برنامه نویسی به زبان C در سیستم عامل GNU

نوشته : مَهدی رسولی

http://www.gnuiran.org

مقدمه

1– هدف

به آموزش GNU C خوش آمدید. هدف از نوشتن این مقاله آموزش نوشتن نرم افزار به کمک زبان برنامه نویسی C به کاربران سیستم‌های GNU است. اصولا این مقاله به عنوان خود آموز برای افراد مبتدی نوشته شده امابرای استفاده به عنوان مرجع توسط کاربران با تجربه نیز قابل استفاده است.

مطالب پایه‌ای در چند فصل اول شرح داده شده اند. کاربران مبتدی باید به دقت آنها را مطالعه کنند اما افراد با تجربه می توانند به صورت سطحی از کنار آنها بگذرند.تمام اطلاعات در اینجا وجود دارد و به هیچ گونه اطلاعات قبلی در مورد برنامه نویسی نیاز نیست.

فرض بر این است که خواننده به یک سیستم عامل GNUای دسترسی دارد. اگر چه هدف اصلی کاربران GNU هستند اما مطالب این کتاب تا ۹۸ درصد برای کاربران – Open BSD Free BSD یا Net BSD نیز قابل استفاده است. تمام فرمان‌ها در کنار نمونه‌های برنامه نویسی ذکر شده اند با این حال آشنایی مختصر با پوسته (خط فرمان یا ترمینال) مفید خواهد بود. شما تنها به مهارت در استفاده از یک ویرایشگر متن احتیاج دارید. هر ویرایشگری قابل استفاده خواهد بود. GNU Emacs یک نمونه بسیار خوب برای برنامه نویسان است. این برنامه به مدت بیش از 20 سال است که توسعه می یابد و دارای صدها خصیصه مفید می باشد. GNU Nano یکی دیگر از ویرایشگرهای متن مفید و ساده است که شما می توانید از آن استفاده کنید. برخی برنامه نویسان مایل به استفاده از vi هستند. اگر شما از قبل یک ویرایشگر محبوب دارید می توانید از آن استفاده کنید. همچنین ویرایشگرهای گرافیکی نظیر Anjuta یا KDevelop برای برنامه نویسان وجود دارند اما اکثر برنامه نویسان ویرایشگرهای محیط‌های متنی را ترجیح می دهند. (Anjuta و KDevelope و نیز GNU Emacs بیش از یک ویرایشگر متن هستند و آن‌ها را IDE نیز

می‌نامند که مخفف Integrated Development Environment است. به معنی محیط توسعه‌ی مجتمع.)

2– مفاد این متن

مطالب این مقاله می توانند به دو بخش تقسیم شوند : الف. هسته زبان سی و ب. توابع استاندارد ارایه شده برای برنامه نویسان. این توابع توسط کتابخانه توابع سی یا GNU libc فراهم شده اند که بخشی از هر سیستم GNU/Linux می باشد. هیچ کدام از این دو بخش بدون دیگری استفاده نمی‌شود اما در این مقاله در شروع تمر کز بر روی هسته زبان C و در بخش‌های آخر بحث‌های بیشتری در مورد libc خواهد بود. سازماندهی مطالب به گونه‌ای است تا آموزش برنامه نویسی C در یک حالت صعودی انجام شود به گونه‌ای که مطالب هر فصل بر روی فصل قبلی بنا می شود. برخی جنبه‌های هسته زبان برای کاربران با تجربه کاربرد دارند بنابراین در فصل‌های پایانی ذکر خواهند شد. زبان C به خودی خود می تواند ساختارهای تصمیم گیری و تکرار دستورات را ایجاد کند و ذخیره سازی داده‌ها و محاسبات ریاضی را انجام دهد. با همین اهمیت یک روش برای استفاده از توابع خارجی نظیر libc را فراهم میکند. libc توابعی برای کارهای نظیر خواندن و نوشتن فایل‌ها – دسته بندی و جست و جوی داده – دریافت ورودی برای کاربر و نمایش داده برای او – ارتباط از طریق شبکه – ساخت برنامه‌های قابل ترجمه و غیره فراهم می‌کند.

3–چرا زبان برنامه نویسیC برای آموزش؟

زبان برنامه نویسی C استاندارد است. C زبان استاندارد برنامه نویسی سیستم‌های بر پایه GNU و BSD است. اغلب این سیستم عامل‌ها به همراه برنامه‌های کاربردی که بر روی آنها اجرا می شوند به زبان C نوشته شده اند. C بیش از سی سال پیش برای نوشتن سیستم‌های عامل و برنامه‌های کاربردی شروع به توسعه یافت. C کوچک است. طراحی قابل بسط به آن این امکان را داد تا به همراه صنعت کامپیوتر توسعه پیدا کند. به دلیل قدمت و محبوبیت C زبانی دارای پشتیبانی بسیار خوب است. ابزارهای زیادی به وجود آمده اند تا برنامه نویسی C را آسان تر کنند . و این ابزارها غالبا بسیار کامل و استاندارد هستند. غریب به اتفاق نرم افزارهایی که ما در این مقاله از آنها استفاده خواهیم کرد با C نوشته شده‌اند.

4– چرا استفاده از سیستم عامل GNU ؟

سیستم عامل GNU/Linux یک سیستم عامل شبه Unix است که بیش از بیست سال پیش شروع به توسعه کرد. شروع توسعه‌ی این سیستم عامل پرژه‌ی GNU است که ریچارد استالمن آن را برای فراهم آوردن آزادی نرم‌افزارها ایجاد کرد. پس از مدتی با فراهم شدن هسته‌ای به نام Linux از سوی لینوس تروولدز سیستم عامل GNU/Linux شروع به کار نمود. نرم‌افزارهای مورد استفاده در سیستم GNU و سیستم عامل GNU/Linux به دلیل آزاد بودن (و متعاقبا اوپن سورس بودن) نرم افزارهایی با پایداری و قابلیت اطمینان و منطبق بر استانداردها هستند. اغلب سیستم‌های GNU از Linux به عنوان هسته استفاده می کنند. این سیستم‌ها غالبا به عنوان سیستم‌های GNU/Linux شناخته می شوند. (برای دریافت اطلاعات بیشتر می‌تونید مقاله‌ی لینوکس چیست را در همین سایت مطالعه کنید.)

5– چرا نرم افزار آزاد؟

مهم‌ترین مطلب درباره‌ی GNU فراهم آوردن نرم‌افزارهایی آزاد است؛ یک نرم افزار زمانی یک نرم افزار آزاد است که چهار شرط را برای کاربران فراهم کند :

۰– آزادی برای اجرای برنامه برای هر هدفی و استفاده از آن در هر زمینه‌ای.

۱–آزادی درمطالعه درباره‌ی نحوه‌ی کار یک برنامه.(در دسترس بودن سورس کد پیش شرط این موضوع است)

۲–آزادی در توزیع مجدد برنامه و یا کپی آن برای همسایگان.

۳–آزادی برای توسعه و بهبود و تغییر برنامه و منتشر کردن دوباره‌ی برنامه. (در دسترس بودن سورس کد پیش شرط این موضوع است)

نرم افزارهایی که آزاد نیستند نرم افزار دارای حق مالکیت (یا انحصاری) نامیده می شوند، چرا که یک نفر ادعای مالکیت آنها را می کند و دیگران را از به اشتراک گذاری و ایجاد تغییر در آن منع می نماید. از دیدگاه اخلاقی نوشتن نرم افزار آزاد راهی اجتماعی تر است. نرم افزار آزاد با اجازه دادن به کاربرانش برای کمک به خود از طریق ایجاد تغییرات دلخواه در نرم افزار (یا استفاده از فردی دیگر برای ایجاد تغییرات)

آنها را قدرتمند می سازند. آن به افراد اجازه می دهد تا از طریق به اشتراک گذاردن نرم افزار با همسایگانشان به آنها کمک کنند. نرم افرار انحصاری بر عکس عمل می کند : آن با گفتن این نکته به افراد که پاسخ مثبت دادن به تقاضای کمک مردم جرم است به اشتراک گذاری نرم افزار را غیر قانونی می سازد. در حالیکه نرم افزار آزاد به افراد اجازه می دهد تا با پخش نسخه‌های بهبود یافته نرم افزار به اجتماع خود کمک کنند.

نرم افزار آزاد همچنین بر ضد افراد فقیر یا افراد کشورهای در حال توسعه موضع نمی گیرد.با اعطای تمام آزادیهای بالا به آنها اجازه می دهدتا بدون اجبار برای پرداخت مبالغ غیر ممکن پول برای حق امتیاز نرم افزار از کامپیوتر استفاده کنند. در نهایت برتری‌های فنی نیز وجود دارد. نرم افزار آزاد فارغ از دسیسه‌های بازاریابی است.آن خود را برای مجبور کردن کاربران به خرید قطعه‌های اضافی برنامه محدود نمی‌کند. هر قطعه از GNU طوری طراحی شده که تا حد ممکن مفید باشد. به عنوان یک برنامه نویس شما از همان نرم افزار برنامه نویسی C استفاده می کنید که در پروژه‌های بزرگ استفاده می شود.

نر افزارهای انحصاری تنها در یک شکل قابل خواندن برای ماشین توزیع می شوند. این یعنی کاربر نمی تواند از آنچه درون نرم افزار اتفاق می افتد اطلاع یابد.در مقابل نرم افزار آزاد باید به همراه کد منبع خود در یک فرمت قابل خواندن برای انسان منتشر شود. به عنوان یک برنامه نویس شما می توانید کدهای هر قطعه از نرم افزار ازاد را که مایل باشید مطالعه کنید. اگر خطایی در برنامه وجود داشته باشد شما می توانید خودتان آنرا رفع کنید.

این آزادی در رفع خطاها و اضافه ساختن امکانات چیزی است که نرم افزار GNU را عالی می سازد. همه کدها برای بازبینی در دسترس اند.

نرم افزار آزاد قصد تغییر جهان برای بهتر شدن را دارد.

بخش اول : دیباچه‌ای بر C

1.1 زبان برنامه نویسی چیست؟

یک زبان برنامه نویسی قالبی برای طرح ریزی برای اجرای موارد سفارش داده شده توسط کامپیوتر تعریف می‌کند. زبان‌های برنامه نویسی به سه دسته تقسیم می شوند. ۱.زبان‌های ترجمه‌ای ۲. زبان‌های کامپایل شده و ۳.زبان ماشین. از این میان تنها زبان ماشین به طور مستقیم توسط کامپیوتر قابل فهم است.

