بدرد مواقعی می خورد که ما تعداد دفعات تکرار حلقه را ندانیم و منتظر براورده شدن یا نشدن شرطی باشیم در اینحالت از این ساختار بهره خواهیم گرفت.

```
While(شرط)
{
دستور یا بلاکی از دستورات لازم برای اجرا
}
```

با رسیدن کنترل اجرای برنامه به این ساختار حلقه ابتدا شرط حلقه ارزیابی می شود و اگر ارزش منطقی این شرط درست بود در این صورت دستور یا بلاکی از دستورات نوشته شده در این ساختار اجرا خواهند شدوگرنه به اولین دستور بعد از این ساختار خواهیم رفت.برای استفاده از این ساختار در حالت حلقه های تکرار بینهایت کافی است while(1) مورد استفاده قرار گیرد چون شرط همیشه برقرار است لذا حلقه تا فشردن ctrl+break ادامه خواهد داشت.به این نوع حلقه های تکرار حلقه های بینهایت گفته می شود.

ساختار do { } while( ) ابتدا در این ساختار دستور یا مجموعه دستورات اجرا می شود و آنگاه شرط while تست می گردد تا زمانیکه شرط مصداق داشت دستور یا دستورات do اجرا خواهند شد فرق این ساختار با ساختار قبلی while در این است که دستور یا دستورات این حلقه حد اقل یکبار اجرا خواهند شد.....(ادامه....ساختارهای switch - continue----break---if-----ifelse

فراخوانی توابع

قبلا گفتیم در اولین جای برنامه بعد از main() که تابع اصلی برنامه است باید الگویی از توابع مورد استفاده در برنامه را به کامپایلر معرفی نماییم این الگو شامل نوع خروجی تابع نام و در داخل پرانتز نام و نوع آرگومانهای ارسالی برای تابع مر بوطه بودند کامپایلر با دیدن نام و آرگومانهای فراخوانی در هر جای برنامه با شناخت قبلی که از تابع مربوطه داشت اگر نوع و تعداد ارگومانهای ارسالی برای تابع فراخوانی شده با نوع و تعداد پارامترهای تعریف تابع فراخوانی شونده یکسان نبود اعلام خطای مناسب می کند .و برنامه نویس را وادار به تصحیح میکرد.

لازم به توضیح است که فراخـوانی هر تابع به سادگی ذکر نام تابع مربوطه و داخل پرانتز آرگومانهای آن می باشد.این کار یعنی فراخوانی توابع داخل هر تابعی امکان پذیر است یعنی هر تابع می تواند یک یا چند تابع دیگر را فراخوانی نماید ولی تعریف توابع باید بیرون از همدیگر انجام گیرد.

اگر هنگام تعریف تابع نوع خروجی تابع ذکر نگردد کامپایلر فرض را بر این می گذارد که تابع مذکور مقدار صحیحی را به تابع فراخواننده بر خواهد گرداند یعنی خروجی توابعی که در هنگام تعریف نوع خروجی آنها ذکر نمی گردد بصورت پیش فرض int می باشد

کلیه توابع باید بعد از ()main تعریف شوند.و برای اجرای هر تابع باید انرا فراخوانی کرد.
متغیر های مورد استفاده هر تابع باید در ابتدای تعریف تابع ذکر شوند همانطور که متغیر های مورد استفاده در تابع اصلی ()main در ابتدای برنامه و بعد از آکولاد شروع main تعریف می شوند.توابع از لحاظ مقداری که به تابع فراخواننده برمیگردانند ۳ نوع اند:
۱-توابعی که هیچ مقداری را برنمی گردانند.
۲- توابعی که فقط یک مقدار برمیگردانند
۳- توابعی که چند مقدار را بعنوان نتیجه عمل برمی گردانند.

تعریف آرگومان و پارامتر:
متغیر هایی که در هنگام فراخوانی تابع مورد استفاده قرار میگیرند آرگومان نام دارند .و متغیر هایی که در هنگام تعریف تابع مورد استفاده قرار می گیرند پارامتر نام دارندنوع و تعداد این دو باید برای یک تابع یکسان باشد اما نام های آنها الزاما نه.

توابعی که هیچ مقداری را برنمی گردانند اصطلاحا توابع void نامیده می شوند که در واقع با فراخوانی این توابع در واقع عملیات خاصی انجام پذیرفته و قرار نیست مقداری به تابع فراخواننده برگردانده شود.بهتر است اگر تابعی خروجی ندارد و یا آرگومانی برای ارسال نمی خواهد با ذکر کلمه کلیدی void آنرا به کامپایلر گزارش نماییم (void sum(void دلیل این کار این است که ممکن است با خطایی از ناحیه کامپایلر روبرو نشویم پس بهتر است محکم کاری نموده و تعیین کنیم که ایا تابع ما ارگومان یا خروجی خواهد داشت یا نه؟
اما اگر قرار باشد تابع مقداری را به تابع فراخواننده بر گرداند این کار با دو روش فراخوانی انجام خواهد شد

۱- روش فراخوانی با ارزش call be value
۲- روش فراخوانی با ارجاع call by refrence

اما روش اول به این معنی است که مقدار دو آرگومان در پارامتر های متناظر کپی می شود لذا هر تغییری در پارامترها بدهیم هیچ گونه تغییری در آرگومانهای تابع نخواهد داشت لذا در این روش هیچ مقداری توسط آرگومان ها یا پارامترها به تابع فراخواننده برگشت داده نمی شودولی می توان با استفاده از دستور (.....)return فقط یک مقدار را به تابع فراخواننده برگرداند. اگر دستور مربوطه را بدون پرانتز استفاده کنیم مفهوم آن فقط خاتمه اجرای تابع خواهد بود و هیچ مقداری به تابع فراخواننده برگشت داده نخواهد شد.اگر با دستور return بخواهیم مقداری را به تابع فراخواننده بر گردانیم این مقدار باید با نوع خروجی تابع فراخوانی شده مطابقت داشته باشد وگرنه با اعلام خطای کامپایلر از اجرای برنامه باز خواهیم ماند.
روش دوم که روش فراخوانی با ارجاع نامیده می شود به این صورت است که بجای مقدار آرگومان آدرس ان در پارامتر متناظر کپی می شود و لذا هر تغییری در پارامتر منجر به همان تغییر در آرگومان مربوطه می شود.بحث بیشتر در این رابطه مقوله اشاره گر ها را باز می کند که در اینجا مجال پرداختن به ان نیست .

کلاسهای حافظه :
تاکنون برنامه هایی را نوشته ایم که در بعضی از آنها از چند تابع نیز استفاده شده است . در این برنامه هامتغیرهای مورد نیاز هر تابع را در همان تابع تعریف کرده ایم . متغیرهایی که در یک تابع تعریف می شوند در موقع انتقال کنترل اجرای برنامه به آن تابع ایجاد شده (حافظه به آنها تخصیص داده می شود) و در موقع برگشت کنترل اجرای برنامه از آن تابع ، حافظه تخصیص یافته به متغیرهای آن تابع از آنها اخذ میشود و به سیستم عامل برگردانده خواهد شد. یعنی این گونه متغیرها فقط در همان تابعی که تعریف شوند اعتبار دارند و در تابع دیگر قابل استفاده نیستند .
این مساله حتی در مورد متغیرهای همنام نیز صادق است . بدین معنی که اگر به عنوان مثال متغیری به نام I در تابع اصلی (main()) وجود داشته باشد و متغیری تحت همین نام در تابع فرعی دیگری که توسط تابع main() فراخوانی می شود تعریف گردد این دو متغیر کاملا" از یکدیگر مجزا خواهند بود و در حافظه ، دو محل جداگانه برای آنها در نظر گرفته می شود و هرگونه تغییر در یکی از آنها ، در دیگری تاثیری نخواهد داشت . این مساله در رابطه با بلاکهای برنامه نیز صادق است یعنی میتوان دو متغیر هم نوع یکی بیرون بلاک و دیگری درون بلاک داشت که مقادیر متفاوت و از انواع متفاوت

بگیرند و هیچ منافاتی با هم نداشته باشند.یعنی دوجای متفاوت از حافظه را در اشغال خود داشته باشند. با مطالبی که تاکنون گفته شد روشن گردید که متغیرهایی که در یک تابع تعریف میشوند فقط درهمان تابع قابل استفاده هستند و به محض برگشت کنترل اجرای برنامه از آن تابع ، این متغیرها عملا" وجود ندارند. به چنین متغیرهایی که محدوده حضور آنها فقط در تابعی است که در آن تابع تعریف شده اند متغیرهای محلی گفته می شود .همه متغیرها در زبان C دارای چنین خصیصه ای نیستند. بلکه انواع دیگری ازمتغیرها نیز وجود دارند که برای معرفی آنها لازم است به تشریح کلاس حافظه بپردازیم.

کلاس حافظه هر متغیر دو چیز را برای آن متغیر مشخص می کند : مدت حضور آن در برنامه یا طول عمرمتغیر . ۲محدوده حضور آن در برنامه (scope visibility) . یکی از علل مطالعه کلاس های حافظه این است که با داشتن اطلاعاتی در مورد مدت حضور متغیرها در برنامه و همچنین محدوده حضور آنها ، می توانیم برنامه هایی
بنویسیم که : از حافظه کامپیوتر به نحو خوبی استفاده گردد . سرعت اجرای آنها بالا باشد
برنامه ها کمتر با خطا مواجه شوند و عیب یابی سهل تر گردد .

کلاس های حافظه ای که در زبان C از آنها استفاده می شود عبارتند از :
کلاس حافظه اتوماتیک . (auto)
کلاس حافظه استاتیک . (static)
کلاس حافظه ثبات . (register)
کلاس حافظه خارجی . (extern)

کلمات کلیدیauto ، static، register،وextern و بترتیب مشخص کننده کلاس های حافظه اتوماتیک ، استاتیک ، ثبات و خارجی هستند. این کلمات قبل از کلمات کلیدی تعیین نوع متغیر بکار می روند . شکل کلی تعیین کلاس ای حافظه بصورت زیر است
بعنوان مثال دستور ; auto int i متغیرهای i را از نوع " صحیح " و با کلاس حافظه اتوماتیک تعریف می نماید .

کلاس حافظه اتوماتیک auto

متغیرهایی با کلاس حافظه اتوماتیک ، بیشتر ازسایر انواع متغیرها مورداستفاده قرار می گیرند . متغیرهایی که تاکنون در برنامه ها استفاده شده اند ، دارای کلاس حافظه اتوماتیک بوده اند. یعنی در زبان C متغیرها در حالت عادی دارای کلاس حافظه اتوماتیک هستند (حالت پیش فرض کامپایلر برای متغیرهای تعریف شده در برنامه) متغیرهای با کلاس حافظه اتوماتیک دارای خصوصیات زیر می باشند: ۱وقتی در یک تابع و یا یک بلاک تعریف می شوند فقط درآن تابع و یا بلاک قابل دسترسی اندیا اصطلاحا محلی آن تابع و یا بلاک محسوب می شوند) . ۲در موقع ورود به تابع و یا بلاک به آنها حافظه اختصاص می یابد . ۳در هنگام خروج از تابع و یا بلاک ، حافظه اختصاص یافته به آنها، پس گرفته می شود.متغیر های اتوماتیک میتوانند مقدار اولیه بگیرند و جالب است که اگر در تابعی مقدار دهی اولیه شوند که n بار فراخوانی شده در هر بار مستقل از دفعه قبل مقدار دهی اولیه می شود بر خلاف متغیر های از نوع کلاس حافظه استاتیک که در آنها مقدار دهی اولیه فقط یکبار انجام می شود و اصلا بخاطر همین نام استاتیک را بر این کلاس حافظه گذاشته اند.

کلاس حافظه عمومی –گلوبال یا extern یا خارجی:

نوع دیگری از متغیرها هستند که می توانند توسط تمام توابع استفاده شده در برنامه به رسمیت شناخته شوند یعنی یکبار آنها را بیرون از تابع اصلی برنامه main() تعریف نماییم و از آنها در تمام طول اجرای برنامه و در تمام توابع بهره بگیریم.البته باید در نظر داشت برنامه ما قرار است از لحاظ حجم حافظه مصرفی صرفه جویی های لازم را کرده باشد.حتی اگر بخواهیم میتوان با ذکر کلمه کلیدی extern برای کامپایلر تعیین کنیم که در فایل دیگری متغیر هایی داریم که از نوع خارجی هستند تا برای استفاده از آنها در این فایل جاری هم مشکلی نداشته باشیم برای حالتی که برنامه ما بیش از یک فایل بوده و در واقع یک فایل پروژه شامل چند فایل می باشد.طول عمر متغیر های خارجی کل زمان اجرای برنامه است.

. درشروع اجرای برنامه برای متغیرهای عمومی ، حافظه اختصاص می یابد و درپایان اجرای برنامه ، این حافظه از متغیرها اخذ می گردد. بنابراین اگر تعداد متغیرهای عمومی زیاد باشد تاثیر بسزایی در میزان حافظه مصرفی دارد . بهر حال در مواقعی که توابع موجود در برنامه به متغیرهای مشترکی نیاز

داشته باشند ، خوب است از متغیرهای عمومی استفاده گردد. ولی باید توجه داشت متغیرهایی که بندرت در سراسر برنامه مورد استفاده قرار می گیرند ، بصورت متغیرهای عمومی تعریف نگردند . اگر برنامه هایی که به زبان C نوشته می شود طولانی باشد ، می توان آن برنامه را به قسمت های منطقی کوچکتری تقسیم کرده ، سپس هر قسمت را در یک فایل قرار داده و پس از کامپایل کردن آنها به طور مجزا ، همه قسمت ها را با یکدیگر اجرا نمود . به هر یک از این قسمت های مجزا ، یک واحد (unit) گفته می شود . اگر متغیرهایی در واحد اصلی تعریف شوند و بخواهیم از آنها در واحدهای فرعی دیگر استفاده کنیم (بدون این که در واحدهای فرعی به آنها حافظه ای اختصاص یابد (باید در واحدهای فرعی توسط کلمه کلیدی extern به کامپایلر گفته شود که این متغیرها در جای دیگری (در واحد اصلی) تعریف شده اند. در غیر این صورت استفاده از متغیرهای مورد نظر در واحدهای دیگر ممکن نیست .

کلاس حافظه استاتیک :static

این کلاس حافظه دو نوع است الف استاتیک محلی ب- استاتیک عمومی

متغیر های استاتیک محلی فقط در همان تابعی که تعریف می شوند قابل دسترسی اندو ارزش دهی اولیه به این کلاس حافظه فقط یکبار صورت خواهد گرفت و در هنگام خروج از تابع آخرین مقداری که در تابع به آنها نسبت داده شده بعنوان مقادیر متغیر شناخته خواهند شد.

