What is an external Hearts player?

An external Hearts player is a separate program, with its own window, that acts as a player in the four player Hearts game. This player has access to everything that the 'internal' players of the game do. As well as being told everything that happens in the game and who played what, this includes the fact that the external player can put and receive comments on the screen just like the other players. The external player has his own window, so the programmer can draw whatever he wants there. So if the external player gets the Queen of Spades just before he thought he would win, a picture of a disgruntled player can appear in the window.

I (Paul) am very excited about this idea of external players, because it allows you to make your own player that could try to roast my own CPU players. All interactive games (and other programs) should have a protocol such as this, yet they rarely do. Aldus Photostyler and Superpaint's drop-in painting modules are like this, but they use DLL's. The After Dark screen saver program not only provides the capability to add your own modules, it gives you the format and example code on every copy of the program. While After Dark uses DLL's, the Windows 3.1+ internal screen saver actually uses separate executable programs, just like Hearts. I am doing this partly to help encourage other software developers to incorporate such features when possible. *Please note that while this present format is correct and final, there may need to be changes to the version of Hearts 2.X that you have. Contact me if you are having any problems getting the external player to work. I WILL have a completely working copy of Hearts to give your free if necessary.*

How Hearts implements an external player

The external Hearts player communicates with the Hearts 'dealer' through DDE. The method of DDE implemented is simple and you only must know the most basic aspects of DDE as explained in the Windows Programmer's Reference or many other introductional books. You may have heard of more modern versions of DDE such as DDEML, or more likely, OLE; the DDE used in Hearts is the very lowest common denominator of these.

You need no special libraries, DLL's or strange source code. If you can create a window and have access to the standard Windows 3.X API, then you can do an external player. This means that most development systems in existence today can be used to make a player. This includes any C compiler, any Pascal Compiler, most BASIC systems, Smalltalk, and Fortran. Whether or not you can create a player with TurboVision, Paradox, or even EXCEL is unknown by me!

For a single paragraph refresher, I will describe here what DDE does and how programs use it. DDE messages are messages that one program can send to another, as opposed to regular messages, which are used for sending from a program or Windows to itself. Along with the DDE message we send to other programs, we can send a handle to some data in the 'long' parameter of the Windows message. This means that we can share data with other programs. It also is the reason we need DDE to send inter-program messages: If you tried to send a handle to your data in a regular Windows message, the recieving program would cause a protection fault if it tried to read the data. The data shared between two programs is in the form of either 1) globally allocated memory with the 'DDE_SHARE' option or 2) globally allocated strings allocated with the 'GlobalAddAtom()' family of functions. The strings are used to identifying the topics of the DDE message, while the global DDE_SHARE data generally holds the specific information that the strings are topics for.

Hearts uses DDE with an external player by allocating a single large DDE_SHARE memory block that will be used for the entire game, and storing information there for the external player to use when he gets the DDE message. For example, the dealer must tell the external player that it is his turn to make a play when that time comes. So what the dealer does is store the trick's cards that the previous players have already played (if any) in the DDE_SHARE memory for the external player to see, so he can decide how to play his own card. In turn, when the external player has decided on what card to play, he stores it in a specific place in the shared memory and sends a DDE message back to the dealer. The dealer all along has been patiently waiting for this DDE message, because she knows that it is the external player's turn.

There are certain conventions that must be followed in order to establish a working communication between the dealer and the external player. The most basic of these is that the Hearts program and the external player(s) must have a way of telling each other that they are present and ready to play the game. In addition, there must be a way to terminate the 'conversation' between the two. The Hearts dealer tells the player what is happening and asks it things that it needs to know. Some messages the dealer sends are merely notifications to the player of things that have occurred, such as: you were just passed these three cards (cards stored in the share), or the rules have just been altered, or player 3 just said to you, "your momma!" Other messages from the dealer require a response, such as, 'It is your turn, what will you play', or 'what is your name, rank, and serial number?' The player must respond to these questions within a set time or default answers will be made for him.

This document is about how to follow these conventions, with examples given in C code. It is assumed that you have some knowledge of how to use DDE. If you aren't familiar with DDE, it's really pretty simple, though not well designed. All you need to know is how to manipulate 'atoms' (takes about 30 mins. to figure out) and understand what the 9 basic DDE messages are (about 1 hour to figure out). You can get an already-made and working external player that has everything written for you (including all DDE and atom manipulation code) except all you have to do is fill in the blanks with your own logic! This code will be available from the author (Paul Pedriana) in a number of languages, hopefully at least C, Pascal, and Visual Basic. Please see the registration information on how to get this source code in addition to or separate from registration. You see, I am writing this document at least 6 weeks before Hearts 2.0 is done, so I haven't decided how to do registrations yet. As of today 1/1/93, I have already written the C code, and the Visual Basic ought to be done by March at the latest.