یک زبان ماشین مجموعه‌ای از راهنمایی‌ها(instructions) است که CPU کامپیوتر (واحد پردازشگر مرکزی) می‌فهمد. تمام راهنمایی‌ها و اطلاعات توسط اعداد نمایش داده می شوند. خیلی سریع برای کامپیوتر و خیلی سخت برای مغز انسان برای خواندن و نوشتن آنها. برای آسان تر ساختن وظیفه برنامه نویسی کامپیوتر مردم زبان آسانتری را ساختند که Assembly نامیده می شد. یک زبان Assembly نامهای متنی برای دستورات در دسترس زبان ماشین فراهم میکند. این مطلب در کنار این واقعیت که زبان اسمبلی به برنامه نویسان اجازه اضافه کرن فاصله‌ها و تب‌ها را در کدها می دهد زبان Assembly را برای برنامه نویسی بسیار راحت می سازد. برنامه‌های Assembly می توانند برای ترجمه به زبان ماشین به منظور فهم CPU به یک Assembler سپرده شوند. استفاده از زبان‌های Assembly خیلی سریع گسترش یافت. آنها به عنوان زبانهای نسل دوم شناخته می شدند. اما هنوز هم دو مشکل وجود داشت، مشکل اول آن بود که هر دستور Assembly تنها یک وظیفه خیلی ابتدایی نظیر جمع دو عدد و یا بارگذاری یک مقدار از حافظه را انجام می داد، استفاده از این دستورات کوچک واقعا ملالت آور بود. مشکل دوم بزرگتر بود، هر کدام از برنامه‌های نوشته شده به Assembly به نوع خاصی از CPU وابسته بود، به عبارت دیگر به یک معماری خاص از پردازنده‌های مرکزی وابسطه بودند. هر نوع از CPU زبان ماشین مخصوص به خود را داشت، بنابراین زبان Assembly مخصوص به خود را داشت. کار مهم بعدی طراحی زبانی بود که به زبان ماشین انواع زیادی از CPU قابل ترجمه باشد. این زبانهای جدید وابسته به ماشین زبانهای نسل سوم یا زبانهای سطح بالا نامیده می شوند. این زبانها که برای آسان خوانده شدن طراحی شده بودند از کلمات انگلیسی – سمبل‌های پایه‌ای ریاضی و علایم نقطه گذاری تشکیل می شدند. این زبانها به عبارات ساده اجازه می دهند تا به طور مختصر بیان شوند به عنوام نمونه جمع دو عدد و ذخیره سازی حاصل در حافظه می تواند به صورت زیر بیان شود:

data = 10 + 200;

که بر عبارت زیر ارجحیت دارد:

Load R1, 10

Load R2, 200

Addi R1, R2

Store R2, L1

1.2 زبان C چیست ؟

سپس ابزاری که کامپایلر نامیده می شود برای تبدیل کدهای سطح بالا به زبان ماشین استفاده می شود. یک برنامه می تواند به زبان C نوشته شده و سپس برای هر کامپیوتری کامپایل شود. درگیری با جزییات سخت افزاری بر عهده کامپایلر است. برای مقایسه برتری زبان C بر Assembly به یک برنامه کوچک که در هر دو زبان نوشته شده نگاه کنید:

نمونه 1.1 C در مقایسه با Assembly :

```
. .section  .rodata
.LC0:
          .string   "Tax Due: %d "
          .text
          .align 2
.globl main
          .type     main,@function
main:
          pushl     %ebp
          movl      %esp, %ebp
          subl      $24, %esp
          andl      $-16, %esp
          movl      $0, %eax
          subl      %eax, %esp
          movl      $1000, %eax
          movl      $400, %edx
          movl      $0x3e6147ae, -12(%ebp)
          subl      %edx, %eax
          pushl     %eax
          fildl     (%esp)
          leal      4(%esp), %esp
          fmuls     -12(%ebp)
          fnstcw    -18(%ebp)
          movw      -18(%ebp), %ax
          movb      $12, %ah
          movw      %ax, -20(%ebp)
          fldcw     -20(%ebp)
```

```
          fistpl    -16(%ebp)
          fldcw     -18(%ebp)
          subl      $8, %esp
          pushl     -16(%ebp)
          pushl     $.LC0
          call      printf
          addl      $16, %esp
          movl      $1, %eax
          leave
          ret
.Lfe1:
          .size     main,.Lfe1-main
```

همان برنامه این‌بار که اینبار به زبان C نوشته شده است :

```
#include

int
main()
{
  int wages = 1000;
  int tax_allowance = 400;
  float tax_rate = 0.22;
  int tax_due;

  tax_due = (wages - tax_allowance) * tax_rate;

  printf("Tax Due: %d euro
", tax_due);

  return 0;
}
```

بدون شناخت از زبان C شما کدام برنامه را برای خواندن آسانتر می دانید؟ خروجی هر دو برنامه مشابه است :

" Tax Due: 131 euro"

کد Assembly نشان داده شده که در مجموعه راهنمایی 80386 نوشته شده در ماشینی که از مجموعه راهنمایی دیگری استفاده می کند کار نخواهد کرد . اما کد C می تواند عملا برای هر کامپیوتر با هر معماری پردازشگری کامپایل شود.

1.3 ابزارهای برنامه نویسی

سیستم‌های GNU به همراه کامپایلری به نام gcc ارایه می شود. gcc از ابتدا مخفف GNU C Compiler بود ولی پس از مدتی که قابلیت کامپایل زبان‌های دیگری غیر از C به آن افزوده شد نام آن را GNU Compiler Colection خطاب می‌کنند. پدید آورنده‌ی اصلی GCC ریچارد استالمن، بنیانگذار پروژه

GNU است. نخستین نسخه GCC در سال ۱۹۸۷ انتشار یافت که یک پیشرفت مهم و اساسی در پیشبرد اهداف GNU Project محسوب می‌شد زیرا محصول جدید اولین کامپایلر بهینه سازی شده قابل حمل ANSI C به عنوان یک نرم افزار آزاد محسوب می شد. در سال ۱۹۹۲ نسخه 2.0 کامپایلر GCC عرضه شد. نسخه جدید قابلیت کامپایل کدهای C++ را نیز داشت.

در سال ۱۹۹۷ یک انشعاب آزمایشی در GCC به نام EGCC به منظور بهینه سازی کامپیایلر و پشتیبانی کامل تر از C++ ایجاد شد. در ادامه EGCC به عنوان نسل بعدی کامپایلر GCC پذیرفته شد و تکامل آن باعث انتشار نسخه سوم GCC در سال ۲۰۰٤ گردید. چهارمین نسخه از کامپایلر GCC در سال ۲۰۰۵ عرضه شد. برای اطمینان از اینکه gccرا نصب کرده‌اید یا خیر دستورات زیر را وارد کنید :

ciaran@pooh:~/$ gcc --version 4.1.1 $ciaran@pooh:~/

نسخه‌ای از gcc که شما استفاده می کنید ممکن است متفاوت باشد. هر چیزی شبیه این از قبیل 3.4.6 یا 4.1.1 مناسب است. اگر پیغام خطای "command not found " را دریافت می کنید، شما gcc را نصب نکرده‌اید. اگر لینوکستان را از روی cd نصب کرده‌اید می توانید gcc را در آن پیدا کنید. اگر شما نمی‌دانید چگونه باید gcc را در سیستم خود نصب کنید می‌توانید از طریق انجمن‌های مختلف موجود در اینترنت اقدام نمایید. کافیست در این انجمن‌ها پرسش خود را بپرسید تا کسانی اطلاعاتی در این زمینه دارند به شما کمک کنند. (برای نمونه می‌توانید به انجمن‌های سایت گنوایران. ارگ مراجعه کنید.)

1.4 معرفی کامپایلر gcc

بسیار خوب، می خواهیم کامپایل کردن یک تکه کوچک کد C را از طریق gcc به شما نشان دهیم. اینجا هدف آموزش نحوه استفاده از gcc است بنابر این در این بخش به شرح سازوکار کد C نمی‌پردازیم. در نمونه ۱.۳ کوچکترین قطعه کد C که gcc قادر به کامپایل آن است را می بینید. این کد هیچ کاری انجام نمی دهد.

نمونه tiny.c 1.3

```
Main()
{
}
```

این قطعه کد را در ویرایشگر متن خود تایپ و با نام tiny.c ذخیره کنید. شما می توانید هر نام دیگری را نیز که به پسوند c. ختم می شود انتخاب کنید. این پسوندی است که برای سورس کدهای نوشته شده با C استفاده می شود. gcc برای کامپایل یک برنامه صحت این پسوند را چک می‌کند. با در اختیار داشتن یک فایل ذخیره شده حالا می توانید به وسیله دستور زیر آنرا کامپایل کنید:

کامپایل برنامه

```
ciaran@pooh:~/$ gcc tiny.c $ciaran@pooh:~/
```

در صورتیکه کهاین فرمان با موفقیت انجام شود بدون پیغام خروجی خواهد بود. در صورتیکه هر گونه پیغام خطایی را دریافت کردید باید صحت برنامه خود را چک کنید. در صورتیکه فرض کنیم شما این برنامه هشت کاراکتری را درست نوشته باشید باید فایلی به نام a.out در دایرکتوری شما ایجاد شود. این برنامه زبان ماشینی است که از کدهای بالا ساخته شده است. در صورتیکه آنرا اجرا کنید خواهید دید که در واقع هیچ فعالیتی انجام نمی دهد. نامگذاریa.out دلایل تاریخی دارد، اما مخفف output Assembler می‌باشد.

اگر چه gcc این کدها را کامپایل کرد اما این کدها کامل نیستند اگر شما اخطارهای gcc را فعال کنید به شما گفته خواهد شد چه کمبودهایی وجود دارد. با اضافه کردن گزینه‌ی Wall- به فرمان کامپایل می‌توانیماخطارهای gcc را فعال می کنیم:

فعال کردن نمایش اخطارها

```
ciaran@pooh:~/$ gcc -Wall tiny.c tiny.c:2: warning: return-type defaults to `int'
tiny.c: In function `main': tiny.c:3: warning: control reaches end of non-void
function ciaran@pooh:~/$
```

دلیل نمایش داده شدن این پیغام‌ها کامل نبودن برنامه‌ی نوشته شده توسز ماست. برای رهایی از این پیغام‌ها باید با افزودن سازوکارهای جدید برنامه‌ای کامل‌تر ایجاد کنیم. در اینجا با افزودن دو سطر جدید به برنامه، آن را به کوتاهترین برنامه معتبر C تبدیل می‌کنیم:

نمونه tiny2.c 2.3

```
int main()
{
return 0;
}
```

حالا برنامه را با گزینهWall- کامپایل می کنیم و شاهد این خواهید بود که پیام خطایی نمایش داده نمی‌شود. گزینه‌ی مفید دیگر o filename- است که نام فایل خروجی را (به جای a.out) مشخص می‌کند.

```
ciaran@pooh:~/$ gcc -Wall -o tiny-program tiny2.c ciaran@pooh:~/$ ls tiny2.c
tiny-program ciaran@pooh:~/$ ./tiny-program ciaran@pooh:~/$
```

1.5 چه کسی c معتبر را تعریف می کند؟

برای شما به عنوان یک برنامه نویس C معتبر توسط کامپایلر تعریف می شود. لهجه‌های مختلف زیادی از C وجود دارد. (یا بهتر است بگوییم انواع مختلفی از زبان C موجود هستند.) خوشبختانه اغلب این انواع بسیار به یکدیگر شباهت دارند. همچنین زبانهای دیگری نیز وجود دارند که بر پایه C ساخته شده اند مانند Objective C و C++. این زبان‌ها در ظاهر بسیار شبیه C هستند اما کاربردی متمایز دارند. gcc خیلی از لهجه‌های زبان C را به خوبی خیلی از زبان‌های دیگر از قبیل C++ و Obcetive C می فهمد. (انواع C در IBM)