متغیر های استاتیک عمومی فقط در یک فایل یا unit برنامه از جایی که تعریف می شوند به بعد قابل دسترسی اند این تفاوت متغیر های عمومی استاتیک با متغیر های عمومی غیر استاتیک است .

کلاس حافظه ثبات :register

کلاس حافظه ثبات متغیر هایی که دارای کلاس حافظه ثبات هستند در صورت امکان در یکی از ثبات های CPU قرار می گیرند . لذا سرعت انجام عملیات با آنها بسیار بالا است و موجب افزایش سرعت اجرای برنامه می شود . کلیه متغیرهای اتوماتیک و آرگومان های توابع می توانند دارای کلاس حافظه ثبات باشند . در مورد کلاس حافظه ثبات محدودیتهای زیر را داریم: ۱از آنجایی که تعداد ثباتهای CPU محدود است ، تعداد کمی از متغیرها را

می توان با کلاس حافظه ثبات تعریف کرد . اگر تعداد متغیرهای با کلاس حافظه ثبات زیاد باشد کامپایلر کلاس حافظه ثبات را از متغیرها حذف می کند . ۲انواع متغیرهایی که میتوانند دارای کلاس حافظه ثبات باشند، درکامپیوترهای مختلف متفاوت است و معمولا" نوع های int ، char،و pointer و می توانند دارای کلاس

حافظه ثبات باشند . ۳آدرس متغیری که دارای کلاس حافظه ثبات باشد، عملا" مفهوم نخواهد داشت و یا

به عبارت دیگر، عملگر & را نمیتوان برای متغیرهای با کلاس حافظه ثبات بکار گرفت) چون این متغیرها در ثبات CPU قرار دارند.

مبحث آرایه ها و رشته ها

(بسیاری از مباحث داخل کلاس مثل برنامه ها یا الگوریتم ها بنا به بعضی ملاحظات در این جزوه نخواهد بود و لذا از بابت آن عذر خواهی بنده را بپذیرید.)

قبلا گفته بودیم یک برنامه نویس لازم است بسته به نوع الگوریتم ساختمان داده های مناسبی را برای برنامه تدارک ببیند.درس این جلسه به ساختمان داده آرایه می پردازداین ساختمان داده در واقع نامی است برای چند متغیر همنوع .یا بعبارتی نامی است برای چند کمیت همنوع که هر کدام از این کمیتها عناصر آرایه ما خواهند بود این عناصر از روی اندیس اشان قابل ذخیره و بازیابی خواهند بود.یعنی برای دسترسی به عناصر ارایه کافی است نام آرایه و اندیس آن عنصر مربوطه را وارد نماییم.برای نمونه اگر کل نمرات یک دانشجو را تحت یک نام بخواهیم داشته باشیم از آنجا که نوع این نمرات همه یکسان و مثلا از نوع float می باشند لذا ساختمان داده آرایه را انتخاب می کنیم بعدا در ادامه دروس خواهیم دید که می توان چند متغیر یا کمیت غیر هم نوع را نیز با یک نام در برنامه داشت و بموقع آنرا فراخوانی نمود.یا بر روی آن عملیات ذخیره و بازیابی و ویرایش را داشت .

برای استفاده از آرایه ها باید ابتدا آنرا در برنامه مثل متغیر های دیگر تعریف نمود تعریف آرایه به اینصورت است:int array[20]; در اینجا ۲۰ عنصر همنوع می توان در این ساختمان داده آرایه ذخیره نمود که نوع داده همه آنها باید الزاما از نوع عدد صحیح باشد .مشخصه هر

آرایه نام و اندیس آن می باشد.نام آرایه از دید کامپایلر اشاره گری است به اولین عنصر آرایه یا بعبارتی نام آرایه به اندیس صفر آرایه اشاره می کند.بنابر این نام آرایه برای کامپایلر خود گویای همه چیز است . اندیس آرایه نیز موقعیت مکانی آرایه را نشان می دهد برای نمونه array[2] از لحاظ موقعیت مکانی بعد از array[1] واقع می شود.و قبل از array[3].

لازم به توضیح است عناصر یا اندیسهای آرایه به صورت سطری و در خانه های متوالی حافظه جای می گیرند.اندیس آرایه در سی از صفر شروع می شود لذا یک آرایه ۲۰ عضوی دارای اندیسی بین صفر و ۱۹ خواهد بود.نام گذاری آرایه نیز قوانین و محدودیتهای نام گذاری و تعریف متغیر ها را دارد و در یک برنامه تا آرایه تعریف نشده باشد از نظر کامپایلر استفاده از آن بیمورد است و نمیتوان از آن استفاده نمود.

کامپایلر خطای boundary checking را نادیده می گیرد یعنی تجاوز از حدود آرایه امکان پذیر است هر چند نتیجه مطلوبی را عاید ما نکند برای نمونه در آرایه ۲۰ عضوی بالا ما ۲۰ اندیس خواهیم داشت ولی اگر در برنامه از اندیس ۲۵ آرایه array استفاده نماییم کامپایلر خطایی را بما گزارش نخواهد کرد ولی اینکار نتیجه درستی را هم عاید ما نخواهد کرد.این به ماهیت آرایه برمیگردد که در واقع آرایه یک نوع اشاره گر است .

مقدار فضایی را که برای آرایه اختصاص می دهیم میتوان ازفرمول حاصل ضرب اندازه حافظه ای که نوع ارایه اشغال می کند در تعداد عناصر آرایه مذکور بدست اورد برای مثال در آرایه int array[20] ::::::::۲۰*۲=۴۰ بایت از فضای حافظه در اشغال این ساختمان داده خواهد بود.

آرایه های ۱ بعدی و آرایه های چند بعدی:

آرایه های دارای یک اندیس را آرایه های یک بعدی و آرایه های دارای چند اندیس را آرایه های چند بعدی می نامند.

تمام عملیات تعریف , مقداردهی و استفاده از آرایه های چند بعدی عینا مثل آرایه های یک بعدی می باشد.منتها در تعریف آرایه های چند بعدی بیش از یک اندیس مطرح است .برای نمونه ; int array[20][10] باعث تعریف آرایه ای شامل ۲۰ سطر و ۱۰ ستون می باشد که کل این ۲۰۰ عنصر آرایه مذکور حتما باید از نوع عدد صحیح باشند . یعنی کامپایلر بما اجازه نخواهد داد داده های از نوع عدد اعشاری را در این آرایه قرار دهیم .

مقدار دهی اولیه به آرایه ها:

اگر بخواهیم در حین تعریف آرایه به عناصر آن مقدار اولیه نسبت دهیم شرط اینکار این است که کلاس حافظه ارایه مذکور حتما یا static و یا extern باشد در غیر اینصورت نخواهیم توانست به این آرایه مقدار اولیه بدهیم .مقادیر متناظر با اندیسها جایگزین خواهد شد واگر تعداد مقادیر کمتر از تعداد اندیسها باشد مقدار بقیه اندیسها صفر منظور خواهد شد.و اگر تعداد عناصر آرایه هنگام تعریف نامعلوم باشد کامپایلر تعداد آنها را مساوی تعداد مقادیر داده شده می پندارد برای نمونه ;static int array[]={1,2,3} باعث تعریف آرایه ای بنام array از نوع عدد صحیح با ۳ عنصر که با مقادیر ۱و۲و۳ به ترتیب مقدار دهی اولیه شده اندمی شود.برای مقدار دهی به آرایه های چند بعدی نیز چند روش وجود دارد:

```
Static int array[2][3]={1,2,3,4,5,6};
Static int array[2][3]={{1,2,3},{4,5,6}};
```

در هر دو روش ۶ عنصر آرایه array مقدار دهی اولیه شده اند.در این مورد نیز اگر تعداد مقادیر کمتر از تعداد اندیسها بود کامپایلر بقیه اندیسها را صفر جایگزین می نمود.لازم است یکبار دیگر یاد آوری شود در حافظه طریقه قرار گیری این عناصر بدینصورت خواهند بود:

```
Array[0][0]
Array[0][1]
Array[0][2]
Array[1][0]
Array[1][1]
Array[1][2]
```

این ۶ عنصر یا اندیس آرایه بصورت متوالی در خانه های متوالی حافظه و بصورت سطری ذخیره خواهند شد.

نمونه بعدی ;static int list[5]={0} در این نمونه آرایه list از نوع عدد صحیح با ۵ عضو که همه این اعضا با صفر مقداردهی اولیه شده اند.

آرایه ها بعنوان آرگومان:

از آنجا که قبلا نیز گفته شد نام آرایه اشاره گری است به اولین عنصر آرایه .تعریف اشاره گر را نیز بعدا خواهیم دید متغیری است که حاوی آدرس متغیر دیگری است بعبارت دیگر متغیر اشاره گر بجای اینکه مقدار متغیر دیگری را در خود داشته باشد آدرس متغیر دیگری را در خود دارد نام آرایه در واقع آدرس اولین عنصرآرایه را در خود دارد و بنابر این با داشتن نام یک آرایه ما همه چیز را در رابطه با آرایه در اختیار داریم.برای فراخوانی تابعی که آرایه ای را بعنوان آرگومان می پذیرد کافی است نام آرایه را داخل پرانتز فراخو.انی تابع ذکر نماییم و دیگر نیاز به اندیسهای آرایه نخواهیم داشت حتی در مورد ارایه های چند بعدی نیز برای فراخوانی تابعی که ان ارایه را بعنوان ارگومان می پذیرد کافی است نام آن ارایه را داخل پرانتز فراخوانی تابع ذکر نماییم.

در طرف مقابل هنگام تعریف تابع برای تعریف پارامترها ۳ راه داریم اول اینکه پارامتر متناظر با آرگومان آرایه را از نوع اشاره گر تعریف نماییم که این موضوع درس جلسات اتی است (هنگام تعریف اشاره گر ها)

راه دوم تعریف آرایه ای با اندیسها یا ابعاد مشخص یعنی تعیین کنیم آرایه ارسالی یک آرایه چند بعدی است و چند در چند؟

راه سوم تعریف پارامترها از نوع آرایه ای با ابعاد نامشخص

برای نمونه هنگام فراخوانی func1(array) باعث فراخوانی تابعی بنام ()func1 خواهد شد که آرگومانی بنام آرایه array خواهد داشت از آنجا که قبلا هنگام تعریف متغیر ها آرایه array را برای کامپایلر تعریف نموده ایم مشکلی پیش نخواهد امد.و کامپایلر آنرا خواهد شناخت چون سابقه ذهنی از آن دارد .

اگر آرایه ای که بعنوان آرگومان ارسال نموده ایم از نوع چند بعدی باشدکافی است هنگام تعریف پارامترهای تابع فراخوانی شده یا تما م ابعاد را درست تعیین نماییم یا پارامتر را از نوع اشاره گر تعریف نماییم و یا حتما از بعد دوم به بعد را برای کامپایلر مشخص نماییم در غیر اینصورت با خطا مواجه خواهیم شد.

رشته ها :

در زبان سی ما datatype ازنوع استرینگ یا رشته نداریم در عوض می توانیم با تعریف آرایه ای از کاراکتر ها با رشته ها کار کنیم char name[10] ارایه ای از نوع کاراکتر برای ذخیره سازی مثلا نام افراد .باید در نظر داشت که این آرایه در انتها توسط کامپایلر یک NULL یا '\0' را در خود دارد لذا برای ذخیره سازی اسامی حد اکثر ۹ کاراکتری می تواند مورد استفاده قرار گیرد.یعنی اگر بخواهیم برای نمونه آرایه ای برای ذخیره سازی نامهای حد اکثر ۲۰ کاراکتری در برنامه داشته باشیم کافی است char name[21] داشته باشیم زیرا سیستم بصورت خود کار آخرین اندیس را مساوی '\0' قرار می دهد.

Scanf("%s",name) مشاهده می شود که دیگر عملگر & در دستور scanf ظاهر نشده این بخاطر آن است که نام آرایه بعنوان یک اشاره گر خود حاوی آدرس آرایه است و دیگر لزومی به ذکر عملگر & نمی باشد البته اگر در جایی از برنامه به اندیس خاصی از یک آرایه احتیاج داشته باشیم باید آدرس آن عنصر را با ذکر عملگر & بیاوریم ولی آوردن کل ارایه فقط نیاز به ذکر نام آرایه دارد چون نام آرایه برای کامپایلر خود گویای همه چیز است.

توابع ورودی و خروجی رشته ها:

در زبان C توابع متعددی وجود دارند که در مورد رشته ها استفاده می شوند . اکثر این توابع برای خواندن رشته ها از صفحه کلید و یا چاپ آنها در صفحه نمایش بکار می روند .

تابع ()gets برای خواندن رشته هااز صفحه کلید بکارمیرود و بصورت زیر استفاده می شود(:متغیر رشته ای)gets

```
main(){
```

```
char fname[21] ;
printf("enter your name: " );
gets(fname );
printf("\nyour name is : %s",fname);}
```

نمونه ای از خروجی برنامه :

```
enter your name : ghavami
your name is :ghavami
```

مثال2

```
main(){
char name[21] ;
printf("enter your name family:");
scanf("%s", name );
printf("\n your name is : %s",name );}
```

نمونه ای از خروجی برنامه مثال ۲ :

```
enter your name : mohammad ghavami your name is : mohammad
```

همانطوری که از خروجی برنامه پیداست ، در پاسخ به درخواست برنامه نام و نام خانوادگی mohammad ghavamiرا وارد نموده ایم که چون بین نام و نام خانوادگی فاصله افتاده است فقط نام (mohammad) در متغیر name قرار گرفته و در خروجی چاپ شده است ولی نام خانوادگی از دست رفته است . اگر بجای scanf از تابع gets استفاده شود این مشکل پیش نخواهد آمد ، زیرا در ()gets فقط کلید enter ، جدا

کننده است .

تابع ()puts

این تابع برای انتقال یک رشته به خروجی بکار می رود و بصورت زیر استفاده می شود(عبارت رشته ای) puts

هنگام استفاده از تابعputs ، می توان از یک متغیر رشته ای و یا یک رشته (که در داخل نقل قول (") باشد) بجای عبارت رشته ای استفاده کرد.

.تفاوت ()gets با ()scanf در این است که تابع ()scanf وجه تمایز رشته ها را هر علامت blank یا tab می داند در حالی که تابع ()gets متمایز کننده رشته ها را از هم زدن enter میداند.

تفاوت ()puts با تابع ()printf در این است که تابع ()puts در هر لحظه یک رشته را به خروجی می برد در حالی که با ()printf همزمان چند رشته را میتوان به خروجی برد.