The Hearts Game

A Hearts game consists of a dealer and four players. The dealer is in control of the game. He brings together the four players, puts the cards on the screen, deals the cards to each player, tells each player when it is his turn to play, etc. The dealer alos enforces the rules of the game; if a player tries to play a card that wasn't dealt to him, then the dealer will 'punish' the player by taking a card from the player's hand that is legal and play it instead. This card is likely to be a poor play.

The dealer is always in a given 'state', each state describes what the dealer will try to do next, whether it be deal a hand of cards or declaring the winner of the game. These states are clearly enumerated. At the beggining of the game, the dealer is in the 'DS_INIT' state. Then the dealer goes into the DS_NEW_GAME state, etc. The diagram shown below shows the order of the dealer's states as the session progresses. Some actions the dealer takes are independent of his state. For example, when the user changes the style of card design and the dealer happens to be in , lets say, the DS_PLAYER_2_PLAY state, the dealer simply changes the card design; that operation has nothing to do with his state, and so is independent of it. The diagram below actually isn't important for the implementation of an external player, but it may help you understand the way the game is played internally.

On the other hand, it is true that for just about every given state, the dealer has a message to pass to the players. While the dealer is in the DS_DEALING_ROUND state, he sends a message (the 'P_TAKE_CARD' message) to each player 13 times to 'take' each card the dealer is dealing to him. The player DOES need to understand the 'P_XXX' messages the dealer sends to him. There are 34 P_XXX messages and they are listed below.

µ §

Below lists the messages a dealer can send a player and the values that the messages have.

```
#define P_ABORT_GAME            101     //The entire game is aborted.  Next message may be NEW_GAME
#define P_ABORT_HAND            102     //Abort only the present hand.  Game continues.
#define P_CARD_SELECTION        103     //The user touched one of your cards with the mouse.
#define P_DO_PASS               104*    //It is your turn to do a pass.
#define P_DONE_HAND             105     //The hand is done, all 13 tricks have been played.
#define P_END_GAME              106     //Some player blew it and reached 100 pts.  This game is over.
#define P_HEARTS_BROKEN         107     //Simple, hearts were broken on the hand just played.
#define P_LEAD_CARD             108*    //It is your turn to lead a card.
#define P_LOSE_CARD             110     //You just played the given card. Now remove it from your hand.
#define P_LOSE_PASS             111     //You just passed the given cards, remove them from your hand.
#define P_NEW_GAME              113     //A new game is about to start.
#define P_NEW_HAND              114     //A new hand is about to start. You'll start getting cards.
#define P_NEW_RULES             116     //The rules have been changed by the user.
#define P_PASS_ERROR            117     //You just tried to pass cards you didn't have.
#define P_PLAY_CARD             118*    //It is your turn to play
#define P_PLAY_ERROR            119     //You just tried to play a card illegally
#define P_PLAYER_COMMENT        121     //A player said something
#define P_PLAYER_SHOT_MOON      122     //Easy- someone made a 'run'
#define P_REGISTER_OPPONENTS    125     //Dealer tells you about your opponents (names, etc.)
#define P_TAKE_CARD             127     //This means you are dealt a card from the dealer
#define P_REGISTER_PLAYER       126*    //Dealer wants to know your personal information
#define P_TAKE_PASS             128     //Dealer 'hands' you the three cards passed to you
#define P_TAKE_TRICK            129     //You have won the most recent trick
#define P_TRICK_DONE            130     //The most recent trick is done.
#define P_WAITING               132     //Dealer's waiting. Info is seconds in present state.
#define P_CARD_PLAYED           133     //A card was played by player.
#define P_SHUT_UP               134     //The player is talking too much (i.e. > 15 times a hand)
#define P_PLAYER_CHANGE         135     //The user decided to change the players in the middle of the game.
```

'*' Marks messages that require a response from the player before continuing the game. Other messagesare not responded to, but the player should pay attention to them.

Some of the numbers are out of sequence; this is because some unnecessary messages have been removed since the previous version of Hearts.

Remember that the dealer sends a *message* to a player and a packet of *information* that gives more details about the message; that packet is the DDE_SHARE Windows globally allocated memory (i.e. allocated by a 'C' call of 'GlobalAlloc(GMEM_DDESHARE | GMEM_MOVEABLE, 1000)'). For example, if the dealer sends a P_TAKE_PASS message to a player, then the three cards passed are in the shared memory, so the player can see what they are. See below for a description of the exact format for each fo the messages above.