Keringhan 1.5.1 و Ritchie

زبان C توسط Dennis Ritchie بین سالهای ۱۹۶۹ تا ۱۹۷۳ ابداع شد؛ در سال ۱۹۷۸ دنیس ریچی به همراه Brian Kernighan یک کتاب آموزشی بسیار خوب برای C با نام " The C programming language"منتشر کرد. این کتاب در واقع اولین تعریف رسمی از این زبان بود و از آنجا که نوع اصلی C محسوب می شود گاهی اوقات به آن C سنتی می گویند. متاسفانه‌این کتاب بسیاری از جنبه‌های این زبان را تعریف نشده باقی گذاشت. بنابر این افرادی که کامپایلرها را می نوشتند مجبور بودند خود در مورد چگونگی رفتار با این جنبه‌ها تصمیم گیری کنند. نتیجه‌ی این رفتار وابستگی شدید C و واکنش آن به کدها به نوع کامپایلر مورد استفاده بود. ممکن بود کدی که در یک کامپایلر نتیجه‌ای مطلوب می‌دهد در کامپایلر دیگری نتایج متفاوتی را در بر داشته باشد. استفاده از این لهجه مدت زیادی طول نکشید. gcc تنها برای کامپایل برنامه‌های خیلی قدیمی از آن پشتیبانی می کند و ما تنها به خاطر مقاصد تاریخی به آن اشاره کردیم.

ISO C 1.5.2نسخه‌ی

در سال ۱۹۸۳ موسسه استاندارد ملی آمریکا (ANSI) کمیته‌ای را به منظور تدوین یک استاندارد صحیح و رفع برخی کاستی‌ها ی زبان C تشکیل داد. کار تدوین این استاندارد در سال ۱۹۸۹ به پایان رسید و مورد پذیرش سازمان بین المللی استاندارد یا (ISO) نیز قرار گرفت. این لهجه جدید با نام C89 شناخته شد. همچنین از آن به عنوان ISO C و ANSI C نیز یاد می شود. gcc یکی از سازگارترین کامپایلرهای موجود با این لهجه است.

C99 1.5.3

کمیته ANSI C جلسات بسیار کمی را برای ارتقای این زبان تشکیل می دهد. آخرین استاندارد ارتقا یافته در سال ۱۹۹۹ عر ضه شد که با نام C99 شناخته می شود. تاکنون کامپایلرهای بسیار کمی به طور کامل از C99 پشتیبانی می کنند زیرا ایجاد تغییرات در یکی از مهمترین نرم افزارهای سیستم عامل کار زمان گیری است. تا زمان نگارش این مقاله پشتیبانی gcc از C99 تا حد قابل توجهی کامل شده است اما توسعه دهندگان هنوز مشغول کار بر روی آن هستند.

GNU C 1.5.4

GNU C بسیار شبیه C89 است اما دارای برخی خصیصه‌های جدید از C99 و برخی امکانات فرعی اضافه (extension) دیگر است. امکانات فرعی برای رفع برخی اشکالاتی که C99 راه حل مناسبی برای آنها ارایه نکرده توسط توسعه دهندگان ارایه شده اند. GNU C نوع پیش فرض gcc و نیز لهجه‌ایست که ما در این مقاله به کار خواهیم برد. ما نهایت تلاش خود را خواهیم کرد تا در مواقعی که از امکانات فرعی GNU C استفاده می کنیم آنها را مشخص کنیم هر چند به طور کلی استفاده کامل از GNU C بهتر است زیرا استفاده از ISO C باعث محدود شدن برنامه‌های شما به استفاده از جنبه‌های مشترک می گرددو کابرد آنها را محدود به موارد معینی می سازد.

1.5.5 انتخاب یک نوع C

در صورتیکه می‌خواهید لهجه‌ای غیر از لهجه پیش فرض انتخاب کنید می توانید با گزینه‌ی =std– انتخاب خود را مشخص کنید. به دنبال این گزینه می‌توانید یکی از گزینه‌های c89، GNU89 c99 و GNU99 را وارد کنید. GNU89 در حال حاظر گزینه پیش فرض است تا زمانی که پشتیبانی از c99 تکمیل شود. در آن صورت GNU99 گزینه پیش فرض خواهد شد. در هر صورت تغییرات چندان قابل توجه نخواهد بود.

1.5.6 آینده استانداردها

امکانات فرعی اضافه از قبیل آنهایی که توسط gcc اضافه می شوند منابع الهام اصلی ISO برای استانداردهای جدید C هستند. زمانی که گروه ANSI C مشاهده می کنند تعداد زیادی از کامپایلرها یکی از امکانات فرعی اضافه را به کار می گیرند آنها لزوم وجود آن امکان را بررسی می نمایند و در صورتیکه آنرا مفید تشخیص دهند یک را استاندارد برای به کار گیری آن ارایه می دهند. برخی از امکانات فرعی اضافه شده توسط gcc ممکن است در آینده به صورت استاندارد در آیند و برخی دیگر ممکن است اینگونه نباشند.

1.6 خاتمه

در این جا معرفی ما خاتمه می یابد. با امید به‌اینکه دید صحیحی درباره زبان برنامه نویسی C بدست آورده باشید، دربخش بعدی به نوشتن برنامه‌های پایه‌ای تشریح نحوه عمل کرد این برنامه‌ها خواهیم پرداخت.

بخش دوم : توابع در زبان C

2.1 توابع چه هستند؟

توابع بلوک‌های ساختاری برنامه‌های C هستند. اکثریت برنامه‌های C از بلوک‌هایی از کد که تابع(function) نامیده می شوند ساخته می شوند. وقتی یک برنامه را می نویسید شما توابع زیادی را برای انجام وظایفی که احتیاج دارید ایجاد خواهید کرد. علاوه بر این تعداد زیادی از وظایف عمومی از قبیل نمایش یک متن بر روی صفحه نمایش وجود دارند که مورد نیاز اکثر برنامه نویسان هستند. به جای اینکه هر برنامه نویس مجبور باشد در این زمینه سرمایه گذاری مجددی صورت دهد سیستم‌های GNU به همراه کتابخانه(library)هایی از توابع پیش ساخته برای انجام بسیاری از این وظایف عرضه می شوند. در طول سالها هزاران مورد از چنین توابعی بر روی هم جمع شده اند. در غیر این صورت اگر شما قصد نوشتن برنامه‌ای برای بازی BINGO داشتید مجبور بودید شخصا توابع مخصوص برنامه را بنویسید امااکنون در می یابید که افراد دیگر توابعی برای تولید اعداد تصادفی – نمایش اعداد بر روی صفحه نمایش – دریافت اعداد ورودی از بازیکن و موارد دیگر نوشته اند.

هر برنامه C ملزم به داشتن تابعی به نام ()main است. اجرای برنامه از این تابع شروع می شود. می توان تمامی کدهای برنامه را در تابع ()main قرار داد اما روش مرسوم تر تقسیم برنامه به چندین تابع کوچک است. به عنوان اولین قطعه کد مفید در این مقاله به یک نمونه کلاسیک نگاه می اندازیم. سپس از کامپایل و اجرا برنامه یک پیام ساده روی صفحه نمایش شما چاپ می‌کند. برنامه تابعی به نام ()main را تعریف (define) و تابعی به نام ()printf را فراخوانی (call) می‌کند. ()prinf تابعی است که توسط کتابخانه ابزار ورودی/خروجی استاندارد (Standard Device Input/Output library) فراهم می شود که‌این کتابخانه به همراه هر سیستم GNU وجود دارد. متن برنامه کوچک ما به صورت زیر است:

نمونه 2.1 hello.c

```
#include

int
main()
{
  printf("hello, world");

  return 0;
}
```

حالا برنامه را کامپایل و اجرا کنید. اگر همه چیز درست باشد عبارت "hello, world" بر روی ترمینال شما (ابزار خروجی استاندارد) نمایش داده خواهد شد. در صورتیکه فرامین مربوط به کامپایل و اجرای برنامه را فراموش کرده‌اید آنها به صورت زیر هستند :

فرامین کامپایل و اجرای برنامه

```
ciaran@pooh:~/$ gcc -Wall -o hello hello.c ciaran@pooh:~/$ ./hello hello, world ciaran@pooh:~/$
```

اگر شما هرگونه پیام خطا و یا اخطار را مشاهده می کنید دقت کنید که کدهای شما دقیقا مطابق کدهای این کتاب باشد. هر پیام خطایی که دریافت کنید سطری را که در آن مرتکب اشتباه شده‌اید به شما اعلام می‌کند. در صورتیکه کدها را صحیح تایپ کرده باشید چنین پیام‌هایی را دریافت نخواهید کرد.

2.2 یک تشریح سطر به سطر

حالا به طور سریع کاری را که هر سطر انجام می دهد تشریح می کنیم.در صورتیکه برخی از سطرها برای شما مبهم است نگران نباشید. در آینده نمونه‌های بسیار زیادی را انجام خواهیم داد. سطر

```
<include></include>
```

این سطر به gcc می گوید از کتابخانه ابزار ورودی/خروجی استاندارد اطلاعاتی را در مورد چگونگی استفاده از توابع برداشت کند. به طور معمول ابزار ورودی استاندارد صفحه کلید و ابزار خروجی استاندارد صفحه نمایش است. این کتابخانه به طور گسترده مورد استفاده قرار می گیرد و ما در این مقاله با توابع بسیار زیادی از آن سروکار خواهیم داشت.

```
int main()
```

این دو سطر تعریف تابع main() را آغاز می کنند. در مورد سطر اول از این دو سطر بعدا توضیحات بیشتری خواهیم داد.

{

آکولاد باز نشانه شروع یک بلوک کد است. تمام کدهای بین آکولاد باز و بسته جزیی از تابع main() هستند.

printf("hello, world");

این سطر در واقع یک فراخوانی تابع (function call) است. تابع از قبل برای شما تعریف شده است. وقتی شما تابع printf() را فرا می خوانید باید یک نشانوند(Argument)برای آن ذکر کنید تا به آن بگویید چه چیزی را نمایش دهد.

return 0;

دستور return به اجرای تابع main() خاتمه می دهد. هیچ دستور دیگری بعد از این سطر اجرا نخواهد شد. با پایان یافتن تابع main() برنامه شما خاتمه می یابد. زمانی که یک تابع به پایان می رسد می تواند مقداری را به محلی که از آنجا فراخوانی شده بازگرداند. این کار با قراردادن یک مقدار پس از return حاصل می شود.تابع main() همواره یک مقدار صحیح (یک عدد مثبت یا منفی بدون اعشار) را باز می گرداند.ما با مقدم ساختن تعریف main() با int به کامپایلر می فهمانیم که منتظر چنین چیزی باشد. قرار بر این است که درصورت اجرای بدون اشکال تابع main() مقدار بازگشتی برابر صفر باشد.