ارزش دهی اولیه به رشته ها نیز می تواند مثل ارزش دهی به آرایه ها انجام پذیرد:char name[]="alireza" و یا char name[]={'a','l','i','r','e','z','a','\0'} در روش دوم دیده میشود که انتهای رشته باید با کاراکتر NULL که از نظر عددی برابر با '\0' می باشد تکمیل گردد در حالی که در روش اول اینکار توسط خود کامپایلر انجام می شود.یعنی در انتهای alireza یک '\0' توسط کامپایلر قرار میگیرد.

آرایه ای از رشته ها

تاکنون یاد گرفتیم در هر مرحله آرایه را چطور با نام افراد پر کنیم دفعه بعدبا وارد کردن نام جدید نام قبلی پاک شده و نام جدید در آرایه ما جایگزین خواهد شد .گاهی ممکن است ، به عنوان مثال بخواهیم تعدادی از نام افراد را در آرایه ای نگهداری کنیم : ولی با اطلاعاتی که تاکنون از رشته ها داریم شاید نتوانیم چنین کاری را انجام دهیم . برای این کار باید آرایه ای از رشته ها داشته باشیم . چون رشته ها خود ازنوع آرایه هستند باید آرایه ای ازآرایه ها را تعریف کنیم (آرایه ای از آرایه ها همان آرایه دوبعدی است)

مثال : برنامه ای که با تعریف آرایه ای از رشته ها ، ۵ نام را در آن قرار می دهد و سپس با خواندن نامی از ورودی آنرا در آرایه جستجو می کند.

```
main(){
int dex,exist ;
int k ;
char name[21] ;
static char list [5][21]=
{ "ali" ,
"ahmad" ,
"alireza" ,
"jalal" ,
"mohammad"};
printf("\nenter one name for");
printf(" search:");
gets(name );
for(dex=0 ; dex<5;dex++)
if(strcmp(list[dex],name)==0){
exist=1 ;
break ;}
if( exist==1){
printf("\nname ");
printf("<%s> exist in table."/name);}
else{
printf("\nname ");
printf("<%s> not exist. "/name);
}}
```

بعضی از توابع رشته ای:

strcpy(name,"alireza"); رشته alireza را در متغیر رشته ای name انتساب میدهد این بخاطر آن است که در سی نمیتوان مستقیما نوشت name="alireza"

strcat(s1,s2); رشته s2 را در ادامه رشته s1 کپی میکندو با اینکار رشته فعلی s1 مجموع دو رشته می شود.

Strlen(s); طول رشته s را برمیگرداند که مثلا چند کاراکتر است.

Strcmp(s1,s2); دو رشته s1,s2 را با هم مقایسه میکند اگر دو رشته مساوی بودند عدد صفر اگر s1>s2 عدد مثبت و در غیر اینصورت عدد منفی برمیگرداند.

ضمیمه کردن فایل ها (file inclusion)

در زبان C می توان در حین ترجمه (compile) برنامه ، یک یا چند فایل را به آن ضمیمه کرده و مورد پردازش قرار داد. به فایلهایی که بدین طریق به برنامه ضمیمه میشوند، header file گفته میشود. ضمیمه کردن فایل ها، توسط دستور پیش پردازنده#include انجام می شود .این دستور بصورت های زیر مورد استفاده قرار می گیرد" :

#include "filename.h"
#include<filename.h>

فایلهای header می توانند به دو دسته تقسیم شوند :

الف) فایلهایی هستند که همراه کامپایلر C وجود دارند و انشعاب همه آنها h.

است .

ب) فایلهایی هستند که توسط استفاده کننده نوشته می شوند .

از روش اول استفاده دستور include # برای ضمیمه فایلهایی استفاده می شود که توسط استفاده کننده نوشته شده اند و از روش دوم ، برای ضمیمه فایل هایی به کار گرفته می شود که همراه کامپایلر C وجود داشته و معمولا" در فهرستی بنام include قرار دارند .

فایلهای header از اهمیت ویژه ای برخوردارند زیرا : ۱بسیاری از توابع مثل ()putchar ()getchar در فایل های header مربوط به سیستم ، بصورت ماکرو تعریف شده اند . ۲با فایل های header معمولی (مربوط به استفاده کننده (علاوه بر تعریف ماکروها ، می توان از بسیاری از تعاریف تکراری جلوگیری کرد .

مثال ۱: دستور زیر ، موجب ضمیمه شدن فایل stdio.h به برنامه می گردد <include<stdio.h> #

فرض کنید کار ما طوری است که همواره برنامه هایی مینویسیم که نیاز به محاسبه مساحت های اشکال مختلفی مثل دایره ، بیضی و غیره دارد . بهتر است برای سهولت کار ، مساحت کلیه اشکال را بصورت ماکرو تعریف کرده و در یک فایل header قرار دهیم . سپس در ابتدای برنامه مورد نظر، آن را توسط دستور include # معرفی کنیم.

تا به برنامه ضمیمه شود . نام فایل header را که برای این منظور نوشته می شود ar.h انتخاب کرده ، محتویات آن را می توان بصورت زیر نوشت

```
#define PI              3.14159
#define AREA_CITCLE(radius )       ( PI*radius*radius)
#define AREA_SQUARE(length,width )          ( length*width)
#define AREA_TRIANGLE(base,height )          ( base*height/2)
#define AREA_ELLIPS(radius1,radius2 )       ( PI*radius1*radius2)
#define AREA_TRAPEZOID(height,side1, side2)        (heigth*(side1+side2)/2)
```

پس از تشکیل فایل ar.h می توان با ضمیمه نمودن این فایل به هر برنامه ای که نیاز به محاسبه مساحت اشکال هندسی ذکر شده در این فایل را دارد ، از ماکروهای تعریف شده در این فایل استفاده کرد .

مثال ۲ :برنامه ای که با استفاده از فایل ar.h مساحت مثلثی را محاسبه میکند .

```
#include<ar.h>
main(){
int bas,height;
float s;
base=10;
height=15;
s=AREA-TRIANGLE(base,hight);
printf("%f",s);}
```

(بعضی از مباحث داخل کلاس مثل برنامه ها یا الگوریتم ها بنا به بعضی ملاحظات مثل کمی وقت و یا الزام شما به شرکت در کلاس در این جزوه نخواهد بود و لذا از بابت آن عذر خواهی بنده را بپذیرید.)

اشاره گر ها :pointers

سهولت کار با رشته ها و آرایه ها و دیگر ساختمانهای داده در سی - همین طور انعطاف پذیری برنامه و اختصاص حافظه پویا از محاسن استفاده از اشاره گرها در سی می باشد.برای نمونه با اشاره گر ها میتوان آرایه ها را بصورت آرایه با ابعاد کمتر مورد استفاده قرار داد.وقی در برنامه از آرایه ای از اشاره گر ها کمک میگیرید به مدیریت و صرفه جویی در حافظه کمک میشود بعدا با ذکر مثالهایی به این مطلب خواهید رسید.و یا اینکه چند مقدار را بصورت رفت و برگشتی بین تابع فراخواننده و تابع فراخوانی شده انتقال داد .مثلا وقتی با اشاره گر ها آرایه ای را به تابعی می فرستیم و عملیات خاصی بر روی ان آرایه انجام می شود با انتقال به تابع فراخواننده کل ان عملیات بر روی تمام اجزای آرایه تاثیر کرده و مثل این است که همزمان چند مقدار را به تابع فراخواننده منتقل نموده باشیم. ضمن اینکه با اشاره گر ها میتوانیم بجای اینکه از ابتدای برنامه حافظه ای را برای ساختمان داده هایمان در برنامه ذخیره کرده و این حافظه را تا اتمام اجرای برنامه در اشغال خود داشته باشیم با اشاره گر ها و توابع تخصیص حافظه پویا میتوان براحتی در زمان اجرا حافظه را از سیستم گرفته و بعد از پایان عملیات اجرا ان را برگردانیم اینکار به صرفه جویی در حافظه مصرفی برنامه ها که یکی از پارامتر های مهم برنامه نویسی است کمک شایانی می کند.از آنجا که تمام متغیرهای برنامه با توجه به نوعشان در حافظه جا اشغال می کنند و این جا طبعا آدرسی از آدرسهای حافظه می باشد لذا

متغیری که بتواند این آدرسها را در خود نگه دارد متغیر از نوع اشاره گر نامیده میشود.یعنی اشاره گر هم همان متغیر است منتها متغیری که بجای مقدار . آدرس در ان قرار گرفته و بخاطر همین تمام عملیاتی که بر روی متغیر ها قابل انجام است بر روی اشاره گر ها قابل انجام نیست .برای مثال اگر ; int x=10 ; px=&x * int بااینکار آدرس متغیر x را در متغیر اشاره گر px قرار داده ایم و از این به بعد متغیر px به آدرس x اشاره خواهد کرد البته میتوان قبل از اینکار نیز اشاره گر px را تعریف نموده و بعد به آن مقدار داد. px* int باعث تعریف اشاره گر px خواهد شد که در طول برنامه باید به متغیر های از نوع صحیح اشاره کند در هنگام تعریف اشاره گر ها اگر نوع اشاره گر با نوع داده ای که قرار است به آن اشاره کند دو نوع مختلف باشد حتی اگر کامپایلر خطایی را اعلام نکند که میکند باز هم روند اجرای برنامه روند درستی نخواهد بود.

برای مثال :int *p; Float f; p=&f; که باعث بروز error خواهد شد قبلا در بحث عملگر ها اشاره شد که عملگر یکانی & آدرس متغیر ها را نشان می داد ما با این عملگر می توانیم آدرس یک متغیر معمولی یا آدرس یک آرایه ساختمان و یا حتی یک اشاره گر دیگر را در اشاره گر قرار دهیم. و عملگر یکانی * معرف محتوایی از حافظه است که به آن اشاره می شود.یا اصطلاحا مقدار یک آدرس .

-بین نام آرایه و نام یک اشاره گر این تفاوت وجود دارد که نام یک اشاره گر در واقع نام یک متغیر است و میتوان با ان مثل یک متغیر عمل کرد ولی در مورد نام آرایه اینطور نیست .برای مثال :++(*ptr) مقداری را که اشاره گر ptr به ان اشاره میکند را یکی اضافه می نماید.نکته جالبی که وجود دارد این است که در مورد مثال جدید ++ptr* بمعنی این است که ابتدا در حافظه ای که ptr به آن اشاره میکند یک خانه یا دو بایت به جلو برو و انگاه مقداری که ادرس جدید به آن اشاره میکند را مورد عملیات مفروض قرار بده به خاطر این ۲ بایت جلو رفتیم چون طول نوع int در حافظه ۲ بایت است و میدانیم متغیر ها با توجه به نوعشان در حافظه جا اشغال می کنند.در مورد مثال گفته شده آدرس جدید الزاما به محتوایی اشاره نخواهد کرد.یعنی ممکن است به خانه ای از حافظه اشاره کند که هیچ مقداری ندارد و یا نوع مقدار آن خانه با نوع اشاره گر همخوانی ندارد.

وقتی اسم یک آرایه به یک تابع بعنوان ارگومان ارسال می شود در واقع ادرس اولین عنصر ان ارایه به تابع ارسال شده یا بعبارتی نام یک آرایه اشاره گری است به اولین عنصر ان ارایه زیرا ادرس اولین عنصر ارایه را در خود دارد. بنابراین اگر بخواهیم یک آرایه را بعنوان ارگومان به تابعی بفرستیم کافی است فقط نام آرایه را بعنوان آرگومان ارسال کنیم .هنگام تعریف پارامترها هم میتوان از ابعاد آرایه چشمپوشی کرد منتها اگر آرایه ما چند بعدی باشد باید از بعد دوم به بعد را هنگام تعریف پارامترها ذکر نمود.اگر بخواهیم ارایه را با اشاره گر ها به تابع ارسال نماییم کافی است اشاره گری تعریف نماییم و نام آرایه یا آدرس اولین خانه ارایه را در ان قرار دهیم در اینصورت با فرستادن اشاره گر آرایه به تابع میتوان آرایه را در تابع فراخوانی شده مورد عملیات قرار داد. برای نمونه:

```
Main(){
Int *list;
void func1(int *list,int *ptr)

Int *ptr;
Printf("how many numbers?");
Scanf("%d",&n);
Ptr=&n;
List=(int *)malloc(n*sizeof(int));

If(!list){
Printf("error in memory allocation");
Getch();
Exit(1);
}
for(i=0;i<*ptr;i++)
scanf("%d",list+i);
printf("the first array is");
for(i=0;i<*ptr;i++)
printf("%d",*(list+i)
func1(list,ptr);
printf("the convertedarray is ")
```

```
for(i=0;i<*ptr;i++)
printf("%d",*(list+i)
}
void func1(int *list,int *ptr)
{
int I;
for(i=0;i<*ptr;i++)
*(list+i)=0;
}
```

در مثال گفته شده واضح است که اشاره گر list برای آرایه تعریف شده با ارجاع ادرس این ارایه به تابع و برگشت به تابع اصلی متوجه می شویم که انگار چند مقدار به تابع اصلی برگشت داده شده و همه عناصر ارایه صفر شده اند در روش فراخوانی با ارجاع که با اشاره گر ها امکان پذیر بود هر نوع تغییری در پارامترها یا همان اشاره گر ها همان تاثیر را بر روی آرگومانهای متناظر در کل تابع فراخواننده می گذاشت.نکته دیگر اینکه (list+i)* در واقع همان list[i] است که مقدار عنصر iام آرایه می باشد و یا list+I معادل همان &list[i] می باشد که به آدرس list[i] برمیگردد.بنابراین بین آرایه ها و اشاره گر ها ارتباط تنگاتنگی وجود دارد.برای نمونه بعدی : list[i][j]معادل(j+(list+i))* می باشد و یا &list[i][j] معادل j+(list+i) میباشد این نمونه در مورد آرایه های دو بعدی بود و همین طور الی اخر...

بعضی نکات کار با اشاره گر ها:

p**یعنی خود p حاوی آدرس یک ا شاره گر دیگر است که انهم آدرس یک متغیر دیگر را در خود دارد.یعنی اشاره گر به اشاره گر در واقع خانه ای از حافظه است که آدرس یک اشاره گر دیگر را در خود دارد.

تعریف آرگومانها در function prototyping یا هنگام معرفی تابع به main لازم نیست حتما با آنچه در تعریف خود تابع آورده می شود یکسان باشد.برای مثال میتوان هنگام prototyping اینگونه اقدام کرد(void rev(int *k,int *t ولی در هنگام تعریف تابع rev میتوانیم اینگونه اقدام کرد:(void rev (int *x, int *y این دو هیچ منافاتی با هم ندارند.

هنگام صدا زدن یک تابع لازم نیست نوع آنرا ذکر کنیم .

(p*)+1 با (p+1)* با هم خیلی فرق دارند اصولا کار با اشاره گر ها ظرافت خاصی میطلبد.

اگر نوع یک اشاره گر از نوع مثلا int باشد حتما باید به داده هایی از نوع int اشاره کند.

اگر بخواهیم از اشاره گر ها استفاده درستی بکنیم ابتدا باید آدرس خاصی را که میتواند متغیر های معمولی –ساختمان و.... باشد را در آن قرار دهیم و بعد اشاره گر را مورد استفاده قرار دهیم اگر آدرسی در اشاره گر قرار نگیرد ولی از اشاره گر استفاده شود اینکار باعث خطاهای بعدی در برنامه می شود.

اعمال بر روی اشاره گر ها بوسعت اعمال روی متغیر های دیگر نیست ولی میتوان گفت حتی با همین تعداد عملیات محدود انعطاف پذیری برنامه با اشاره گر ها بسیار بالاست.-۱ عمل انتساب ۲- جمع و تفریق و ۳ مقایسه اشاره گر ها مشروط به اینکه هر دو اشاره گر به داده یک نوعی اشاره کنند .;pv=0;* یعنی محتوایی از حافظه را که pv به آن اشاره دارد را صفر بگذار.Int *p1,*p2; p1=&x;p2=p1 باعث میشود آدرس x هم در p1 وهم در p2 قرار داشته باشد.

لازم بذکر است متغیرهای اشاره گر میتوانند هم آدرس متغیر های معمولی و هم آ درس آرایه ها –توابع و یا حتی اشاره گر های دیگر را هم در خود داشته باشند.

اگر یک اشاره گر به داده های از نوع int اشاره کند با افزودن یک واحد به آن به دو بایت بعدی حافظه اشاره خواهد کرد چون در حافظه هر int دو بایت فضا میگیرد. int *p ++p .

اشاره گر ها و توابع :

قبلا گفتیم در نوع فراخوانی call by value کپی از آرگومانها به تابع فراخوانده شده ارسال میشد و لذا هر تغییری در پارامترهای تابع فراخوانده شده هیچ گونه تاثیری در آرگومانهای تابع فراخواننده یا اصطلاحا آرگومانهای متناظر با پارامترها نداشت . ولی در روش

فراخوانی call by refrence آدرس ارگومانها به جای کپی مقدارشان در پارامترهای تابع فراخوانده شده قرار میگیرد و لذا پارامترهای تابع فراخوانده شده به جای مقدار. آدرس آرگومانهای تابع فراخواننده را در خود دارد و لذا هر نوع تغییری در پارامترها باعث ایجاد تغییر در آرگومانهای تابع اصلی می شود. اینکار تنها با بهره گیری از اشاره گر ها مقدور است.یعنی آرگومانها باید از نوع آدرس بوده و پارامتر ها باید از نوع اشاره گر باشند.برای مثال برنامه ای آورده شده که دو عدد را گرفته و مقدارشان را با هم عوض می کند.

برای تمرین شما اینکار را برای ارایه های عددی انجام دهید.؟؟؟

همین طور که پیداست ادرس دو متغیر int به عنوان اشاره گر به تابع فراخوانده شده ارسال شده اند و در پارامترهای متناظر قرار گرفته اند تغییر روی پارامترها بمنزله تغییر روی آرگومانها ی تابع اصلی می باشد.

```
main()
{
void convert(*int p,*int q);
int x,y;
printf("\n enter 2 number to be reversed");
scanf("%d%d",&x,&y);
convert(&x,&y);
printf("the converted numbers are%d%d",x,y)
}
void convert( int *p,int *q)
{
int temp ;
temp=*p;
*p=*q;
*q=temp;
}
```

برنامه ای که با روش call by refrence و اشاره گر ها ۳ عدد از ورودی خوانده و بزرگترین انهارا پیدا کرده و چاپ نماید:

```
Main()
{
void max (int *p,int *q, int *w);
int x,y,z;
scanf("%d%d%d",&x,&y,&z);
 printf("the max is %d",max(&x,&y,&z));
}
max(int *p,int *q, int *w)
{
if ((*p)>(*q ))
{if ((*p)>(*w))
return(*p);
     else if ((*p)<(*w)) return(*w);}
            else if  *q>*w return(*q);
                  else return(*w)
}
```

حتی توابع هم می توانند در سی توسط اشاره گر ها منتقل شوند یعنی آدرس تابع را که همان نام تابع باشدبعنوان نقطه ورود به تابع را بعنوان آرگومان به اشاره گر منتقل میکنیم .

اشاره گر ها میتوانند اطلاعات را بصورت رفت و برگشتی بین تابع و نقطه رجوع به ان منتقل کنند ضمن اینکه آرایه ها و اشاره گر ها ارتباط تنگاتنگی با هم دارند زیرا اشاره گر ها یک روش دسترسی به عناصر مجزای آرایه را فراهم میکنند ضمن اینکه آرایه های چند بعدی را میتوان بوسیله اشاره گر طوری تعریف کرد تا مجموعه ای از رشته ها درون یک آرایه تک بعدی نشان داده شوند حتی اگر این رشته ها از لحاظ طولی متفاوت باشند.

اشاره گر ها میتوانند مشروط به اینکه به یک نوع داده اشاره کنند با هم مقایسه شوند اینکار معمولا در لینک لیستها و استک کاربرد دارد.

یک متغیر اشاره گر میتواند در متغیر اشاره گر دیگر قرار بگیرد.حتی میتوان مقدار صفر را در یک متغیر اشاره گر قرار داد:;pv=px;px=0

متغیر های اشاره گر هم مثل بقیه انواع متغیر ها باید برای اینکه در سی از آنها استفاده کنیم باید انها را تعریف نمودقبل از استفاده.: float *pv=&c

جایگزینی مقدار صحیح در متغیر اشاره گر مفهومی ندارد اما جایگزینی صفر در اشاره گرها استثناست برای بررسی بعضی شرایط خاص .اگر احیانا مقدار صحیحی در اشاره گر قرار بگیرد یعنی بجای آدرس .مقدار صحیح در آن قرار دهیم کامپایلر خطای cant conver int to int* را صادر خواهد کرد در چنین مواقعی باید در مقدار دهی به متغیرهای اشاره گر تجدید نظر کرد.

مثالی در رابطه با فراخوانی یک آرایه در یک تابع با کمک اشاره گر ها: برنامه ای که تعداد کاراکتر ها و .....را در یک عبارت می شمارد. بعنوان تمرین شما اینکار را برای حالتی انجام دهید که آرایه نیز بکمک اشاره گر ها ارسال می شود.

```
Main()
{
void accounting(char name[80],int *nl,int *nw,int *nc,int *nb);
int nl=0;int nw=0;int nc=0;int nb=0;
char name[80];
for(count=0;count<80;count++)
while(name[count]=getchar()!=EOF)
name[count]=getchar();
accounting(name,&nl,&nw,&nc,&nb);
printf(the num of line is%d the num of word is%d the num of character s%d the num of blank is%d,nl,nw,nc,nb)
}
void accounting(char name[],int *nl,int *nw,int *nc,int *nb)
int count;
for(count=0;count<80;count++)
while(name[count]=getchar()!=EOF)
{if ((name[count]>='a') &&(name[count]<='z'))|| ((name[count]>='A') &&(name[count]<='Z'))
*nc++;
else if ((name[count]=='\b') ||(name[count]<='\t'))
*nb++;
else if (name[count]=='.' ||(name[count]<='\t'|| name[count]=='\n')
*nw++;
}}
```

نکاتی در رابطه با استفاده از اشاره گر ها در برنامه:

- ✓ نمی توان به متغیر های از نوع اشاره گر یک مقدار صحیح را نسبت داد ولی در مورد صفر اینگونه نیست :;ptr=15 یک مقداردهی نادرست است هر چند که ;ptr =15* یک مقدار دهی درست بحساب می آید.

- ✓ آرایه از نوع عددی چنانچه بخواهد مقداردهی اولیه شوداعلان معمولی می طلبد و لی اگر لازم نباشد مقدار دهی اولیه داشته باشیم می توان با اشاره گر ها آنرا تعریف نماییم هر چند در مورد متغیر های اشاره گر از نوع char مساله فرق میکند و میتوان انرا مقدار دهی اولیه کرد یعنی متغیر اشاره گر کاراکتری را میتوان با یک رشته کاراکتری مقدار دهی اولیه کرد char x[]="this is a test"=========char *x="this is a test" خلاصه مطلب اینکه آرایه از نوع اشاره گر نمیتواند مقدار اولیه بگیرد.برای تعریف آرایه از نوع اشاره گر:

- ✓ Int *array1; برای آرایه عددی یک بعدی

- ✓ Int (*array2)[10]; برای آرایه عددی دو بعدی که بعد دوم در اینجا باید ذکر شود .

- ✓ Int (*array3[10][20]); برای آرایه عددی سه بعدی که از بعد دوم به بعد در اینجا ذکر شده است.

- ✓ قبلا گفته بودیم که آرایه ها تنها در صورتی امکان مقدار دهی اولیه را دارند که یا از نوع کلاس حافظه استاتیک باشند و یا از نوع گلوبال.

- ✓ همیشه در اشاره گر باید یک ادرس قرار بگیرد این ادرس میتواند ادرس یک متغیر معمولی از نوع float int char و غیره باشد و یا آدرس آرایه – تابع و یا حتی آدرس اشاره گر دیگری باشد .یا بعبارتی میتوان در یک اشاره گر آدرس یک اشاره گر دیگر را قرار داد که خود اشاره گر دوم به یک مقدار در حافظه اشاره میکند.بعدا خواهیم دید که اشاره گر ها می توانند ساختارهای داده دیگری را هم پشتیبانی نمایند.

- ✓ قرار دادن علامت & در سمت چپ عبارات محاسباتی کار درستی نیست.
- ✓ مقایسه اشاره گر ها و انتساب آنها بهم در صورتی امکان پذیر است که هر دو به یک نوع داده اشاره نمایند.

- ✓ اشاره گر ها را نمیتوان در هم ضرب و یا بر هم تقسیم نمود اما در بعضی شرایط بنا به ملاحظاتی میتوان مقدار صحیحی را از یک متغیر اشاره گر کم کرد و یا مقدار ی را به آن اضافه نمود.برای مثال ptr+2 در صورتی که ptr اشاره گری از نوع int باشد ۴ بایت مارا جلوتر خواهد برد که ادرس جدید لزوما به جایی اشاره نخواهد داشت.

- ✓ اسم آرایه دقیقا اشاره گری به اولین عنصر یا ایندکس آرایه است لذا دو دستور ذیل عادلند:x[i]=*(x+i)
- ✓ چنانچه آرایه عددی را از نوع اشاره گر تعریف کرده باشیم در هیچ شرایطی مقدار اولیه نخواهد پذیرفت.

- ✓ چنانچه قبلا گفتیم تعریف مرسوم آرایه منجر به اخذ یک بلوک ثابت حافظه میشود با آدرس ابتدا و انتهای مشخص .که تا زمان پایان برنامه در اختیار ارایه خواهد بود.ولی گاها احتیاج میشود ارایه ای از نوع داینامیک داشته باشیم که در حین اجرای برنامه برای آن حافظه اخذ نموده و هر موقع صلاح دانستیم حافظه اختصاص یافته را آزاد نماییم به اینکار اختصاص داینامیکی حافظه گفته می شود که صرفا با اشاره گر ها امکان پذیر است یعنی اگر آرایه ای از نوع اشاره گر داشته باشیم میتوان با تابع کتابخانه ای malloc حافظه لازم را برای آن اخذ نموده و با تابع realloc چنانچه بخواهیم این مقدار حافظه را تغییرمی دهیم یعنی کم یا زیاد نماییم و نهایتا هر جا لازم بدانیم با تابع کتابخا نه ای free() حافظه پویای تخصیص یافته را میتوان ازاد کرد .میتوان از جزییات استفاده از این توابع با ctrl+f1 روی نام این توابع آگاه شد برای مثال از نام فایل هدر مربوطه اطلاع پیدا کرد.مثال مربوطه برای اخذ حافظه پویا در بالا آورده شده.
- ✓ (روش تخصیص حافظه پویا)

- ✓ فرض نمایید int *x آرایه از نوع int باشد یعنی به داده های int اشاره نماید .جالب است بدانیم در این مورد یک بلوک ثابت حافظه به طور خودکار به x تعلق نمیگیرد اگر چه در تعریفات سابق int x[10] برای مثال ۱۰ خانه حافظه متوالی برای این آرایه اشغال میشد.

- ✓ ولی در اینجا باید اینگونه عمل کرد:x=(int *)malloc(10*sizeof(int)); بهتراست بدانیم که تابع malloc یک اشاره گر بر میگرداند که از نوع int بوده و در x می نشیند حالا میتوان با یک دستور for ساده عناصر آرایه را خواند و در آدرسها قرار داد:for(i=0;i<10;i++) scanf("%d",x+i);

- ✓ یک متغیر اشاره گر میتواند از متغیر اشاره گر دیگر کم شودبه شرط آنکه هر دو به یک آرایه اشاره کنند این در حالی است که دو متغیر اشاره گر نمیتوانند با هم جمع شوند.

- ✓
- ✓ برای اعلان آرایه های دوبعدی بصورت اشاره گر کافی است از بعد دوم به بعد را ذکر نماییم برای تعریف آرایه های دو بعدی اینگونه اقدام می نماییم int (*x)[20] به جای int x[10][20]
- ✓ که در آن x+3 به چهارمین آرایه ۲۰ عنصری اشاره دارد.
- ✓
- ✓ اگر بخواهیم آرایه ای از اشاره گر ها داشته باشیم باید پرانتز مذکور را حذف نماییم و ابعاد از ۱ تا n-1 آرایه را با براکت ذکر نماییم . int *x[10] برای نمایش آرایه ای از اشاره گر ها مورد استفاده قرار میگیرد.که در ان x[0] به شروع اولین سطر اشاره داردو.....برای تمرین شما با استفاده از این روش جمع دو ماتریس را با ۳ تابع یکی برای خواندن و پر کردن هر آرایه و یکی برای محاسبه حاصل جمع و یکی برای چاپ آرایه سوم یا در واقع جواب انجام دهید.