}

آکولاد بسته نشانه پایان بلوک کدی است که تابع main() را می سازد. دو سطری که بدنه تابع main() را می سازند با نام تقریر (statement) شناخته می شوند. به صورت دقیقتر این‌ها تقریرات ساده (simple statements) هستند (در مقابل تقریرات مرکب (compound statements) که در بخش چهار به آنها می رسیم). (لینک به بخش چهار-- یادداشت ٤) تقریرات در زبان C به منزله جملات برای زبانهای گفتاری هستند. در انتهای هر تقریر ساده یک نقطه–ویرگول (semi-colon) قرار می گیرد. اضافه نمودن خطوط خالی در برنامه اختیاری است.C هیچگونه نیازی به آنها ندارد اما استفاده از آنها باعث خوانایی برنامه می شود. متذکر شدیم که تابع main() ما مقدار صفر را باز می گرداند. اغلب توابع مقدار بازگشتی خود را به برنامه بازمی گردانند در حالیکه تابع main() مقدار بازگشتی خود را که نشانه پایان برنامه است

برای پوسته ارسال می‌کند. مقدار بازگشتی برنامه توسط پوسته ذخیره می شود. درصورتیکه مایل به مشاهده آن هستید از دستورات زیر استفاده کنید:

```
ciaran@pooh:~/$ gcc -Wall -o hello hello.c
ciaran@pooh:~/$ ./hello hello, world
ciaran@pooh:~/$ echo $?
0
ciaran@pooh:~/$
```

2.3 **توضیحات**

توضیحات یا Commentها راهی برای اضافه نمودن متون توضیحی در برنامه هستند. آنها توسط کامپایلر نادیده گرفته می شوند بنابراین هیچ تاثیری بر روی برنامه شما ندارند. هر چنداین کار حجم برنامه شما را افزایش می دهد اما شما پی خواهید برد که استفاده از Commentها وسیله مفیدی است تا به شما یادآوری کند مشغول انجام چه کاری هستید.در نمونه‌های این کتاب ما Commentها را برای توضیح آنچه در حال انجام است به کار می بریم.با دو روش میتوانید یک Comment را وارد برنامه خود سازید:عادی ترین روش اینست که در شروع و انتهای متن توضیحی خود به ترتیب از /* و */ استفاده کنید. توضیحات در این روش می توانند از چندین سطر تشکیل شوند. روش دوم استفاده از // درابتدای سطر است که محتویات بعد از خود تا پایان سطر جاری را به متن توضیحی مبدل می سازد.

در این جا برنامه hello, world ما به همراه متون توضیحی آورده شده است:

نمونه 2.2 hello2.c

```
/*The purpose of this program is to
 * display some text to the screen
 * and then exit.
```

```
/*

#include

 int
main()
{
 */  printf() displays a text string/*
  printf("hello, world");

  return 0;  //zero indicates there were no errors
}
```

پس از کامپایل از لحاظ اجرایی این کدها مشابه کدهای نمونه قبل هستند. خطوط 2 و 3 توضیحات بالا با یک ستاره آغاز می شوند . این کار اگر چه لازم نیست اما این نکته که توضیحات در چهار سطر ادامه دارد را روشن می سازد.

2.1 ساختن توابع شخصی

در نمونه قبلی ما تنها بک تابع معرفی کردیم. برای اضافه کردن توابع دیگر به طور کلی شما باید دو کار انجام دهید: ابتدا باید به همان صورت که برای تابع ()main انجام دادیم آنها را تعریف کنیم. کار دیگری که باید انجام دهیم اعلان (declare) توابع است. اعلان یک تابع به منزله‌این است که به gcc بگوییم تا منتظر آن تابع باشد. ما مجبور به اعلان تابع ()main نیستسم زیرا آن یک تابع مخصوص است و gcc می داند که باید منتظر آن باشد. نامی که به تابع می‌دهیم باید در هر دو جنبه اعلان و تعریف ظاهر شود.

نام تابع می تواند از حروف الفبا یعنی کاراکترهای A تا Z و a تا z و نیز ارقام 0 تا 9 و کاراکتر خط زیر(underscore) یعنی "_" تشکیل شود. این کاراکترها به هر ترتیبی می توانند کنار هم قرار بگیرند اما

حرف نخست نام تابع نمی تواند عدد باشد. زبان C حساس به بزرگی و کوچکی متن (case-sensitive) است بنابرین My_Function کاملا با my_function متمایز است. نام هر تابع باید متمایز بوده و طول آن از یک تا 36 کاراکتر مجاز است. به همراه نام باید به هرتابع یک نوع و یک بلوک کد اختصاص دهید. نوع تابع برای کامپایلر مشخص می سازد که مقدار بازگشتی تابع چگونه است. تابع می تواند بدون مقدار بازگشتی باشد. مقدار بازگشتی تابع printf() یک عدد صحیح و برابر تعداد کاراکترهای است که بر روی ترمینال نمایش داده است. این اطلاعات برای ما مهم نیستند بنابرین آنها را در برنامه خود نادیده می گیریم. در بخش بعدی در مورد جزییات انواع داده‌ای صحبت خواهیم کرد و سپس به تفصیل مقادیر بازگشتی توابع خواهیم پرداخت.

در اینجا برنامه‌ای را داریم که ۳ تابع را تعریف می کند:

نمونه three_functions.c 2.3

```
#include
 */function declarations/*
int first_function(void);
int goodbye(void);
int
main()          // function definition
{
  printf("the program begins" …);
  first_function ();
  goodbye ();

  return 0;
}
```

```
int
first_function()  // function definition
{
 */  this function does nothing/*
  return 0;
}
int
goodbye()        // function definition
{
  printf("...and the program ends" .);
  return 0;
}
```

در نمونه بالا ما تابع ()first_function را نوشتیم که هیچ کاری انجام نمی دهد و نیز تابع ()goodbye که یک پیغام را در صفحه نمایش چاپ می‌کند. اعلان توابع باید قبل ار فراخوانی آنها صورت بگیرد یعنی در نمونه ما باید توابع قبل از تابع ()main اعلان می شدند. به طور کلی مرسوم است که اعلان توابع بعد از سطرهای شامل عبارت #include و قبل ار آغاز تعریف توابع صورت می گیرد.

2.5 فایل‌های چندگانه

اجباری برای اینکه برنامه‌ها تنها در یک فایل معین نوشته شوند وجود ندارد. کدهای شما میتوانند در هر تعداد فایل که بخواهید پخش شوند. به عنوان نمونه اگر برنامه‌ای از ٤٠ تابع تشکیل شده باشد شما می توانید هر تابع را در یک فایل جداگانه قرار دهید؛ اگر چه این کار یک افراط در تکه تکه کردن برنامه است. غالبا توابع بر حسب موضوع در گروه‌هایی دسته بندی شده و در فایل‌های جداگانه قرار می گیرند. برای نمونه در صورتیکه وظیفه یک برنامه محاسبه قیمت یک پیتزا و نمایش نتیجه است شما می توانید توابع مربوط به محاسبه قیمت را در یک فایل و توابع نمایش نتایج را در فایل دیگری قرار داده و از فایل سوم برای نگهداری تابع main() استفاده کنید. حال می توانید از فرمان زیر برای کامپایل برنامه استفاده کنید:

کامپایل برنامه تشکیل شده از سه فایل

```
ciaran@pooh:~/$ gcc -o pizza_program main.c prices.c display.c
```

یاد آوری می کنیم در صورتیکه تابعی را در prices.c تعریف و قصد فراخوانی آن را در main.c دارید باید آنرا در main.c اعلان کنید.

2.6 فایل‌های سرآیند

ادامه پیگیری اعلان توابع می تواند موجب شلوغ کاری شود. به همین دلیل فایل‌های سرآیند برای جا دادن کدهای C که شما قصد به کارگیری آنها در فایل‌های چندگانه را دارید استفاده می شوند. قبل از این شما به طور عملی از یک فایل سرآیند به نام stdio.h استفاده کردید، این فایل سرآیند شامل اعلان توابع بسیار زیادی است. از جمله شامل تابع printf() و نیز اعلان آن است. شما نیز می توانید یک فایل سرآیند شامل اعلان توابعی که قصد به اشتراک گذاری آنها را دارید ایجاد کنید و سپس با استفاده از #include آن را در اختیار هر فایل C که به اطلاعات آن احتیاج دارد قرار دهید. تنها تفاوت موجود در این است که شما باید به جای علائم < و > نام فایل خود را بین دو "،" یا کوتیشن مارک قرار دهید. یعنی از "my_header.h" به جای استفاده کنید. برای توضیح این نکته برنامه پیتزا که قبلا به آن اشاره کردیم را خواهیم نوشت.

2.7 یک (غیر) برنامه بزرگتر

نکات کوچک برنامه نویسی که تا کنون بررسی کردیم برای نوشتن یک برنامه آراسته و تاثیر گذار کافی نیستند. برای حفظ سادگی برنامه ما تنها یک اسکلت کلی از برنامه را می نویسیم تا بدینوسیله ساختار و روش استفاده از فایل‌های سرآیند را تشریح کنیم بدون اینکه شما در باتلاق مفاهیم جدید فرو روید. در بخش سوم مقاله یک نسخه کامل از این برنامه را خواهیم نوشت. کدهای زیر می توانند کامپایل و اجرا شود اما عملیاتی برای محاسبه قیمت و یا سوال از کاربر برای وارد کردن اطلاعات انجام نخواهد داد.

نخست با فایل main.c سرو کار داریم که تنها شامل تابع main() می شود.main() برخی از توابعی که در فایل‌های دیگر تعریف شده اند را فراخوانی خواهد نمود. متذکر می شویم که فایل main.c فاقد سطر #include می باشد زیرا از هیچ یک از توابع عضو کتابخانه ابزار ورودی/خروجی استاندارد استفاده نمی‌کند.

نمونه 2.4 main.c

```
#include "display.h"
#include "prices.h"
int
main()
{
  display_options();
  calculate_price();
  display_price();
  return 0;
}
```

در مرحله بعد به بررسی display.c می‌پردازیم. این فایل شامل دو تابع است که هر دو از طریق تابع main() فراخوانی می شوند و ما اعلان آنها را در یک فایل سرآیند با نام display.h قرار داده‌ایم. نمونه

display.c 2.5

```
#include

int
display_options()
{
  printf("Welcome to the pizza parlor");
  printf("What size pizza would you like? (in inches)");

  return 0;
}

int
display_price()
{
  printf("Your pizza will cost 0.00");

  return 0;
}
```

نمونه 2.6 display.h

```
*/header file just contains function declarations, an file that wants

* to use either of these functions just has to #include this file/*

int display_options(void);

int display_price(void);
```

و در نهایت prices.c را داریم که شامل توابعی برای دریافت داده از کاربر و محاسبه قیمت مجموع هزینه‌های پیتزا است. تنها یکی از این توابع از طریق ()main فراخوانی می شود بنابر این اعلان دو تابع دیگر در بالای همین فایل صورت می گیرد. کدهای کامل این توابع را در فصل سوم خواهیم نوشت.

نمونه 2.7 prices.c

```
int get_size(void);

int get_toppings(void);


int

calculate_price()

{

 */  insert code here.  Will call get_size() and get_toppings/* .()

  return 0;

}
```

```
int
get_size()
{
 */ insert code here/*
  return 0;
}

int get_toppings()
{
 */ insert code here/*
  return 0;
}
```

نمونه prices.h 2.8

```
int calculate_price(void);
```

حالا می توانید gcc را با دستور زیر برای کامپایل فایل‌های بالا به کار گیرید:

```
gcc -Wall -o pizza_program main.c prices.c display.c
```

این برنامه در زمان اجرا پس از نمایش یک پیغام خوش آمدگویی اعلام می کند که قیمت پیتزای شما برابر 0.00£ است.

2.8 یک تابع جدید دیگر

قبل از ادامه بیایید نگاهی به یکی دیگر از توابع عضو کتابخانه ابزارهای ورودی/خروجی استاندارد بیاندازیم: printf(). دستور چاپ قالبدار یکی از شکل‌های پیشرفته تابع printf() است. رشته‌ای که برای prinf() ارسال می شود می تواند شامل کاراکترهای کنترلی که دارای معانی مخصوصی هستند باشد. رفتن به سطر جدید به طور خودکار توسط printf() صورت نمی گیرد و برای اینکار باید کاراکترهای را اضافه نمایید.

2.9 خلاصه اصول اولیه

مطالبی را که تا کنون از آنها عبور کردیم نباید چندان سخت باشند. اگر قصد کسب تجربه را دارید برنامه‌هایی مشابه که دارای چندین سطر خروجی هستند ایجاد کنید. برنامه خود را به دو تابع تقسیم کرده و آنها را در دو فایل مجزا قرار دهید. همواره در هنگام کامپایل اخطارهای gcc را فعال کنید. اخطارها خبر از مبهم بودن و یا ناکامل بودن کدهای شما می دهند.در این حالت gcc معنای صحیح آنها را حدس می زند و اغلب این کار را به درستی انجام می دهد اما شما نباید به آنها اعتماد کنید. جست و جو برای تصحیح پیغام‌های خطا به مهارت شما در استفاده از زبان کمک میکند. اغلب پیغام‌های خطا به همراه شماره سطری که مشکل در آن وجود دارد می آیند. اگر شما در آن سطر هیچگونه خطایی مشاهده نمی کنید سطر فوقانی آنرا چک کنید. در صورتیکه یک دستورالعمل ناکامل باشد gcc تا زمانی که به ابتدای دستورالعمل بعدی نرسیده متوجه وجود خطا نخواهد شد.نقطه ویرگول‌های خود را فراموش نکنید.

بخش سوم: داده‌ها و عبارت‌ها

برنامه‌های مفید واقعی شامل داده‌ها، انجام عملیات بر روی آنها و نمایش آنها در خروجی هستند.در زبان C شما از قطعات نامگذاری شده حافظه که متغیر (variables) نامیده می شود برای نگهداری داده‌ها استفاده می کنید. برنامه‌های C می توانند با کمک نام متغیرها داده‌های درون آنها را در هر زمان تغییر دهند. هر متغییر یک نام منحصر به فرد (identifier) دارد که شما برای استفاده یا تغییر مقدارش به آن رجوع می

کنید. یک عبارت (expression) هر چیزیست که بتواند مورد ارزیابی قرار گیرد. برای نمونه 1+1 عبارتی با مقدار 2 است. در این عبارت علامت بعلاوه یک عملگر دوتایی (binary operator) است زیرا بر روی دو مقدار(value) برای ساخت یک مقدار عمل می‌کند. قواعد نامگذاری متغیرها مشابه نامگذاری توابع است.شما می توانید از اعداد،ارقام و کاراکتر زیر خط در نامگذاری آنها استفاده کنید به شرطی که کاراکتر اول عدد نباشد. همچنین به مانند توابع متغیرها نیز باید قبل از به کارگیری اعلان شوند. نامیکه برای متغیر در نظر می گیرید بهتر است گویای هدفی باشد که متغیر را برای آن به کار می گیریم. این عمل مطالعه کدهای شما را آسانتر می سازد. شما میتوانید متغیرهای خود را شخصا تعریف کرده و یا از انواعی (types) از قبل تعریف شده اند استفاده کنید.

قبل از اینکه در باتلاق اصطلاحات فنی فرو رویم بیایید با هم نگاهی به یک قطعه کد کوچک بیاندازیم تا نشان دهیم همه‌اینها چقدر ساده است. در این نمونه ما دو متغییر از نوع پیش تعریف شده int را به کار می گیریم: نمونه bicycles.c 3.1

```
include#

int
main()
{
  int number_of_bicycles;
  int number_of_wheels;

  number_of_bicycles = 6;
  number_of_wheels = number_of_bicycles * 2;

printf("I have %d bicycles ", number_of_bicycles);
```

```
  printf("So I have %d wheels ", number_of_wheels);

  return 0;
}
```

3.1 تشریح برنامه دوچرخه

مطالب جدید کمی برای بررسی وجود دارد که برای تشریح آنها برنامه را به چند بخش تقسیم می کنیم :

```
int number_of_bicycles;
int number_of_wheels;
```

هر یک از این دو سطر یک متغیر را تعریف می‌کند. Int یکی از انواع داده است که به صورت توکار در زبان C وجود دارد. متغییرهایی از نوع int توانایی نگهداری مقادیر مثبت و یا منفی صحیح را دارا هستند.

```
number_of_bicycles = 6;
```

این سطر مقدار 6 را در متغییر number_of_bicycles ذخیره می‌کند. علامت مساوی به عنوان عملگر تخصیص (the assignment operator) شناخته می شود. این عملگر مقدار عبارت سمت راست خود را به به متغییر سمت چپش اختصاص می دهد.

```
number_of_wheels = number_of_bicycles * 2;
```

در این سطر علاوه بر عملگر تخصیص از عملگر ضرب (multiplication operator) نیز استفاده شده است. ستاره (asterisk) نیز یک عملگر دوتایی است زیرا دو مقدار را برای ایجاد یک مقدار در هم ضرب

می‌کند. در اینجا این عملگر مقدار 12 را ایجاد می کند که درون متغیر number_of_wheels ذخیره می شود.

```
printf("I have %d bicycles ", number_of_bicycles);
printf("So I have %d wheels ", number_of_wheels);
```

در اینجا نیز باز با ()printf سر و کار داریم که البته در شکلی متفاوت از آنچه قبلا دیده بودیم به کار رفته است. در این نمونه ()printf دو نشانوند (arguments) پذیرفته است که توسط یک کاما یا ویرگول از یکدیگر جدا شده اند. نشانوند اول ()printf به عنوان رشته قالب (format string) شناخته می شود. وقتی یک d٪ در رشته قالب وجود داشته باشد ()printf متوجه می شود که باید منتظر یک نشانوند اضافی باشد تا مقدار آن جایگزین d٪ گردد. به ازای هر d٪ باید یک نشانوند اضافی وجود داشته باشد.

با دانستن این مطلب جدید از اینکه پس از کامپایل و اجرای این قطعه کد سطور زیر نمایش داده شود غافلگیر نخواهید شد :

خروجی برنامه

```
I have 6 bicycles
So I have 12 wheels
```

مثل همیشه در صورتیکه برخی قسمت‌ها خوب متوجه نشده‌اید نگران نباشید زیرا نمونه‌های بیشتری را انجام خواهیم داد.

3.2 انواع داده‌ای

تمام انواع داده‌ای که در C تعریف شده اند از قسمت‌هایی از حافظه که بایت نامیده می شوند ساخته شده اند. در اکثر معماری‌های کامپیوتر یک بایت از هشت بیت تشکیل شده است. هر بیت یک مقدار صفر یا یک را نگهداری می‌کند. هشت بیت که هر کدام دارای دو حالت باشند در کنار هم ۲۵٦ ترکیب مختلف را ایجاد می کنند. (۲ به توان ۸ حالت) بنابر این یک مقدار صحیح که از دو بایت تشکیل شده است می توانند عددی بین ۰ تا ٦٥٥۳٥ را در خود نگه دارد. (از صفر تا ۲ به توان ۱٦) اما معمولا متغییرهای نوع صحیح از اولین بیت خود برای نگهداری علامت مثبت و یا منفی عدد استفاده می کنند بنابراین می توانند عددی بین مثبت و منفی ۳۲۷٦۸ را در خود نگه دارند. اگر چنانچه‌این متغییرها از نوع بدون علامت باشند تنها قادر به نگهداری اعداد بزرگتر یا مساوی صفر خواهند بود. نوع داده‌ای متغییرهای بدون علامت در ابتدا دارای عبارت unsigned می باشد.

همانطور که قبلا اشاره کردیم ۸ نوع داده پایه‌ای در زبان C تعریف شده اند. ٥ نوع برای نگهداری اعداد صحیح با مقادیر گوناگون و ۳ نوع برای نگهداری مقادیر گوناگون اعداد با ممیز شناور. (اعداد اعشاری) زبان C هیچگونه نوع داده پایه‌ای را برای متن فراهم نکرده است. متن‌ها از کاراکترهای تکی ساخته می‌شوند و کاراکترها توسط اعداد نمایندگی می‌شوند. در نمونه قبلی از یکی از انواع داده‌ای به نام int استفاده کردیم که دارای بیشترین کاربرد در زبان C است. بیشتر اعدادی که در برنامه‌های کامپیوتری استفاده می‌شوند اعداد صحیح هستند. ما کمی بعد درباره انواع با ممیز شناور صحبت خواهیم کرد. به ترتیب اندازه مقادیر صحیح علامت دار و بدون علامت از کوچک به بزرگ عبارتند از:

ترتیب	نوع داده ای	اندازه در سیستم 32 بیتی	محدوده اعداد قابل نگهداری در سیستم 32 بیتی
1	char	8	127تا -127

2	unsigned char	8	0 تا255
3	short	16	-32,767 تا32,767
4	unsigned short	16	0 تا65,535
5	int	32	-2,147,483,647 تا 2,147,483,647
6	unsigned int	32	0 تا 4,294,967,295
7	long	32	-2,147,483,647 تا 2,147,483,647
8	unsigned long	32	0 تا 4,294,967,295
9	long long	64	-9,223,372,036,854,775,807 تا 9,223,372,036,854,775,807
10	unsigned long long	64	0 تا 18,446,744,073,709,551,615

انواع داده‌ای کوچکتر دارای این مزیت هستند که مقدار کمتری از حافظه را اشغال می‌کنند. انواع داده‌ای بزرگ باعث ایجاد خطای اجرایی (performance penalty) می شوند.متغییرهایی از نوع داده‌ای int بزرگترین عدد صحیح ممکن را که باعث ایجاد خطای اجرایی (performance penalty) نمی‌شود نگهداری می‌کنند به همین دلیل مقدار متغییرهای نوع Int بسته به نوع کامپیوتری که شما استفاده می‌کنید متفاوت است. نوع داده‌ای char معمولا یک بایت است. وجه تسمیه آن به‌این دلیل است که‌این نوع داده‌ای معمولا برای نگهداری کاراکترهای تنها استفاده می شود. اندازه سایر انواع داده‌ای به نوع سخت افزار کامپیوتری که شما استفاده می‌کنید بستگی دارد. اغلب رایانه‌ای رومیزی 32 بیتی هستند که اشاره به اندازه داده‌ای که آنها برای پردازش آن طراحی شده اند دارد.در رایانه‌های 32 بیتی نوع داده‌ای int از 4 بایت تشکیل شده است (2 به توان 32 حالت).نوع داده‌ای short معمولا کوچکتر از int و نوع داده‌ای long از

نوع داده‌ای int بزرگتر و یا با آن مساوی است و در نهایت نوع داده‌ای long long برای نگهداری مقادیر عددی خیلی خیلی بزرگ است.