آرایه هایی از اشاره گر ها :

قبلا یادگرفتیم که چگونه ارایه ها ی یک یا چند بعدی را بصورت ساده یا با استفاده از اشار ه گرها در برنامه بکار برده و یا به عنوان آرگومان انها را به توابع بفرستیم ضمنا فرا گرفتیم که چون آرایه شامل متغیر های از یک نوع می باشد لذا هر اندیس آن مقدار فضای ثابت و مساوی با اندیس های دیگر در حافظه اشغال میکند لذا میتوان نتیجه گرفت که با داشتن نام آرایه مثل این است که کامپایلر همه چیز را در رابطه با آرایه ها می داند و بخاطر همین در هنگام ارسال آرایه ها به توابع بعنوان آرگومان چه آرایه یک بعدی باشد یا چند بعدی فقط نام آرایه را در پرانتز تابع ذکر می نماییم اینکار باعث می شد که کل آرایه به تابع ما ارسال شود. البته مشروط به اینکه ما می خواستیم کل آرایه را به تابع منتقل نماییم .

در این درس میخواهیم یاد بگیریم چگونه یک ارایه چند بعدی را به جای اشاره گری به گروهی از آرایه های پیوسته برحسب آرایه ای از اشاره گر ها بیان کنیم .در این صورت لازم است از بعد یک تا بعد n-1 را همراه با نام و ستاره اشاره گر ذکر نماییم برای مثال int *arr1[10] می تواند اعلانی برای آرایه ای از اشاره گر باشد ابعاد آرایه می تواند برای مثال ۴۰*۱۰ باشد . که در اعلان باید بعد اول اورده شود..در این صورت هر اشاره گر به شروع یک ارایه n-1 بعدی مجزا اشاره خواهد داشت . یعنی ۱۰ اشاره گر موجود هر کدام به شروع یک سطر اشاره خواهند داشت.

Int *arr1[بعد اول][بعد دوم][...][...][بعد n-1];

توجه داشته باشید که اسم آرایه دیگر داخل پرانتز قرار ندارد و این بدان معنی است که arr1[0] به شروع اولین سطر اشاره دارد و arr1[1] به شروع دومین سطر اشاره میکند.ضمن اینکه تعداد عناصر داخل هر سطر به وضوح تعیین نشده است . وبنابر این عنصر arr1[2][5] میتواند به این صورت دستیابی شود (arr[2]+5)*که ایندو مقدار معادل با هم هستند.و به همین صورت اگر بخواهیم آرایه سه بعدی را با تکنیک آرایه از اشاره گر ها تعریف کنیم به اینصورت عمل می کنیم int arr2[20][30][40]=========int *arr2[20][30] این دو عبارت با هم می توانند معادل باشند.و لذا اگر بخواهیم به عنصر arr2[2][5][4] دستیابی پیدا کنیم باید :به آدرس (arr2[2][5]+4) رفته و مقدار این آدرس را در عملیات خود شرکت دهیم یعنی arr2[2][5][4]==*(arr2[2][5]+4) بعنوان نمونه مساله جمع دو ماتریس با استفاده از آرایه ای از اشاره گر ها پرینت گرفته شده که میتوان از آن الگو برداری نمود.

آرایه های اشاره گر روش خوبی برای کار با رشته ها ارایه می دهنددر اینصورت هر اشاره گر موجود به شروع یک رشته اشاره خواهد داشت.نحوه تخصیص حافظه پویا برای آرایه از اشاره گر ها را در ذیل مشاهده می کنید.

```
static Char *name[10]={"reza","ali","ahmad","taghi","susan"."anita","Diana","alireza"."javad","faramarz"};
For(i=0;i<10;i++)
```

Name[i]=(char *)malloc(12*sizeof(char));

Struct **یا ساختمان**

تاکنون یاد گرفتیم چگونه یک قلم داده را در یک متغیر در حافظه جای دهیم ضمن اینکه یاد گرفتیم چگونه چند قلم داده از یک نوع را در ساختمان داده ای بنام آرایه میتوان قرار داد و از ان در برنامه استفاده نمود اما اگر بخواهیم تعدادی قلم داده ای را که از یک نوع نیستند با هم و تحت یک نام در حافظه ذخیره کنیم تکلیف چیست و چه باید کرد در اینجا ساختمان داده ای بنام struct یا ساختمان جوابگوی مشکل ماست با این نوع داده میتوان اقلام مختلفی از لحاظ نوع را در حافظه تحت یک نام ذخیره کرد برای مثال میتوان مشخصات یک کارمند از قبل نام شماره کد ملی تعداد اولاد و حقوق ثابت ماهیانه همین طور نام و نام خانوادگی او را در ساختمان قرار داد همان طور که می دانید هر کدام از این قلم داده ها از یک نوع خاص متفاوت با دیگری می باشد یکی int و دیگری char و غیره .

این ساختار مثل رکورد در پاسکال می ماند که هر رکورد از تعدادی فیلد تشکیل می شد در اینجا نیز صرف تعیین یا تعریف نوع داده struct هیچ مشکلی را برطرف نمیکند بلکه باید متغیر هایی از نوع struct در برنامه تعریف گردد تا بتوان از آنها در فرصت مقتضی بهره گرفت .همانطور که صرف تعریف int یا float در برنامه کافی نبود و می بایست متغیر هایی از این نوع در برنامه تعریف می شد. تا بتوانیم از آنها در برنامه استفاده کنیم

اجزا یا عناصر یا فیلدهای struct میتوانند متغیر های معمولی یا آرایه ها و یا حتی خود struct دیگری باشند همین طور union که بعدا در مورد ان صحبت خواهیم کرد.برای مثال :

```
 Struct  employee{
char name[30] ,family[80], address[20];
int  age;
char sex;
Float tax;
}emp1,emp2;
```

در مثال نوعی ذکر شده دو متغیر emp1 ,emp2 متغیر هایی از نوع struct هستند که هر کدام فیلدهای نام- فامیلی و سن و... دارند.

روش تعریف متغیر های از نوع ساختمان به دو صورت امکان پذیر است :

بهتر است بیرون از ()main ساختمان تعریف شود و در برنامه هر جا لازم شد متغیر های از نوع ساختمان تعریف گردد.با دستور روبرو:

struct employee emp1,emp2; توجه داشته باشید که نامگذاری ساختمان از قوانین نام گذاری متغیر ها پیروی می کند. تا اینجا یاد گرفتیم که struct نامی است برای مجموعه از داده ها که میتوانند از یک نوع نباشند.چنانچه تعداد متغیر های از نوع ساختمان بیش از یکی باشد باید آنها را با کاما از هم جدا نمود.:(ناگفته نماند که اجزای رکورد یا عناصر ساختمان می توانند از یک نوع باشند.)

struct employee emp1,emp2,emp3,emp4; در اینجا ۴ متغیر از نوع ساخنمان تعریف می شود برای کامپایلر.

روش دوم به این صورت است که بلافاصله بعد از تعریف struct متغیر های از نوع ساختمان را بعد از تعریف ذکر نمود.

```
struct employee{
int code;
float tax;
char name[40];} emp1,emp2,emp3,emp4;
```

روش دسترسی به عناصر ساختمان اینگونه است که نام عنصر مر بوطه را با یک نقطه بعد از نام متغیر ساختمان ذکر می کنیم

```
 printf("%Yd",emp1.grade);

  Scanf("%s",emp1.name);
   Scanf("%d",&emp2.code);
For(i=0;i<10;i++)
Printf("%8.3f",emp4.tax)
Strcpy(emp1.name,"alireza");
```

آخرین مورد مثال باعث کپی شدن رشته علیرضا در فیلد نام از متغیر ساختمانی emp1 می شود.

قبلا گفتیم یک آرایه در حالت معمولی فقط چنانچه از نوع کلاس حافظه استاتیک یا گلوبال باشد می تواند مقدار اولیه بگیرد همین آرایه اگر با اشاره گر نوشته میشد اگر از نوع عددی می بود در هیچ حالتی نمیتوانست مقدار اولیه بگیرد ولی اگر آرایه کاراکتری بود می شد

حتی اگر از نوع استاتیک هم نبود رشته ای را بعنوان مقدار اولیه در آن گذاشت حالا در بحث ساختمان اگر یک ساختمان از نوع کلاس حافظه استاتیک یا گلوبال باشد می تواند مقدار اولیه بگیرد در غیر اینصورت نمی تواند مقدار اولیه بگیرد مقادیر داده شده بعنوان مقدار اولیه –بصورت نظیر به نظیر در فیلدهای ساختمان خواهند نشست یعنی مقدار اول برای فیلد اول و مقدار دوم برای فیلد دوم و.....اگر تعداد مقادیر کمتر از تعداد فیلدهای ساختمان باشد بقیه فیلدهای عددی صفر مقدار دهی خواهند شد از ناحیه کامپایلر و بقیه فیلدهای غیر عددی با مقدار null مقدار دهی خواهند شد.

برای مثال{0}=struct employee emp1 تمام فیلدهای عددی را مساوی صفر و تمام فیلدهای غیر عددی با null مقداردهی خواهند شد. ضمنا میتوان دو ساختمان را به هم منتسب کرد. Emp1=emp2 که در این حالت نیز فیلدهای emp2 متناظر با فیلدهای emp1 در این ساختمان قرار خواهند گرفت .در نگارشهای قدیمیتر سی اینکار اغلب با error روبرو می شد.

انتقال اجزا یا عناصر ساختمان به عنوان آرگومان یک تابع :

اگر بخواهیم کل یک ساختمان را بعنوان یک آرگومان به تابعی ارسال نماییم کافی است

```
func(struct employee emp1);
```

در اینصورت کل ساختمان emp1 بعنوان آرگومان ارسال خواهد شد به تابع ()func ضمن اینکه در طرف مقابل باید typechecking رعایت شده باشد یعنی در هنگام تعریف تابع ()func پارامتری شبیه به آرگومان منتقل شده از لحاظ تعداد و نوع فیلدها وجود داشته باشد. تا کامپایلر خطایی را اعلام نکند البته این هرگز به ان معنی نیست که پارامتر باید همنام با ارگومان باشد از لحاظ نام خود یا اجزایش.ولی از لحاظ نوع و تعداد اجزای دو ساختمان یکی آرگومان و دیگری پارامتر باید با هم همخوانی داشته باشند.

برای انتقال تک تک اجزای ساختمان به تابع فیلدهای ساختمان اگر از نوع متغیر معمولی باشند براحتی میتوان آنها یا آدرس انها را به تابع منتقل نمود:;(func(emp1.grade یا برای فراخوانی از نوع call by refrence میتوان اینگونه عمل کرد;(func(&emp1.grade تا آدرس فیلد مورد نظر به تابع مذکور انتقال یابد بجای انتقال کپی مقدار فیلد در پارامتر متناظر.

اما اگر فیلد ساختمان- آرایه باشد که با انتقال آرایه به تابع بخوبی آشنا هستیم اگر کل ارایه بعنوان یک فیلد از ساختمان قرار است منتقل شود نام آن کفایت می کند: (func(emp1.name ولی اگر بخواهیم تعدادی از عناصر آرایه را انتقال دهیم کافی است اندیس مذکور را قید نماییم :.(func(&emp1.name[i]

حتی اگر فیلد یک ساختمان یک ساختمان دیگر باشد : (func(emp1.st.name و الی آخر

آرایه ای از ساختمان :

تاکنون می توانستیم در یک مورد مثلا یک کارمند اطلاعات متفاوتی از او را در ساختمان و تحت یک نام در برنامه داشته باشیم تا هر جا لازم بود انرا مورد دسترسی و عملیات و یا ارسال به تابع خاصی برای پردازش موردی قرار دهیم اما اگر بخواهیم برای مثال مشخصات تعدادی کارمند را وارد نماییم کافی است آرایه ای از ساختمان تعریف کنیم : یعنی در برنامه ارایه ای تعریف نماییم که هر عنصر آن یک ساختمان باشد.

```
struct employee{
Char name[20] ,family[80];
Int code;
Float tax;
}emp[100];
```

در اینصورت میتوان برای ۱۰۰ کارمند رکوردهای آنها را با اطلاعات لازم پر کرد.

```
For(i=0;i<max;i++)
{Scanf("%d",&emp[i].code);
scanf("%s",emp[i].name);}
```

انتقال ساختمان یا اجزای آن به تابع بعنوان آرگومان

قبلا فراگرفتیم چگونه با استفاده از ساختمان با توابع کار کنیم یکبار دیگر با هم مرور می کنیم چنانچه قرار باشد کل ساختمان را بعنوان آرگومان به یک تابع ارسال نماییم کافی است مشخصات فیلدهای رکورد مربوطه و یا اجزای ساختمان را در بیرون از ()main تعریف نموده و آنگاه آنرا بعنوان آرگومان به تابع انتقال دهیم دقیقا مثل یک متغیر معمولی .حتی میتوان برای سرعت بخشی و بالا بردن انعطاف پذیری برنامه آدرس ساختمان را بعنوان ارگومان ارسال نمود در این صورت باید در طرف مقابل یا در تابع فراخوانده شده اشاره گری تعریف نمود که این ادرس در آن قرار بگیرد.به مثال توجه کنید:

```
Struct student{
Int stu;
Float grade;
Char name[40];}
main()
{
struct student s1,s2;
.....
func(s1);
....
}
func (struct student  xxx)
{
.....
}
```

در این مثال در بیرون از تابع اصلی برنامه. ساختمان با تمام اجزایش تعریف شده و در برنامه اصلی متغیر های از نوع ساختمان نامبرده تعریف شده و بعنوان ارگومان به تابع ()func فرستاده شده است در تابع فراخوانده شده ()func نیز پارامتر متناظر با آرگومان ارسالی xxx نام دارد که خود مشابه با s1 می باشد یعنی خود xxx متغیری از نوع ساختمان student می باشد.برای اینکار یک روش دیگری هم وجود داشت که توصیه نمی شود و آن اینکه در هر تابع برای خودش ساختمان ها را جداگانه تعریف نماییم البته باید توجه داشت که تعداد و نوع اجزا باید مثل هم باشند در دو ساختمان و لی نامها لزوما نه.قبلا گفته بودیم که از نظر کامپایلر مساله type checking انجام می شود و اگر خطایی در این زمینه دیده شد کامپایلر آنرا اعلام خواهد نمود. البته اینکار برنامه را طولانی تر نموده و باعث ایجاد بعضی اشکالات می گردد.پس بهتر است ابتدا در بیرون از تابع اصلی یا main ساختمان مربوطه تعریف گردد و در دوتابع فراخواننده و تابع فراخوانی شده متغیر هایی از نوع ساختمان مذکور تعریف گردند.این در حالی است که اگر می خواستیم براحتی میشد اجزا یا فیلدهای هر ساختمان را نیز بعنوان آرگومان به تابع func ارسال نمود برای نمونه اگر بخواهیم فیلد stu را ارسال نماییم :func(s1.stu) و در طرف مقابل یا بعبارتی در تابع فراخوانی شده func کافی است یک عدد int را بعنوان پارامتر این تابع برای دریافت آرگومان تعریف کنیم .برای ارسال آدرس ساختمان بجای مقدارمربوطه کافی است به مثال مربوطه کمی دقت نماییم :

```
Struct student{
Int stu;
Float grade;
Char name[40];
};
main()
{
Struct student s1;
Struct student *p;
P=&s1;
//func(struct student *p)
.....
func(p)
.........
}
```

```
func(struct student *p)
{
....
Printf("%3d",p->stu);}
```

دقت نمایید دراین حالت یک اشاره گر از نوع ساختمان ایجاد گردید که ادرس ساختمان مفروض ما یعنی s1 در آن قرار داده شد حالا فقط کافی است در هنگام فراخوانی این آدرس را به تابع مورد نظر یا در اینجا (func بعنوان آرگومان ارسال نماییم در مورد محاسن استفاده از آرگومانهای از نوع آدرس یا call by reference قبلا مفصل صحبت کردیم و گفتیم که استفاده از اشاره گر ها در برنامه چقدر به ما برای داشتن برنامه های پویا انعطاف پذیر و سریع از لحاظ زمان اجرا کمک می کنند .بنابر این بهتر است با چند مثال کار با اشاره گر های ساختمان را و همچنین error های احتمالی کامپایلر را بشناسیم.