نوع داده‌ای متغییری که شما استفاده می‌کنید تاثیر زیادی بر روی کاهش سرعت اجرا یا اشغال حافظه برنامه شما ندارد.جز در موارد خاص شما می توانید تنها از متغییرهای نوع int استفاده کنید. در دهه گذشته اغلب رایانه‌ها پردازنده‌های 16 بیتی داشتند که اندازه متغییرهای نوع Int را به 2 بایت محدود می کرد. در حال حاضر متغییرهای از نوع دادهای short از 2 بایت و متغییرهای از نوع داده‌ای long از 4 بایت تشکیل شده اند. در حال حاضر با پردازنده‌های 32 بیتی نوع داده‌ای پیش فرض یعنی int به اندازه کافی برای نگهداری متغییرهای که سابقا از نوع داده‌ای long برای نگهداری آنها استفاده می شد گنجایش دارد. برای اطلاع از اندازه هر یک از انواع داده‌ای بر روی رایانه خود قطعه کد زیر را کامپایل و اجرا کنید. در این برنامه از عملگر جدیدی به نام ()sizeof تشکیل شده که مقدار حافظه‌ای را که توسط هر نوع داده‌ای اشغال می گردد بیان می کند. نمونه sizeof_types.c 3.2

```
int
main()
{
  printf("sizeof(char) == %d\n", sizeof(char));
  printf("sizeof(short) == %d\n", sizeof(short));
  printf("sizeof(int) == %d\n", sizeof(int));
  printf("sizeof(long) == %d\n", sizeof(long));
  printf("sizeof(long long) == %d\n", sizeof(long long));

  return 0;
```

}

3.3 نمونه دیگری از انتساب

وقت آن رسیده که نمونه دیگری را بررسی کنیم. در این نمونه هم چند مطلب تازه وجود دارد که در یک دقیقه آنرا توضیح خواهیم داد.

نمونه 3.3 displaying_variables.c

```
#include

int
main()
{
  short first_number = -5;
  long second_number, third_number;

  second_number = 20000 + 10000;

  printf("the value of first_number is %hd\n", first_number);
  printf("the value of second_number is %ld\n", second_number);
  printf("the value of third_number is %ld\n", third_number);
  return 0;
}
```

در نمونه بالا از یک متغییر نوع short و دو متغییر نوع long استفاده کرده‌ایم. می توانستیم هر سه متغییر را از نوع int در نظر بگیریم اما از انواع داده‌ای دیگر استفاده کردیم تا نشان دهیم این انواع داده‌ای چقدر به هم شبیه اند. در نخستین سطر تابع main ما متغییری را اعلان و همزمان آن را مقدار دهی کرده‌ایم که این کار بسیار متداول است. در سطر دوم دو متغییر را با جدا کردن آنها از هم به وسیله کاما اعلان نموده‌ایم. این کار ممکن است نشانه مهارت ما باشد ولی بهتر آنست که هر متغییر را در سطری جدا گانه اعلان کنیم تا به خوانا بودن برنامه کمک شده باشد. سطر سوم بسیار شبیه به برخی کدهای نمونه اول است. عملگر جمع مقدار 30000 را تولید و سپس این مقدار در متغییر second_number قرار می گیرد. آخرین مطلب قابل اشاره‌اینست که در رشته قالب بندی تابع ()printf به جای ٪d از ٪hd برای متغییرهای از نوع داده‌ای short و از ٪ld برای متغییرهای از نوع داده‌ای long استفاده شده است. به‌این عبارت‌های کوچک کاراکتری اصطلاحا شاخص‌های تبدیل (conversion specifiers) گفته می شود. هر نوع از انواع داده‌ای نصریح کننده تبدیل مختص خود را دارد. در صورتیکه خواسته باشید کاراکتر درصد (٪) را چاپ کنید باید از ٪٪ استفاده نمایید. هنگامییکه برنامه بالا را کامپایل و اجرا کنید مقدار متغییرهای خود را مشاهده خواهید کرد. مقدار متغییر third_number عجیب خواهد بود زیرا در طول برنامه هیچ مقداری به آن اختصاص داده نشده است. وقتی که شما یک متغییر را اعلان می نمایید سیستم عامل مقداری از حافظه را به آن اختصاص می دهد. شما راهی برای فهمیدن اینکه‌این قسمت از حافظه قبلا برای چه کاری استفاده شده نخواهید داشت .تا زمانیکه مقداری را به متغییر خود انتساب نداده‌اید مقدار ی که از قبل در خانه‌های حافظه اختصاص یافته به متغییر وجود دارد به متغییر اختصاص خواهد داشت. بنابر این مقدار متغییرها تا قبل از انتساب مقدار مشخص به انها ضرورتا عددی تصادفی و غیر قابل پیش بینی خواهد بود. فراموش کردن انتساب مقدار به متغییرها یک اشتباه رایج در میان برنامه نویسان مبتدی است.

3.4 توضیح سریع تابع ()printf

شما احتمالا متوجه عبارت دو حرفی \n در مواقعی که از تابع ()printf استفاده می کنیم شده‌اید. این عبارت در خروجی چاپ نمی‌شود بلکه ()printf را متوجه می کند که باید به سطر بعد برود. به طور کلی اسلاش وارونه (\) کاراکتر گریز (escape character) زبان سی محسوب می شود و هرگاه در داخل 2 کوتیشن مارک قرار گیرد کاراکتر بعد از آن معنای مخصوصی را خواهد داد.نمونه دیگر \t است که به اندازه یک tab فاصله خالی در خروجی ایجاد می کند. کاراکتر مخصوص دیگری که ()printf آنرا مراعات می

کند درصد (٪) است که از ()printf میخواهد کاراکترهای بعد از خود را در هنگام چاپ با مقدار یک متغییر مشخص جایگزین نماید. d٪ رشته کاراکتری است که یک متغییر از نوع داده‌ای Int را برای چاپ با استفاده از سیستم شمارش بر مبنای ده یا ده‌دهی نمایندگی می کند. به ازای هر d٪ در رشته قالب بندی شما باید به تابع ()printf بفهمانید که چه متغییری را قصد جایگزین کردن با آن دارید. در نمونه زیر برخی استفاده‌های بیشتر تابع ()printf را مشاهده می‌کنید:

نمونه 3.4 more_printf.c

```
Int
main()
{
  int one = 1;
  int two = 2;
  int three = 4;  /* the values are unimportant here/*

  printf( "one ==\t%d\ntwo ==\t%d\nthree ==\t%d\n", one, two, three );

  return 0;
}
```

3.5 عملیات ریاضی ساده

در ابتدای این فصل اشاره کردیم که یک برنامه معمولا با انجام عملیات بر روی داده‌ها سر و کار دارد. با به کار گیری علایم استاندارد ریاضی انجام عملیات ریاضی در زبان c به صورت بسیار آسان قابل فهم و خواندن است. به نمونه زیر توجه کنید:

نمونه wages.c 3.5

```
int
main()
{
  int hours_per_day;
  int days_per_week;

  hours_per_day = 8;
  days_per_week = 5;

  printf("I work %d hours a week.\n", (days_per_week * hours_per_day));

  printf("%d %d hour days\n", days_per_week, hours_per_day);

  return 0;
}
```

3.6 متغییرهای عمومی و محلی

متغییرهایی که تا به حال از آنها استفاده کردیم متغییرهای محلی (local variables) بودند این متغییرها در همان تابعی که قصد استفاده از آنها را داریم تعریف شده پس از اجرای تابع نابود می شوند و همانطور که گفتیم تا قبل از انتساب مقدار آنها غیر قابل پیش بینی است. دسته دیگری از متغییرها که متغییرهای عمومی

(global variables) نام دارند در خارج از توابع و معمولا بالای تابع ()main تعریف می شوند. این متغییرها از داخل تمام توابعی بعد از اعلان آنها تعریف شوند قابل استفاده بوده در ابتدای اجرای برنامه‌ایجاد و تا پایان اجرای برنامه در حافظه باقی می مانند. بر خلاف متغییرهای محلی به محض اعلان این متغییرها مقدار اولیه صفر به آنها تعلق می گیرد.

نمونه 3.6 global_variable.c

```
#include
int global_varialble;
int first_function(void;(
int
main()
{
  printf("The first value of global_varialbles is %d \n",global_varialble);
  first_function();
  return 0;
}
int
first_function ()
{
global_varialble = 5;
printf("Now the value of global_varialble is %d \n",global_varialble;(
  return 0;
}
```

3.7 حوضه متغییر

منظور از حوضه (scope) یک متغییر محدوده‌ایست که متغییر در آن قابل دسترسی و استفاده می باشد. به عنوان نمونه همانطور که در بالا دیدیم حوضه متغییرهای محلی همان تابعی بود که در آن تعریف شده بودند و حوضه متغییرهای عمومی تمام توابعی بود که بعد از اعلان آن متغییرها تعریف می شدند.