در هنگام استفاده از فیلدها در تابع فراخوانده شده از علامت <- استفاده خواهیم نمود برای نمونه p->stu معرف فیلد صحیح stu از ساختمان s1 می باشد که آدرس آنرا در اشاره گر p قرار داده بودیم .این شکل نمایش اشاره گر ساختمان است . این دقیقا مثل تعریف های قبلی ماست که داشتیم (*p).stu حالا هم میتوان مثل قبل از همین روش استفاده نمود.

در ادامه بررسی حالات مختلف در برنامه – نمونه ذکر شده می توان مستقیما ادرس s1 را بدون استفاده از اشاره گر p به تابع ارسال نمود.

```
Struct student{
Int stu;
Float grade;
Char name[40];
};
main()
{
Struct student s1;
func(struct student *p)
.................
Func(&s1);}
func(struct student *p) {
...
Printf("%3d",p->stu); }
```

روش دیگر در اسفاده از فیلدها در تابع فراخوانی شده مثل روشهای قبلی است برای نمونه برای استفاده از فیلد stu در تابع func کافی است هر جا لازم باشد به این صورت عمل شود.(*p).stu=10;printf("%3d",(*p).stu) و به این صورت می توان به تک تک فیلدهای ساختمان دسترسی داشت .اما بهتر است از روش اول استفاده کنیم بعدا خواهیم دید چگونه میتوان از اشاره گر های ساختمان برای لیستهای پیوندی استفاده کرد.

برای نمونه برنامه ساده ای که نحوه شبیه سازی یک تایمر را نشان می دهد در ذیل اورده شده تا شما با فراخوانی توابعی که آدرس ساختمان را بعنوان آرگومان می فرستند و همچنین دستیابی به فیلدهای ساختمان در تابع فراخوانی شده آشنا شوید.

```
Struct timer {
Int hour;
Int  min;
Int  sec;
};
main()
{
void  update(struct timer *t);
void  display(struct timer *t);
struct timer time={0};
for(;;)
{
update(&time);
display(&time);
}}
void  update(struct timer *t)
{
(*t).sec++;============================t->sec++;
if((*t).sec==60)=======================if(t->sec==60)
{(*t).sec=0;
(*t).min++;
```

```
}
if((*t).min==60)
{
(*t).min=0;================t->min=0;
(*t).hour++;================t->hour++;
}
if((*t).hour==24)
(*)t).hour=0;
}
void display(struct timer *t)
{
gotoxy(10,10);
printf("%3d",(*t).hour);
printf("%3d",(*t).min);
printf("%3d",(*t).sec);}
```

آرگومانهای تابع main() یا آرگومانهای سطح فرمان

بعنوان بحث پایانی این جلسه درموردآرگومانهای تابع اصلی برنامه لازم است بدانیم اولین و مهمترین تابعی که در برنامه ما اجرا می شود تابع main نام دارد این تابع نیز مثل بقیه توابع می تواند آرگومان داشته باشد به این آرگومانها آرگومانهای سطح فرمان میگویند تابع main بعد از اجرا یک فایل اجرایی با پسوند exe می سازد که در سطح سیستم عامل یعنی بعد از پرامپت داس براحتی با ذکر نام فایل اجرایی ساخته شده میتوان برنامه را از سطح سیستم عامل اجرا کرد.ضمنا میتوان پارامترهای main را نیز برای اجرا بعد از نام فایل ذکر کرد. برای مثال فرض کنید شما برنامه ای با نام test.c را نوشته و با موفقیت آنرا کامپایل نموده باشید در این صورت یک فایل اجرایی با نام test.exe خواهید داشت که میتوان آنرا در سطح سیستم عامل اجرا نمود .کافی است نام فایل اجرایی مذکور تایپ گردد. و یا احیانا پارامترهای آنرا نیز ذکر نمایید تابع main دارای دو پارامتر بنام های argc و argv می باشد که پارامتر argc از نوع صحیح بوده و بصورت پیش فرض مساوی با یک است زیرا نام برنامه خود بعنوان یک آرگومان می باشد.لذا اگر ما بخواهیم برنامه test.exe برای دو آرگومان اجرا شود کافی است argc را مساوی ۳ بدانیم پارامتر argv آرایه از اشاره گر های کاراکتری می باشد که به آرگومانهای خط فرمان اشاره دارندیا بعبارتی تمام آرگومانهای تابع اصلی main بصورت رشته ای فرض می شوند:اگر بخواهیم با اعداد کار کنیم باید آنها را با توابع لازم از حالت رشته ای به عددی تبدیل کنیم .

```
Main(int argc,char *argv[])
```

لازم است بدانیم argv[0] بنام برنامه و argv[1] به آرگومان اول و.....اشاره دارند .فرض میکنیم میخواهیم برنامه ای بنویسیم که عبارت hello را جلوی اسم ورودی چاپ کند.

```
Main(int argc,char *argv[])
{
for(i=۱;i<10;i++)
printf("hello %s",argv[i]);
```

دراینصورت در سطح فرمان با این فرض که نام برنامه ما test باشد کافیست تایپ کنیم c:\> test ali با اینکار عبارت hello جلوی اسم ali چاپ شده و اجرای اینکار روی مونیتور دیده خواهد شد دقیقا مثل اینکه تابع یا برنامه را یکبار برای آرگومان ali اجرا کرده باشیم .

فایلها در زبان سی

(بعضی از مباحث داخل کلاس مثل برنامه ها یا الگوریتم ها بنا به بعضی ملاحظات مثل کمی وقت و یا الزام شما به شرکت در کلاس در این جزوه نخواهد بود و لذا از بابت آن عذر خواهی بنده را بپذیرید.)

قبلا فرا گرفتیم چگونه با استفاده از متغیر ها ی معمولی ساختمان یا آرایه ها داده ها را به برنامه وارد کنیم و مورد پردازش قرار دهیم اما این data Structure ها چون در حافظه اصلی نگهداری می شدند با قطع برق و یا خاتمه اجرای برنامه محتوای آنها که همان داده های برنامه بودند پاک شده و ما مجبور بودیم دفعه بعد مجددا داده های برنامه را از نو در آنها درج کنیم که البته این کار از لحاظ زمانی مقرون به صرفه نبود . مخصوصا اگر حجم داده های وارد شده زیاد هم باشد. برای رفع این معضل می توانیم از ساختمان داده ای بنام فایل که توانایی ضبط شدن بر روی رسانه های جانبی مثل دیسک یا دیسکت را در اختیار ما میگذارد استفاده نماییم تا با قطع برق و یا خاتمه اجرای برنامه مجبور به وارد کردن مجدد داده ها به برنامه نباشیم.

هر فایل شامل مجموعه ای از داده های مرتبط با هم است(اعداد –رکورد ها و کاراکتر ها) که میتوان برای نمونه فایل دانشجویان را که خود ترکیبی از داده های مرتبط با هم هستند را ذکر کرد(رکورد های دانشجویان همان داده های مرتبط با هم هستند.) هر کدام از اجزای یک فایل را رکورد و هر جزء رکورد را یک فیلد می گویند بنابر این در فایل دانشجویان اطلاعات شخصی هر دانشجو تشکیل یک رکورد خواهد داد و این رکورد می تواند شامل فیلد های نام ، نام خانوادگی شماره دانشجویی و.... باشد.که هر کدام انها میتوانند نوع خاص خودشان را داشته باشند پیشتر در رابطه با struct یا رکورد صحبت کردیم . کلا در مورد فایلها اینگونه عمل می شود که ابتدا باید فایل تعریف شود که اینکار با تعریف یک اشاره گر فایل امکان پذیر است فایل باز شده باید از لحاظ درستی باز شدن تست شود که بعدا خواهیم دید اشاره گر فایل باید با ماکروی NULL مقایسه شود تا از صحت باز شدن فایل مطمئن باشیم بعد در صورت لزوم اطلاعاتی را در فایل نوشت یا داده ای را از آن خواند و در نهایت باید پس از خاتمه کار با فایل . فایل مورد نظر را بست .(در کل این مبحث هر جا نیاز به ضمیمه کردن فایلهای هدر داشتید با ctrl+f1 روی تابع مورد نظر میتوان از فایل هدر ان مطلع شده و آن را به برنامه ضمیمه کرد.

انواع فایل (از لحاظ نوع اطلاعات مورد ذخیره سازی) :

۱- فایلهای باینری که از لحاظ سرعت و فضای اشغالی مقرون به صرفه تر هستند زیرا اطلاعات یا داده های برنامه در آنها بصورت ۰و۱ ذخیره می شود.

۲- فایلهای اسکی یا متنی text که اعداد بصورت ارایه ای از کاراکتر ها در آن می نشیند . برای نمونه عدد ۵۰۰۰۰ در ۵ بایت می نشیند که این عدد برای نمونه در فایلهای باینری در دو بایت که برای ذخیره سازی اعداد صحیح لازم است می نشیند.

تفاوتهای این دو نوع فایل به قرار زیر است

الف –(در نحوه ذخیره سازی اعداد)

در فایل متنی اعداد همانطور که وارد می شوند روی دیسک ذخیره می گردند یعنی بصورت آرایه ای از کاراکتر ها برای مثال اگر بخواهیم عدد صحیح ۲۲۳ را در یک فایل متنی ذخیره کنیم این عدد به کمک ۳ بایت در روی دیسک ذخیره خواهد شد در حالی که در زبان سی برای ذخیره سازی اعداد صحیح دو بایت کفایت می کرد لذا مشاهده می شود که فایل text فضای بیشتری را اشغال می نماید.در فایلهای باینری اعداد به گونه دیگری روی دیسک خواهند نشست و ان اینکه در فایلهای دودویی اعداد همان گونه که در حافظه قرار میگیرند بر روی دیسک نیز خواهند نشست .

ب–(در تعیین انتهای خط)

در فایل متنی کاراکتر تعیین انتهای خط هنگام ذخیره شدن بر روی دیسک باید به line feed / carriage return تبدیل شود و بالعکس هنگام بازیابی باید عمل معکوس انجام گردد لذا کار با فایل های متنی زمان گیر تر است چون این تبدیلات - زمان میگیرد.

ج –(در تعیین انتهای فایل)

در فایل متنی پایان فایل با کاراکتر 1A یا ۲۶ در مبنای ۱۰ نمایش داده می شود که با فشردن CTRL+Z روی صفحه کلید ایجاد می گردد در حالی که این کاراکتر 1A در فایل باینری ممکن است جزو اطلاعات خود فایل باشد .یعنی در واقع انتهای فایل در دو نوع TEXT و binary با هم متفاوت است و هر کدام از دید خود پایان فایل را تلقی می کند هر چند که طول فایل توسط خود سیستم نگهداری شده و با توجه به این طول انتهای فایل تعیین می گردد.بعدا خواهیم دید سیستم از یک متغیر اتوماتیک بنام موقعیت سنج فایل برای تعیین محل خاتمه فایل استفاده میکند.برای تعیین اینکه به انتهای فایل رسیده ایم یا نه در مورد فایلهای متنی از تابع EOF استفاده می کنیم و در مورد فایل های باینری از تابع feof(fp) استفاده خواهیم کرد هرچند برای فایلهای متنی نیز می توان از تابع feof(fp) استفاده کرد . fp اشاره گر فایل باز شده است.

از لحاظ سازمان فایل فایلها دو نوع اند :

بر اساس قانون حاکم بر ذخیره و بازیابی اطلاعات از فایل فایلها به دو دسته تقسیم می شوند.

۱- فایلهای ترتیبی(sequential) :

رکورد ها در فایل ترتیبی به همان ترتیبی که خوانده می شوند در فایل ذخیره شده و برای بازیابی آنها هم لازم است رکورد های قبلی را خوانده تا نوبت به رکورد مد نظر ما برسد یعنی برای دستیابی به رکورد ۶۶ لازم است ۶۵ رکورد قبلی خوانده شوند تا نوبت به رکورد مطلوب ما برسد که همان رکورد ۶۶ می باشد.اما در فایلهای تصادفی با یک مکانیسم آدرسدهی می توان مستقیما به رکورد مطلوب دستیابی داشت بعدا خواهیم دید چگونه با تابع کتابخانه ای () fseek می توان مستقیما به سراغ یک رکورد خاصی از فایل تصادفی رفته و آن را بازیابی نمود.فایلهای ترتیبی دارای یک فیلد کلیدی (شاخص رکورد) هستند که بر اساس ان مرتب می باشند برای نمونه شماره دانشجویی می تواند یک فیلد کلید برای فایل ترتیبی دانشجویان باشد.این فیلد کلید یکتایی رکورد ها را تضمین می کند.فایلهای تصادفی نیز که سرعت ذخیره و بازیابی بهتری دارند نیز دارای یک فیلد کلید هستند اما دارای شماره رکورد نیز هستند که این شماره رکورد ها با استفاده از یک الگوریتم محاسباتی تعیین می شود که ارتباط نزدیکی با فیلد کلید دارد.در هنگام تغییر روی فایلهای ترتیبی لازم است کل فایل بازنویسی شود و برای اینکار رکورد مورد تغییر را در فایل جدیدی نوشته و رکورد های بدون تغییر را نیز به ان فایل برده و نام فایل جدید را مساوی نام فایل قدیم می گذاریم و فایل قدیمی را حذف می نماییم این کار یعنی بازنویسی کل فایلهای ترتیبی بهنگام تغییرات روی یک رکورد لازم است.