3.8 کلاسهای حافظه

کلاس حافظه خصوصیتی از متغییر است که طول عمر و حوضه متغییر را مشخص می کند.منظور از طول عمر زمان ایجاد و نابود شدن متغییر می باشد. کلاسهای حافظه انواع گوناگونی دارد که در این جا دو مورد از پرکاربرد ترین آنها یعنی کلاس حافظه‌ایستا (static) و کلاس حافظه خارجی (external) را توضیح می دهیم:

3.8.1 کلاس حافظه خارجی

بعضی اوقات کدهای برنامه ما در بیش از یک فایل قرار دارد.شما میتوانید از متغییرهای عمومی که در یکی از فایلها اعلان شده است در فایل دیگری استفاده کنید به شرط اینکه آن متغییر را درفایل دوم نیز اعلان نمایید. دستور اعلان متغییر در فایل دوم باید با کلمه کلیدی extern آغاز گردد. در نمونه زیر کدهای برنامه از دو فایل تشکیل شده است:

نمونه 3.7 main.c

```
int main()
{
  extern int my_var ;
my_var = 500;
 print_value();
return 0;
```

```
}
```

نمونه secondary.c 3.8

```
        # include
int my_var;
void print_value()
{
printf("my_var = %d\n", my_var);
}
```

3.8.2 کلاس حافظه‌ایستا

در زبان C هرگاه یک تابع را بیش از یک بار فراخوانی کنیم با هر بار فراخوانی تابع متغییرهای محلی تعریف شده در تابع از ابتدا ایجاد شده سپس مقدار اولیه گرفته و در انتهای اجرای تابع نابود می شوند اما اگر این متغییرها از نوع کلاس حافظه‌ایستا باشند تنها یک بار مقدار اولیه گرفته در هنگام خروج از تابع آخرین مقدار خود را حفظ کرده و تا انتهای اجرای برنامه در حافظه باقی می مانند. برای اینکه متغییرهای محلی یک تابع را از نوع ایستا تعریف کنیم باید در دستور اعلان آنها ابتدا کلمه کلیدی static را ذکر نماییم تا آن متغییر از نوع محلی ایستا تعریف شود. اما متغییرهای عمومی به طور پیش فرض از نوع ایستا بوده و ذکر یا عدم ذکر کلمه کلیدی static در ابتدای فرمان اعلان آنها بی تاثیر است. در صورتیکه متغییر محلی از نوع ایستا تعریف نشود کلاس حافظه آن از نوع اتوماتیک (automatic) خواهد بود.کلاس حافظه اتوماتیک با آوردن کلمه کلیدی auto در ابتدای دستور اعلان متغییر مشخص می شود. البته چون

متغییرهای محلی در صورتیکه کلاس حافظه دیگری برای آنها بیان نشود به طور پیش فرض از نوع اتوماتیک هستند آوردن و یا نیاوردن کلمه کلیدی autoدر ابتدای دستور اعلان آنها یکسان است.

نمونه static_variable.c 3.9

```
#include
int first_function(void;(
int
main()
{
  first_function();
  first_function();
  first_function();

  return 0;
}
int
first_function()
{
  int x=0;
  static int y=0;
  printf(“automatic x = %d   static y = %d \n”,x,y);
  x=x+1;
```

```
  y=y+1;
  return 0;
}
```

3.9 متغییرهای غیر قابل تغییر

یک عادت خوب برنامه نویس اینست که هیچ گاه عددی غیر از صفر یا یک را در خلال کدهایتان به کار نبرید!. اگر به ثابت عددی دیگری غیر از صفر و یک احتیاج داشتید آنرا تبدیل به یک متغییر از نوع غیرقابل تغییر (constant) نمایید. برای این کار در ابتدای دستور اعلان متغییر از کلمه کلیدی const استفاده نمایید. نمونه :

```
Const int my_age = 20;
```

مقدار این متغییر در تمام طول برنامه 20 باقی خواهد ماند و هر کجا که قصد تغییر آن را داشته باشید با خطای کامپایلر مواجه خواهید شد. عدد 20 معنای مختصری می دهد در حالیکه مشخصه‌ای مانند my_age اطلاعات بیشتری در مورد عملی که تابع انجام می دهد در اختیار ما قرار می دهد. مزیت دیگر استفاده از متغییرهای غیر قابل تغییر در اینست که برای تغییر مقدار ثابت عددی باید در تمام برنامه به دنبال آن ثابت گشته و آن را تغییر دهیم ولی برای تغییر مقدار متغییرهای غیر قابل تغییر تنها فرمان اعلان آنها را تغییر می دهیم.

بخش چهارم : کنترل جریان

زبان برنامه نویسی c دو سبک برای تصمیم گیری فراهم نموده است : حلقه‌ها (looping) انشعاب‌ها(Branching). انشعاب یعنی تصمیم گیری در مورد اینکه چه عملیاتی انجام شود و حلقه یعنی اینکه یک عملیات خاص چند مرتبه تکرار شود.

4.1 انشعاب

وجه تسمیه انشعاب به‌این دلیل است که برنامه در مواجه با دو انشعاب در کد خود انتخاب می کند که کدام یک را ادامه دهد. دستور if یکی از ساده ترین ساختارهای انشعاب است. این دستور از یک شرط (expression)که داخل پرانتز قرار گرفته و نیز یک دستور(statement) و یا بلوکی از دستورات(block of statements) که توسط دو آکولاد باز و بسته یعنی { و } محاصره شده اند تشکیل شده است.در صورتیکه شرط صحیح باشد (ارزشی غیر صفر داشته باشد) در آنصورت دستور و یا بلوک دستورات اجرا خواهند شد و در غیر این صورت دستورات نادیده گرفته خواهند شد. دستور if دارای یکی از دو شکل کلی زیر است:

فرم کلی اول با یک دستور

```
if (expression)
  statement;
```

فرم کلی دوم با بلوک دستورات

```
if (expression)
{
   statement1;
   statement2;
   statement3;
}
```

در زیر نمونه‌ی ساده از کاربرد دستور if را مشاهده می‌کنید: نمونه 4.1 using_if.c

```
#include

int
main()
{
  int cows = 6;

  if (cows > 1)
    printf("We have cows\n");

  if (cows > 10)
    printf("loads of them!\n");

  return 0;
}
```

کامپایل – اجرا و خروجی برنامه بالا به صورت زیر است:

```
ciaran@pooh:~/$ gcc -Wall -Werror -o cows using_if.c
ciaran@pooh:~/$ ./cows
We have cows
ciaran@pooh:~/$
```

دستور printf() دوم در کد بالا به دلیل اشتباه بودن شرط آن (با ارزش صفر) اجرا نمی گردد.

4.2 if...else

شکل دومی از دستور if نیز وجود دارد که به شما امکان می دهد تا بلوکی از کدها را مشخص نموده تا در صورت اشتباه بودن شرط اجرا شوند. این ساختار با نام دستور if ... else شهرت دارد و به وسیله قرار دادن کلمه کلیدی else ((reserved word و یک بلوک کد دیگر در انتهای ساختار معمولی if شکل می گیرد. پس ار تست شرط دستر if یکی از دو بلوک کد بسته به صحیح یا غیر صحیح بودن شرط اجرا خواهند شد. به نمونه زیر توجه کنید:

نمونه cows2.c 4.2

```
int
main()
{
  int cows = 0;

  if (cows > 1)
    {
      printf("We have cows\n");
      printf("%d cows to be precise\n", cows);
    }
  else
    {
      if (cows == 0)
        printf("We have no cows at all\n");
      else
        printf("We have only one cow\n");
  }
```

```
  if (cows > 10)
    printf("Maybe too many cows.\n");

  return 0;
}
```

شما اکنون باید قادر به حدس زدن خروجی برنامه باشید:

خروجی برنامه

```
ciaran@pooh:~/$ ./cows2
We have no cows at all
ciaran@pooh:~/$
```

در نمونه بالا یک دستور if ... else در درون یک دستور if ... else دیگر وجو داشت. این ساختار در زبان c کاملا منطقی و پرکاربرد است.

4.3 دستور switch

این ساختار انشعاب نسبت ساختارهای قبلی قدری پیچیده تر است. اگر چه دستور switch بسیار انعطاف‌پذیر است اما تنها برای تست کردن داده‌های صحیح و کاراکتری کاربرد دارد. فرم کلی آن به صورت زیر است:

فرم کلی ساختار switch

```
switch (integer or character expression)
```

```
{
  case constant1 : statement1;
  break;

  case constant2 : statement2;
  break;

  case constant3 : statement3;
  break;
}
```

با اجرای دستور switch ابتدا عبارت داخل پرانتز با تمام مقادیری که جلوی آنها عبارتهای case نوشته شده مقایسه شده و با هرکدام از آنها که برابر بود دستورات بعد از آن تا رسیدن به عبارت break اجرا می شود.سپس با اجرای دستور break کامپایلر از حلقه خارج می شود.

نمونه morse.c 4.3

```
#include

int main();
void morse (int);


int main()
{
  int digit;
```

```
  printf ("Enter any digit in the range 0 to 9" :);
  scanf ("%d", &digit);

  if ((digit < 0) || (digit > 9))
{
    printf ("Your number was not in the range 0 to 9.\n");
}
  else
{
    printf ("The Morse code of that digit is" );
    morse (digit);
}
  return 0;
}

void morse (int digit)       /* print out Morse code/*
{
  switch (digit)
  {
```

```
    case 0 : printf;("-----")
      break;
    case 1 : printf;("----.")
      break;
    case 2 : printf;("---..")
      break;
    case 3 : printf;("--...")
      break;
    case 4 : printf;("-....")
      break;
    case 5 : printf;(".....")
      break;
    case 6 : printf;("....-")
      break;
    case 7 : printf;("...--")
      break;
    case 8 : printf;("..---")
      break;
    case 9 : printf;(".----")
}
  printf ("\n\n");
```

```
}
```

نمونه fs.c 4.4

```
#include
int main()
{
  printf ("Will you join the Free Software movement" ?);
  if (yes())
  {
    printf("Great!  The price of freedom is eternal vigilance!\n\n;("
 }
  else
  {
    printf("Too bad.  Maybe next life...\n\n;("
  }

  return 0;
}


int yes()
{
```

```
  switch (getchar(()
  {
    case 'y: '
    case 'Y' : return 1;
    default  : return 0;
  }
}
```

4.3 **حلقه‌ها**

حلقه‌ها راهی را برای تکرار فرمان‌ها و نیز کنترل اینکه فرمان‌ها چند مرتبه انجام شود فراهم می‌کنند.فرضا شما قصد دارید حروف الفبا را بر روی صفحه نمایش چاپ کنید. می توانید این کار را با یک بار فراخوانی تابع ()printf انجام دهید. این یک راه حل برای انجام این کار است اما به نظر نمی رسد که بهترین راه باشد.حال بر فرض اگر خواسته باشید اعداد یک تا هزار را در یک ستون چاپ کنید چه می‌کنید؟ آیا باز هم از یک تابع ()printf بسیار طولانی استفاده می‌کنید و یا هزار بار تابع ()printf را در کدهایتان تایپ می‌کنید؟!. البته می توانید راه حل‌های ذکر شده را به کار برید ولی بهتر است انجام کارهای تکراری را به کامپیوتر سپرده و وقت خود را صرف پرداختن به قسمت‌های جذاب تر برنامه خود کنید.

4.4 حلقه while

اساسی ترین حلقه در زبان c حلقه while است.دستور while کاری شبیه تکرار دستور if را انجام می دهد. اگر شرط دستور صحیح باشد دستورات انجام می شوند با این تفاوت که پس از اجرای دستورات شرط مجددا بررسی می شود تا در صورتیکه هنوز صحیح باشد دستورات بار دیگر انجام شوند. و این حلقه تا زمانی تکرار می شود که شرط مقدار غلط پیدا کند. در صورتیکه در همان مرتبه اول بررسی شرط مقدار ناصحیح داشته باشد دستورات به هیچ وجه اجرا نمی گردند. به عبارت دیگر درصورتیکه شرط هرگز مقدار غلط پیدا نکند حلقه به طور بی نهایت تکرار می شود. برای کنترل تعداد دفعاتی که یک حلقه کدهایش را اجرا می کند شما حداقل یک متغییر در جمله شرط دارید که در بلوک کد متعاقب تغییر می یابد. این موضوع به جمله شرط امکان می دهد در برخی مراحل نادرست گردد. نمونه‌ی را که پیش رو دارید یک بازی ساده حدس زدن عدد است:

نمونه 4.5 guess_my_number.c

```
#include
int
main()
{
  const int MAGIC_NUMBER = 6;
  int guessed_number;

  printf("Try to guess what number I'm thinking of\n");
  printf("HINT: It's a number between 1 and 10\n");

  printf("enter your guess" :);
```

```
  scanf("%d", &guessed_number);

  while (guessed_number != MAGIC_NUMBER);
{
     printf("enter your guess" :);
     scanf("%d", &guessed_number);
   }

  printf("you win.\n("

  return 0;
}
```

بلوک کد حلقه while در نمونه بالا تا زمانی که بازیکن عدد 6 را حدس بزند مرتب تکرار می شود.