۲- فایلهای تصادفی :

(random)بر خلاف فایلهای ترتیبی در فایلهای تصادفی می توان با استفاده از یک مکانیسم آدرسدهی و با توابع کتابخانه ای لازم که بعدا خواهیم خواند به رکوردها به طور مستقیم دسترسی خواهیم داشت . فایلهای تصادفی هر رکوردشان دارای یک شماره است لذا اگر فایل دانشجویان دارای اطلاعات ۶۰ دانشجو باشد یعنی ۶۰ رکورد داریم که از شماره ۱ تا ۶۰ شماره گذاری شده اند .برای دیتابیس های بزرگ که تعداد زیادی رکورد قرار است در آن قرار بگیرد از مکانیسم فایلهای ترتیبی نمیشود استفاده کرد و لذا از فایلهای تصادفی که سرعت خوبی در ذخیره و بازیابی اطلاعات دارند استفاده می شود.در فایل های تصادفی کل فایل به ناحیه هایی تقسیم می شود و محلی که برای یک رکورد در نظر گرفته میشود ناحیه رکورد نام دارد یعنی هر رکورد در ناحیه مربوط به خودش نوشته می شود فرض کنید در مورد دانشجویان هر رکورد در ناحیه ای که با شماره دانشجویی یا فیلد کلید رکورد ها مشخص می شود نوشته می شودفرض کنید ۵ دانشجو داریم اولی با شماره ۱۰۰ دومی با شماره ۵۰۰ مشاهده می شود که رکورد دانشجوی اول باید در ناحیه ۱۰۰ نوشته شود و بنابراین ۹۹ ناحیه قبلی خالی می ماند و این یکی از معایب فایلهای تصادفی است که این معضل با پیدایش الگوریتم های آدرسدهی hashing که در واقع ارتباط تنگاتنگی با فیلد کلید دارد و موقعیت ناحیه رکورد را تعیین می کند تا فضای فایل به هدر نرود برطرف شده است.انواع مختلفی از این نوع توابع آدرسدهی بوجود امده اند برای نمونه میتوان برای تعیین ناحیه رکورد شماره دانشجو را بر بزرگترین عدد اول کوچکتر از تعداد دانشجویان تقسیم نمودو یک واحد به باقیمانده تقسیم اضافه نماییم و اگر در این حالت دو یا چند رکورد با این الگوریتم باز هم به

ناحیه یکسانی اشاره داشتند باید دیده شود که ناحیه مورد اشاره از قبل اشغال شده یا نه.برای مثال دانشجوی شماره ۲۳۹۸ و دانشجوی شماره ۱۲۳۴ هر دو به یک ناحیه آدرسدهی خواهند شد که همان ناحیه ۷۱ می باشدزیرا ۹۷/۱۲۳۴+۱=۹۷/۲۳۹۸+۱در این گونه موارد برای رفع مشکل از یک فیلد وضعیت برای اشغال بودن یا نبودن ناحیه استفاده میکنیم در واقع فیلد وضعیت مشخص میکند در ناحیه رکورد داده ای وجود دارد یا نه؟

برای دسترسی به رکورد ها در فایل تصادفی از تابع fseek(fp,int numbyte,origin) استفاده خواهیم کرد که در آن numbyte تعداد بایت هایی است که قرار است جلو یا عقب برویم و origin یکی از سه حالت SEEK_SET یعنی از شروع فایل SEEK_CUR یعنی از موقعیت فعلی فایل و در نهایت SEEK_END یعنی از انتهای فایل.برای نمونه:

Fseek(fp,100,SEEK_CUR) یعنی از موقعیت فعلی فایل ۱۰۰ بایت بجلو(بطرف انتهای فایل)حرکت نماییم که منظور در واقع متغیر اتوماتیک موقعیت سنج فایل می باشد.یا اگر اینگونه از این تابع استفاده کنیم :fseek(fp,100,SEEK_END) منظور این خواهد بود که از انتهای فایل ۱۰۰ بایت بجلو رویم اینکار یعنی افزایش طول فایل به اندازه ۱۰۰ بایت .

برای استفاده از فایل باید فایل را باز نمود که اینکار با تعریف یک اشاره گر فایل امکان پذیر است .کافی است بدانیم هنگام باز کردن فایل برای استفاده لازم است نام و مسیر فایل و نوع آن از لحاظ ورودی یا خروجی و یا هر دو و نیز نوع فایل از لحاظ باینری یا متنی معلوم باشد به مثال های زیر توجه کنید:

پس برای استفاده از فایل یا باز نمودن آن برای استفاده کافی است بدانیم :

نام و مسیر احتمالی فایل را

نوع فایل از لحاظ باینری یا متنی بودن را

نوع فایل از لحاظ ورودی یا خروجی بودن را

بد نیست تعریفی از فایلهای ورودی – خروجی داشته باشیم :

اگر فایلی بصورت خروجی باز شود منظور این است که می توان داده هایی را در ان نوشت اگر نام فایلی را که بصورت خروجی باز کرده ایم قبلا موجود باشد اطلاعات قبلی آن از بین خواهد رفت .بنابر این هنگام باز کردن فایل باید دقت نمود.بهمین صورت اگر فایلی را برای خواندن باز کنیم که وجود نداشته باشد با پیغام خطای مناسب روبرو خواهیم شد.راهکاری وجود دارد که فایلهای موجود را طوری باز کنیم که بتوان به انتهای آن چیزی افزود در ادامه مورد های مختلف باز کردن فایل را خواهیم خواند.

اگر فایلی بصورت ورودی تعریف شود تا باز شده و از استفاده شود در این صورت قادر خواهیم بود فقط از فایل بخوانیم و نه اینکه در آن بنویسیم .readonly

اگر فایلی هم بصورت ورودی و هم بصورت خروجی تعریف شود در اینصورت قادر خواهیم بود هم در آن بنویسیم و هم از آن بخوانیم .

```
FILE  *fp
fp=fopen("a:\\test.dat","w");
If(fp=NULL)
{
Printf("u cannot open file \n");
Printf("enter a key to continue");
Getch();
Exit(0);
}
```

همانطور که دیده شد ابتدا یک اشاره گر فایل بنام fp تعریف گردید که در برنامه به فایل مورد نظر ما اشاره نماید.بعدا با تابع کتابخانه ای fopen() مسیر و نام فایل مورد نظر وارد شد که طبق مثال فایل test.dat روی درایو a باز خواهد شد بهتر است پسوند فایلهای داده ای از

نوع dat باشد و اگر مسیر ذکر نگردد مسیر جاری برای فایل لحاظ خواهد شد.اما "w" به این معنی است که مود فایل از نوع write only می باشد یا بعبارتی فایل ما از نوع متنی خروجی خواهد بود یعنی خواهیم توانست داده هایی را در آن وارد نماییم
اما همانطور که دیده می شود باز کردن فایل به تنهایی کفایت نمیکند بلکه باید تست کرد که اگر فایل بدرستی باز نشده یا بعبارتی اشاره گر فایل تهی می باشد یا به جایی اشاره نمی کند با پیغام مناسب کاربر را در جریان قرار دهیم که فایل باز نشده اینکار با مقایسه اشاره گر فایل با NULL که الگوی آن مثل بیشتر توابع این درس در فایل هدر STDIO.H قرار دارد امکان پذیر است .بعضی دیگر از مودهای باز شدن فایل بقرار ذیل است : لازم است بدانیم تابع exit(0) مارا از برنامه خارج نموده و تمام فایلها را می بندد.

مود های مختلف باز کردن فایل

"w" برای اینکه فایل متنی جدیدی بعنوان خروجی باز شود یعنی قادر باشیم داده هایی را در آن وارد نماییم .اگر فایل موجودی را مجددا با این مود باز کنیم به منزله حذف اطلاعات قبلی فایل خواهد بود.
"r" برای اینکه فایل موجودی را از نوع متنی به عنوان ورودی باز کنیم یعنی قادر باشیم داده های این فایل متنی را بازیابی کنیم.
"a" چنانچه بخواهیم فایل متنی را ایجاد کرده و یا فایل موجود متنی را باز کنیم طوریکه قادر باشیم اطلاعاتی را به انتهای آن اضافه نماییم از این مود استفاده میکنیم اگر فایل متنی قبلا موجود باشد اطلاعات آن حذف نخواهد شد.بلکه قادر خواهیم بود به انتهای ان اطلاعات اضافه نماییم .
"wb" برای اینکه فایل جدیدی از نوع باینری را بعنوان خروجی ایجاد کنیم

"rb" چنانچه فایل موجودی از نوع باینری را بخواهیم بعنوان ورودی باز کنیم و یا اینکه بعبارت دیگر اطلاعاتی را از آن فایل بازیابی نماییم

"ab" چنانچه بخواهیم فایل باینری بسازیم و یا فایل باینری موجودی را باز کنیم طوری که قادر باشیم اطلاعاتی را به انتهای آن اضافه نماییم از این مود استفاده میکنیم اگر فایل موجودی اینگونه باز شود اطلاعات قبلی آن از بین نمی رود.

"r+" فایل موجودی از نوع متنی را بصورت ورودی خروجی باز می کند.

"w+" فایل جدیدی را از نوع متنی باز میکند که هم ورودی باشد و هم خروجی یعنی هم قابل خواندن و هم قابل نوشتن

"a+" فایل موجود یا جدیدی را از نوع متنی هم بصورت ورودی و هم بصورت خروجی تعریف می کند.

"W+b" فایل جدیدی از نوع باینری و بصورت ورودی خروجی می سازد.

"r+b" فایل موجود از نوع باینری را هم بصورت ورودی و هم بصورت خروجی تعریف می کند.

"a+b" فایل جدید یا موجودی را از نوع باینری بصورت ورودی خروجی تعریف میکند.

بستن فایل :

بعد از خاتمه کار فایل حتما باید انرا بست که اینکار با تابع کتابخانه ای (fclose(fp قابل انجام است برای نمونه برای بستن فایلی که اشاره گر فایلی fp به ان اشاره میکند لازم است ;(fclose(fp اگر اشاره گر fp ذکر نگردد یا اینکه اگر از تابع ()fcloseall استفاده نماییم تمام فایلهای باز بجز فایل های stdin-stdout-stdaux-stderr-stdprn بسته خواهند شد.تابع ()fcloseall در صورت موفقیت صفر و در غیر اینصورت EOF را برخواهد گرداند.لازم به توضیح است قبل از بسته شدن فایل تمام بافرها یا حافظه های میانی اختصاص داده شده به فایل در فایل تخلیه می شودبه طور اتوماتیک .هر چند خودمان نیز می توانیم با تابع (fflush(fp تمام بافرها را در فایل تخلیه نماییم این کار برای اطمینان بیشتر لازم است.

اما طریقه ورود و خروج داده به فایل ٤ نوع است

۱- اطلاعات یا داده های ما بصورت کاراکتر به کاراکتر در فایل نوشته و یا از آن خوانده می شوند با توابع کتابخانه ای ()putc , ()getc یا برای نمونه در نگارش های قدیمیتر با توابع ()fgetc,()putcبرای مثال فرض کنید می خوهیم تمام کاراکتر های خوانده شده را تا علامت $ در فایل input.dat بریزیم

```
FILE *in,*out;
If((in=fopen("input.dat","w")==NULL)
{Printf("cannot open file")
Exit(1);
}
While(ch=getchar()!='$')
Putc(ch,in);
If((in=fopen("output.dat","r")==NULL)
{Printf("cannot open file")
Exit(1);
}
while(ch=getc(in)!=EOF)
putc(ch,out);
```

لازم بذکر است سیستم برای تعیین محلی که باید کاراکتر ها در فایل نوشته و یا از ان خو انده شوند یک موقعیت سنج فایل دارد که با آن تعیین می کند کاراکتر بعدی کجای فایل بنشیند و یا از کجا خوانده شود.در واقع با هر بار خواندن و یا نوشتن بر روی فایل این موقعیت سنج فایل بطور اتوماتیک تغییر میکند تا موقعیت فعلی فایل را تعیین نمایدو عمل نوشتن یا خواندن از روی فایل از جایی شروع می شود که این موقعیت سنج اعلام میکند.

نکته دیگری که هنگام کار با فایلها باید مد نظر داشت این است که هنگام خواندن از فایل لازم است انتهای فایل تست شود و چنانچه به انتهای فایل رسیده باشیم با پیغام مناسبی کاربر را مطلع نماییم تا دستور خواندن بعدی صادر نگردد.چنانچه به انتهای فایل رسیده باشیم و باز هم دستور خواندنی از فایل صادر گردد پیغام خطایی مبنی بر نبودن اطلاعات در فایل صادر خواهد شد.در فایلهای متنی در صورت رسیدن به انتهای فایل تابع ()getc مقدار EOF را بر میگرداند که موید انتهای فایل است بنابر این میتوان تستی قرار داد که اگر به EOF رسیدیم به منزله پایان فایل باشد و دستور خواندن بعدی صادر نگردد.اما در فایلهای باینری برای تست کردن انتهای فایل از تابع کتابخانه ای ()feof استفاده می شود هرچند این تابع را برای تست تعیین انتهای فایلهای متنی نیز میتوان بکاربرد.

۲- اطلاعات بصورت رشته ای در فایل نوشته و یا از ان خوانده می شود با توابع کتابخانه ای ()fputs,()fgets

برای نمونه اگر بخواهیم رشته ای را در فایل بنویسیم :

```
FILE *fp
Char name[20];
Gets(name);
Strcat(name,"\n");
Fputs(name,fp);
Rewind(fp);
Fgets(name,19,fp);
```

در تابع ()fputs اشاره گر name به رشته ای اشاره میکند که قرار است در فایلی که fp به آن اشاره دارد نوشته شودیعنی در واقع fpاشاره گری است که به فایل مورد نظر ما اشاره دارد.مفهوم تابع ()fgetsمقداری متفاوت از تابع قبلی است زیرا در ان اشاره گر اول به جایی اشاره

میکند که قرار است ۱۹ بایت از فایلی که fp به ان اشاره دارد برداشته شده و در این متغیر قرار داده شود. به عبارتی تابع fgets(name,n,fp) از ابتدای فایل شروع به خواندن میکند تا یک رشته بطول n کاراکتر را از فایلی که fp به آن اشاره دارد بردارد و در اشاره گر name قرار دهد.

تفاوت gets()با fgets() در تابع gets() کاراکتر پایان خط جز رشته برداشته شده نیست ولی در تابع fgets() کاراکتر پایان خط در زمره کاراکتر های برداشته شده توسط تابع است .

۳- خواندن و نوشتن در فایل با فرمت خاص :دقیقا مثل توابع scanf(),printf() که می توانستیم متغیر هایی را با فرمت دلخواه خودمان مقدار دهی کنیم یا بخوانیم در اینجا نیز می توان با فرمت دلخواه در فایل نوشت یا از فایل خواند.با توابع کتابخانه ای fscanf(fp,"%d%s"&d,name) که با فرمت مذکور دو متغیر عددی صحیح d و آرایه رشته ای name را مقدار دهی می کند در واقع از فایلی که fp به ان اشاره می کند این دو متغیر مقدار دهی می شوند.

و یا با تابع fprintf(fp,"%s%d",name,i) که باعث وارد کردن داده های name از نوع استرینگ واز نوع صحیح می شود در هر دو تابع مذکور fp به فایلهایی اشاره میکند که قرار است یا در آن نوشته و یا از خوانده شود.

در واقع با تابع fscanf() lمی توانیم با فرمت خاص از فایل بخوانیم و با تابع fprintf() می توان در فایل نوشت .

۴-نوشتن در فایل و خواندن از ان بصورت رکورد به رکورد:

روش چهارم که موضوع درس این هفته است خواندن و نوشتن بر روی فایل با استفاده از رکوردهاست یعنی بصورت رکورد به رکورد در فایل نوشته و یا از آن بخوانیم که پر کاربردترین روش ذخیره و بازیابی بر روی فایلهاست .که با دو تابع کتابخانه ای fread() و fwrite() انجام میشود.در واقع روش چهارم ذخیره و بازیابی فایلها این است که ساختمان به ساختمان در فایل نوشته و از آن بخوانیم .

Fread(اشاره گر فایل,تعداد ,طول رکورد,آدرس متغیر اشاره گر)

Fwite(اشاره گر فایل,تعداد ,طول رکورد,آدرس متغیر اشاره گر)

Fread(name,sizeof(char),10,fp) باعث خوانده شده ۱۰ تا یک بایتی از فایلی که fp به ان اشاره دارد خوانده شده و این ۱۰ بایت در آدرسی که متغیر name به آن اشاره دارد قرار خواهند گرفت .

Fread(&emp,sizeof (struct em),1,fp) که باعث خوانده شدن ۱ عدد ساختمان (رکورد) از فایلی که fp به آن اشاره دارد میشود این ساختمان یا رکورد درآدرس ساختمان emp قرار خواهد گرفت لازم بذکر است متغیر emp در طول برنامه باید از نوع struct تعریف شده باشد.

Fwrite(name,sizeof(char),10,fp) باعث نوشته شدن ۱۰ بایت از متغیر name در فایلی که fp به آن اشاره دارد میشود.

Fwrite(&emp,sizeof(struct em),1,fp) باعث نوشته شدن ۱ ساختمان یا رکورد emp در فایلی که fpبه آن اشاره دارد میشود.بنابر این دیده میشود که با دوتابع سریع fread() , fwrite() میتوان رکورد به رکورد در فایل نوشت و یا از آن خواند.در ذیل تابعی که تعداد کل واحد ها و نمره کل دانشجویان را میگیرد و معدل هر دانشجو رادر خروجی چاپ میکند آمده است .

```
.#define max 100
struct student{
char name[10];
char family[30];
int sumvahed;
float sumgrade;
}st[max];
main()
{
FILE *fp;
If(fp=fopen("in.dat","w+b")==NULL)
{Printf("you cannot open this file\n");
Printf("press any key to continue");
```

```
Getch();
Exit(0);
}
gotoxy(2,2);
printf("how many student?");
scanf("%d",&i);
printf("the name\tfamily\tsumvahed\tsumgrade\n");

printf("_____");
k=4;
for(j=0;j<I;j++)
{
gets(st[i].name);
if(!st[i].name[0])
break;
gotoxy(5,k);
scanf("%d",&st[j].sumvahed);
gotoxy(8,k);
scanf("%f",&st[j].sumgrade);
fwrite(&st[j],sizeof(struct student),1,fp);
k++;
}
rewind(fp);
clrscr();
gotoxy(10,10);
printf("output is\n");
puts("name   ……..average\n");
for(j=0;j<I;j++);
{
fread(&st[j],sizeof(struct student),1,fp);
while(!feof(fp)
{
gotoxy(3,g);
puts(st[j].name);
gotoxy(12,g);
puts(st[j].family);
gotoxy(25,g);
printf("%f",st[j].sumgrade/st[j].sumvahed);
g++;
}}}
```

تابع rewind()

لازم به توضیح است اگر فایل ما از نوع ورودی –خروجی باشد لازم است مدام به ابتدا و انتهای فایل رفت و آمد داشته باشیم اینکار برای برگشت به ابتدای فایل که در اثر آن موقعیت سنج فایل به ابتدای فایل می آید لازم است ومکانیسم این روش بابسته و باز کردن مجدد فایل مقدور است اما برای پرهیز از این روش ناکارامد کافی است از تابع کتابخانه ای rewind(fp) استفاده کنیم در اینجا fp یک مثال است از اشاره گری که به فایل مورد نظر ما اشاره داردبا اینکار از هر جای فعلی فایل که هستیم به ابتدای آن گریز میزنیم .البته این مساله بیشتر در مورد فایلهای ورودی خروجی عینیت دارد تا در مورد فایلهای صرفا ورودی یا صرفا خروجی .

تابع remove()

برای حذف فایل مورد استفاده قرار میگیرد تا هارد از شر فایلهای بدرد نخور خلاص شود.

تابع rmdir("dos") باعث حذف یک مسیر در اینجا مسیر dos خواهد شد. به شرط اینکه مسیر مذکور خالی باشد.

تابع ferror()

برای عیب یابی و بررسی مشکلات احتمالی در بازکردن و یا کلا کار کردن با فایل هامثل عدم وجود فضای کافی برای باز کردن فایل و یا آماده نبودن دستگاهی که قرار است فایل در آنجا تشکیل شودو یا مواردی از این قبیل که منجر به بروز خطا در هنگام کار با فایل ها می شود لازم است از تابع کتابخانه ای ferror(fp) استفاده نمود که در اینجا fp مثالی است که بیانگر اشاره گر به فایل ماست و چنانچه برای کار با فایل مذکور با error ای روبرو باشیم این تابع مقدارغیر صفر را بر میگرداند و در غیر اینصورت مقداری صفر را .اگر خطایی تشخیص داده شد برای نمونه از فایل تازه باز شده ای که هیچ اطلاعاتی ندارد بخواهیم بخوانیم در این صورت باید یا فایل مربوطه بسته شود و یا با تابع clearerr() و یا rewind(fp) خطای تشخیص داده شده را reset نمودبرای ادامه کار با فایل.

برای حذف فایلهای غیر ضروری نیز لازم است با تابع کتابخانه ای remove(fp) فایلی را که مثلا fp به ان اشاره دارد را حذف نمود تا فضای دیسک بیهوده در اشغال این فایلها نباشد .الگوی این فایلها در فایل هدر stdio.h قرار دارد.

تابع clearerr(fp): باعث می شود تا نشانگر های خطای فایل reset شده و برنامه بتواند ادامه یابد

تابع rename("st1.dat","st2.dat") باعث تغییر نام فایل می شود.اگر با موفقیت انجام شود صفر والا یک مقدار غیر صفر برمیگرداند.

توابع fd=fileno(fp) که برای تعیین مشخصه فایل که در متغیر fd قرار میگیرد و تابع filelength(fd) که طول فایل را بر حسب بایت و بصورت یک عدد صحیح بر میگرداند .این دو تابع با هم کاربرد دارند.

بافر :(buffer)

برای تسریع در اعمال ورودی و خروجی فایل ها سیستم به آنها حافظه ای اختصاص می دهد که معروف به حافظه بافر است در هنگام خروج از برنامه لازم است با تابع کتابخانه ای fflush(fp) مقادیری که در بافر وجود دارد یا اصطلاحا داده های بافر را در فایل تخلیه نماییم اگر اینکار با موفقیت انجام شود مقدار صفر و در غیر اینصورت EOF را بر خواهد گرداند.البته در هنگام بستن فایل اینکار بطور خود کار انجام می شود.البته در مثال ما این کار بر روی فایلی اعمال میشود که اشاره گر fp به آن اشاره دارد.الگوی تابع مذکور در فایل هدر STDIO.H قرار دارد و چنانچه fp داخل پرانتز ذکر نگردد این کار بر روی تمام فایلهای خروجی باز شده انجام می شود. اگر کار این تابع با موفقیت انجام شود ، مقداری که توسط آن برگردانده می شود برابر با صفر وگرنه برابر با EOF خواهد بود.وجود این تابع بخاطر اطمینان از آخرین انتقال داده ها به روی فایل می باشد.

ضمن اینکه فرا گرفتیم هنگام باز کردن فایل اشاره گر آنرا با NULL مقایسه نماییم تا از درست باز شدن فایل مطمئن شویم و یا در طول برنامه بعد از هر کاری روی فایل ها از تابع ferror(fp) برای تعیین اینکه روی این فایل خطایی روی داده است یا خیر استفاده نماییم نمونه ای از این خطا ها می تواند این باشد که جای کافی برای این فایل وجود نداشته باشد .همین طور گفته شد که در مورد فایل های ورودی و خروجی برای اینکه موقعیت سنج فایل را به ابتدای فایل بیاوریم اصلا لزومی به بستن و باز کردن مجدد فایل نداریم و کافی است با rewind(fp) موقعیت سنج فایلی را که fp به آن اشاره دارد را به ابتدای فایل بیاوریم.موقعیت سنج فایل متغیر اتوماتیکی بود که خود سیستم برای تعیین محل خواندن یا نوشتن بر روی فایل از آن کمک می گرفت.و با هر بار صدور فرمان خواندن یا نوشتن مقدار آن اتوماتیک تغییر میکرد.

همچنین فراگرفتیم در هنگام اجرای برنامه به زبان سی ۵ فایل بطور اتوماتیک باز شده و همانطور که بصورت اتوماتیک باز شدند همانگونه نیز در هنگام پایان کار و بهنگام خروج از برنامه خودبخود بسته می شوند.یعنی همانگونه که باز شدن این فایلها به کاربر ربطی نداشت همانطور هم کاربر در بسته شدن این فایلها نقشی ندارد.این فایلها که از آن بعنوان دستگاههای ورودی خروجی استاندارد نیز نام برده می شوند کانال ار تباطی بین برنامه و فایل برقرار می کنند برای نمونه stderr باعث می شود برنامه بتواند خطاهای خود را از طریق مونیتور به اطلاع کاربر برساندو یا stdin باعث می شود برنامه بتواند داده های مورد نیاز خود را از کاربر و از طریق کیبورد دریافت نماید این دستگاههای استاندارد ورودی و خروجی عبارتند از:

Stdin که در واقع دستگاه استاندارد ورودی یا کیبورد است .

Stdout که در واقع دستگاه استاندارد خروجی یا مونیتور است .

Stdprn که در واقع دستگاه استاندارد جهت چاپ یا چاپگر موازی می باشد.

Stdaux که در واقع پورت سریال می باشد

Stderr که در واقع دستگاه استاندارد جهت ثبت پیامهای خطا می باشد و یا بعبارتی مونیتور.

برای نمونه دستور ;(fscanf(stdin,"%d,%d",&x,&y باعث خوانده شدن x,y از دستگاه استاندارد ورودی یا صفحه کلید خواهد شد.در اینجا stdin اشاره گری است به دستگاه استاندارد ورودی

در این در س با بعضی از توابع کتابخانه ای کار با فهرستها نیز آشنا می شویم این توابع بما کمک می کنند که در برنامه مدیریت فهرستها را بعهده بگیریم و اگر خواستیم فهرستی را حذف یا ایجاد کنیم یا مسیر جاری را عوض نموده و دیسک فعال را عوض کنیم و از این دست کارها .

این توابع در سی استاندارد نیستند و در توربو سی وجود دارند.و فایل هدر انها dir.h می باشد

تابع mkdir("dos")

باعث ساخته شدن فهرست یا دایرکتوری dos در مسیر جاری خواهد شداگر این تابع با موفقیت انجام شود مقدار صفر را برمیگرداند و در غیر اینصورت مقدار ۱-

تابع ;chdir("c:\\reza\\dos")

باعث تغییر مسیر جاری به مسیر ذکر شده خواهد شد همانطور که دیده می شود برای معرفی \ به کامپایلر از کاراکتر \ استفاده شده است .

.این تابع نیز اگر با موفقیت انجام شود صفر و الا ۱- را برمیگرداند.

تابع ;rmdir("dos")

این تابع باعث میشود دایرکتوری dos پاک شود البته مشروط به اینکه این فهرست خالی بوده و دایرکتوری جاری نباشد.این تابع نیز اگر با موفقیت انجام شود صفر و الا ۱- را برمیگرداند.

تابع int getdisk(void)

باعث تعیین شماره درایو فعال می شودپیش فرض این است که شماره درایو a صفر b یک و الا آخر .

تابع searchpath("edit.com")

باعث پیدا شدن مسیری که توسط دستور path داس برای فایل edit.com تعیین شده می شود و چنانچه بخواهیم میتوانیم مسیر تعیین شده را در خروجی داشته باشیم

تابع setdisk()

که برای مثال setdisk(0) باعث تعیین درایو a بعنوان درایو فعال خواهد شد.

توابع دیگری هم هستند که در برنامه ها میتوانند کمک زیادی به برنامه نویس باشند.که در اینجا مجال بررسی همه آنها وجود ندارد.

تاکنون یاد گرفتیم به ۴ صورت میتوان در فایل نوشت و یا از آن خواند

کاراکتر به کاراکتر مثل putc(ch,fp) و یا برای خواندن از فایل getc(fp)

به صورت رشته ای از کاراکتر ها از فایل بخوانیم و یا در فایل بنویسیم :fputs(name,fp) و یا برای خواندن fgets(name,79,fp) که باعث میشد رشته ای حد اکثر ۷۹ کاراکتری از فایلی که fp به آن اشاره دارد خوانده شده و درآدرس متغیر رشته ای name قرار بگیرد.

حالت سوم نوشتن و خواندن از فایل با فرمت خاص بود دقیقا مثل توابع scanf() و یا printf() در اینجا هم برای خواندن از فایل و نوشتن بر روی فایل از توابع fprintf() و fscanf() استفاده میشود.(fcanf(fp,"%d",&i

ویا fprintf(fp,"%5d",i)