4.5 حلقه for

معمولا در مواردیکه تعداد دفعات تکرار حلقه از قبل معیین است از حلقه for استفاده می کنیم. در ساختار حلقه for متغییری وجود دارد که تعداد دفعات تکرار حلقه را تعیین می کند و اصطلاحا شمارنده نامیده می شود. در پرانتز بعد از کلمه for سه عبارت وجود دارند که توسط سمی کالون (;) از یکدیگر جدا شده اند. اولین عبارت مقدار اولیه متغییر شمارنده را مشخص می کند. آخرین عبارت معیین می کند که با هر بار اجرای دستورات حلقه مقدار متغییر شمارنده به چه میزان اضافه یا کم می شود و بالاخره عبارت وسط که شرط حلقه است مشخص می کند حلقه تا چه زمانی باید اجرا شود.

فرم کلی حلقه if به صورت زیر است:

```
for (initialization; expression; action )
{
   statement1;
   statement2;
   statement3;
}
```

در نمونه زیر متغییر i متغییر شمارنده حلقه for می باشد:

نمونه for_ten.c 4.6

```
#include

int
main()
{
  int i;
 */  display the numbers from 0 to 9*/
  for (i = 0; i < 10; i++)
    printf("%d\n", i);

  return 0;
}
```

4.6 حلقه do... while

حلقه do ... while شبیه حلقه while عمل می کند با این تفاوت که شرط حلقه به جای ابتدا در انتهای حلقه بررسی می شود. پس حتی اگر شرط حلقه مقدار نادرست داشته باشد دستورات حلقه حداقل یک بار اجرا می شوند. در ذیل نمونه‌ی از کاربرد این حلقه آمده است:

4.7 نمونه guess_my_number.c

```
#include

int
main()
{
  const int MAGIC_NUMBER = 6;
  int guessed_number;

  printf("Try to guess what number I'm thinking of\n;("
  printf("HINT: It's a number between 1 and 10\n;("

  do
{
    printf("enter your guess;(" :
    scanf("%d", &guessed_number;(
}
```

```
  while (guessed_number != MAGIC_NUMBER;(


  printf("you win.\n("


  return 0;
}
```

4.7 عملگر شرطی

عملگر ?: شبیه دستور if ... else عمل می کند با این تفاوت که چون یک عملگر است (و نه یک دستور)در میان عبارت‌ها نیز کار برد دارد.

نمونه 4.8 apples.c

```
#include

int
main()
{
  apples = 6;


  printf("I have %d apple%s ", apples, (apples == 1) ? "" : "s");


  return 0;
}
```

?: تنها عملگر سه گانه در زبان c است.در نمونه بالا عبارت (apples == 1) شرط ماست که درستی آن بررسی می شود. در صورتی که شرط صحیح باشد (که در نمونه بالا نیست) در خروجی کلمه apple بدون s و در صورت اشتباه بودن شرط کلمه apple همراه با s چاپ خواهد شد. 4.9 break و continue

زبان c راه بسیار ساده‌ای برای خارج شدن از حلقه‌های تکرار در مواقع لزوم فراهم می‌کند. برای این کار باید از دستور break که پیش از این به منظور بیرون جهیدن از ساختار switch از آن بهره برده بودیم استفاده کرد. نمونه زیر اعداد یک تا 12 را چاپ می کند:

نمونه 4.9 break; } printf ("%d " ,i); } return 0 ;} { if (i == 12) { break; for (i = 1; i <= 20; i++) { #include int main() break.c

در یک حلقه تکرار باعث می continue به کار بردن دستور { ;} شود تا دستورات بعدی نادیده گرفته شده و مفسر به ابتدای حلقه باز گردد. نمونه زیر حاصل تقسیم عدد بیست را بر اعداد محدوده منفی ده تا مثبت ده چاپ می کند اما برای جلوگیری از تقسیم شدن عدد بیست بر عدد صفر از یک شرط و دستور continue استفاده شده است.

نمونه 4.10 continue.c

```
#include
int
main()
{
 for (i = -10; i <= 10; i++)
  {
```

```
  if (i == 0)
{
    continue;
    }
  printf ("%d", 20/i);
}
return 0;
}
```

بخش پنجم : اشاره گرها

5.1 اصول اولیه

محدودیتی که ممکن است شما نیز متوجه آن شده باشید این است که توابع تنها از طریق مقادیر بازگشتی خود در برنامه شما تاثیر گذراند. با این وصف اگر بخواهید تا یک تابع با بیش از یک متغییر سر و کار داشته باشد چه خواهید کرد؟ پاسخ استفاده از اشاره گرهاست.

یک اشاره گر نوع خاصی از متغییرهاست که برای نگهداری آدرس‌های حافظه کاربرد دارد یعنی یک اشاره گر آدرس متغییر دیگری را به عنوان مقدار خود نگهداری می‌کند. اعلان اشاره گرها به مانند اعلان متغییرهای عادی است با این تفاوت که شما باید یک کاراکتر ستاره "*" را به ابتدای نام اشاره گر اضافه کنید. دو عملگر جدید برای کار با اشاره گرها وجود دارد که شما باید آنها را بشناسید:عملگر & و عملگر * که هر دو جزء عملگرهای یگانی پیشوندی محسوب می شوند . زمانی که شما یک علامت آمپرسند "&" را در آغاز نام یک متغییر اضافه می کنید شما آدرس آن متغییر را دریافت خواهید کرد که می تواند در یک اشاره گر نگهداری شود. اما زمانیکه علامت ستاره را در آغاز نام یک اشاره گر به کار می برید شما مقدار نگهداری شده در آدرس حافظه‌ای را که اشاره گر به آن اشاره می کند دریافت خواهید کرد. مثل همیشه بحث را با یک نمونه دنبال می کنیم :

نمونه 5.1 pointers_are_simple.c

```
#include

int
main()
}
  int my_variable = 6, other_variable = 10;
  int *my_pointer;

  printf("the address of my_variable is    : %p\n", &my_variable);
  printf("the address of other_variable is : %p\n", &other_variable);

  my_pointer = &my_variable;

  printf("\nafter \"my_pointer = &my_variable\":\n");
  printf("\tthe value of my_pointer is %p\n", my_pointer);
  printf("\tthe value at that address is %d\n", *my_pointer);

  my_pointer = &other_variable;

  printf("\nafter \"my_pointer = &other_variable\":\n");
```

```
  printf("\tthe value of my_pointer is %p\n", my_pointer);
  printf("\tthe value at that address is %d\n", *my_pointer);

  return 0;
}
```

خروجی برنامه آدرس حافظه دو متغییر را به شما نشان می دهد. مقادیر در رایانه من با رایانه شما متفاوت خواهند بود. در تابع printf() همانطور که شما هم متوجه شده‌اید از p٪ برای نمایش آدرس‌ها بهره بردیم که شاخص تبدیل برای تمام اشاره گرهاست. به هر حال خروجی برنامه در رایانه من به صورت زیر بود :

خروجی برنامه

```
the address of my_variable is    : 0xbffffa18
the address of other_variable is : 0xbffffa14

after "my_pointer = &my_variable:"

        the value of my_pointer is 0xbffffa18
        the value at that address is 6

after "my_pointer = &other_variable:"

        the value of my_pointer is 0xbffffa14
        the value at that address is 10
```

5.2 آدرس یک متغییر

برنامه‌ها در زمان اجرا ، وقتی به اعلان یک متغییر می رسند از سیستم عامل درخواست تخصیص مقداری حافظه برای آن متغییر را می نمایند. سیتم عامل قطعه‌ای از حافظه را که اندازه آن برای نگهداری مقادیر متغییر مناسب است انتخاب و آدرس آن را به برنامه اعلام می‌کند. هر زمان که برنامه قصد خواندن داده‌هایی را که درون آن متغییر نگهداری می شوند داشته باشد ، به آدرس حافظه آن متغییر مراجعه کرده و به برابر با اندازه آن متغییر بایت‌های حافظه را می خواند. در صورتی که شما یک بار دیگر برنامه‌ای را که در اغاز این فصل ذکر کردیم اجرا کنید ممکن است نتایج مشابه یا متفاوتی را نسبت به بار اول در مورد آدرس‌های حافظه بدست آورید که‌این امر بستگی به شرایط سیستم شما دارد اما حتی اگر در بار دوم نتایج یکسانی را نسبت به یار اول بدست آورید هیچ تضمینی برای اینکه اجرای برنامه در فردا نیز همین نتایج را داشته باشد وجود نخواهد داشت. در واقع این احتمال که فردا نیز برنامه همین نتایج را داشته باشد تقریبا محال است.

5.3 اشاره گرها در جایگاه آرگومان (نشانوند)های تابع

یکی از بهترین مزیت‌های اشاره گرها این است که آن‌ها به توابع این امکان را می دهند تا متغییرهایی در خارج از حوزه خود را تغییر دهند. با ارسال یک اشاره گر برای یک تابع ، شما به آن تابع امکان خواندن و نوشتن بر روی داده‌های نگهداری شده در متغییر مربوط به آن اشاره گر را می دهید.

مثلا شما قصد نوشتن تابعی را دارید که مقادیر دو متغییر را جا به جا می‌کند. بدون بهره گیری از اشاره گرها نوشتن این برنامه غیر ممکن است. در زیر کدهای برنامه مورد نظر را می بینید:

Example 5-2. swap_ints.c

```
#include

int swap_ints(int *first_number, int *second_number);

int
main()
```

```
{
  int a = 4, b = 7;

  printf("pre-swap values are: a == %d, b == %d\n", a, b)

  swap_ints(&a, &b);

  printf("post-swap values are: a == %d, b == %d\n", a, b)

  return 0;
}

int
swap_ints(int *first_number, int *second_number)
{
  int temp;

 ❊/  temp = "what is pointed to by" first_number; etc/❊ ...

  temp = *first_number;
❊  first_number = *second_number;
❊  second_number = temp;

  return 0;
```

```
}
```

همانطور که مشاهده می کنید اعلان تابع swap_ints () به کامپایلر gcc می گوید که انتظار دو اشاره گر (آدرس متغییر) را داشته باشد. همچنین عملگر & به منظور ارسال آدرس دو متغییر به جای مقدار آن‌ها به کار رفته است.

موفق باشید.