Note that messages can be categorically seperated into to types: *queries* and *notifications*. The queries are messages from the dealer that require an answer from the player before play can proceed. The notifications are messages from the dealer that simply notify the player that something has happened. Some notifications can be ignored, but some are important. For example, the P_WAITING message can usually be ignored, but the P_TAKE_CARD message is important because it is the dealer telling you he just dealt you the given card (if you don't want your player to be any good, then you can ignore this notification and thus not know what cards are in your hand. Also note that the query and notification messages and their information packet are seperate in the global share. A notify message is in the 'nNotifyMessage' place, and the query is in the 'nQueryMessage' place. These two types of messages are seperated in order to allow them to be asynchrous. Think about a situation in which your external player is taking a long time to think about a play for some reason. If the notify messages and query messages were synchrous and queued, you may not have gotten your P_PLAY_CARD message yet but you already are being sent P_WAITING messages. The dealer would then overwrite the P_PLAY_CARD message before you got it. Do ya get it?

An Example of How the Dealer Sends Messages to the External Player

Note that the actual format of the information sent in the global memory and the format of information that the player sends back to the dealer are given below in the section, 'The Protocol Itself'. The dealer sends out messages as often as 5 times a second or so, though often this will in reality be more like one every second.

In all cases below, the player would do best to answer all queries immediately. Though this is not strictly required, it minimizes the chances of time-outs, which cause the dealer to think you abandoned ship and thus make a (lame) play for you. Also the player must remember for the query messages to copy the 'nQueryMessage' value back into 'nReturnMessage' so the dealer knows what message the player is responding to (important for if the system slows down and time's out a lot).

What the dealer sends to the player	What the player should do
P_REGISTER_PLAYER*	You are being asked by the dealer for information about yourself. You need to fill the shared memory with at least a name for the dealer to refer to you with (a 'playerinfo' structure). Note that you are one of an array of four players numbering 1-4. The dealer will pass you a 'playerinfo' struct that is empty except that it will tell you what index you are of the 4 players. On the playing screen, the first player is the left one and counts clockwise.
P_REGISTER_OPPONENTS	The share holds the information regarding the three opponents and yourself that each of you filled in at getting the P_REGISTER_PLAYER message. The share simply holds an array of four 'playerinfos'. You can ignore this if you want.
P_NEW_GAME	He needn't do anything
P_NEW_HAND	He needn't do anything
P_TAKE_CARD (sent 13 times in a row)	The card is in the global memory. The player should maintain an array of 13 cards, and add this card to it.
P_DO_PASS*	Pick 3 cards to pass and put them in the appropriate place in the global shared memory.
P_LOSE_PASS	The three cards that you decided to pass in the P_DO_PASS message will be given in the shared memory, unless you tried to pass something you didn't have, in which case the three cards here will be different! Don't subtract cards from your hand when responding to the P_DO_PASS message, subtract them when receiving this P_LOSE_PASS message.
P_PASS_ERROR (only if you tried to cheat)	You would receive this immediately after you tried to pass cards you didn't have. After this, you would receive a P_LOSE_PASS message with a some other cards in it that you DO have!
P_LEAD_CARD*	You are the leader of this trick, either because it is the first round and you have the 2 of Clubs, or you have won the previous trick. You can lead any legal card; you put the card you want to lead into the appropriate place in the shared memory for the dealer to pick up when he gets your answer.
P_PLAY_CARD*	It is your turn to play in a trick that someone else lead. The shared memory contains a 'trickstruct' that describes the lead suit and has an array of four cards. The cards that have been played will be in their places in the array and your place and the player(s) after you, if any, will be empty.
P_LOSE_CARD	This is like P_LOS_PASS. The card you lose is in the share. You will get this immediately after you play a card. Assuming you played a legal card, this will be the card you put into the shared memory after receiving the P_PLAY_CARD message.
P_CARD_PLAYED	You can ignore this. The dealer sends this to everyone right after a player ahs played a card. The share simply hold a copy of the the state of the present trick as a 'trickstruct'.
P_TAKE_TRICK	You will recieve this message if you won the just-passed trick. The share holds a copy of the completed trick in a 'trickstruct'. You don't need to do anything.

P_TRICK_DONE	Everyone receives this message after a trick is done. You don't need to do anything.
P_HAND_DONE	Everyone recieves this when all 13 tricks have been played. the share holds a 'donehandstruct' which describes some stats for the hand. Nothing here is important.
P_END_GAME	Someone reached the maximum allowable points and this game is over. This message only comes after a P_HAND_DONE message. The shared memory holds a 'endgamestruct', which is nothing more that an array of 4 'scorestructs'- one for each player. You needn't do anything, but you can return a comment like, 'I was robbed!'
P_PLAYER_COMMENT	One of the players made an on-screen comment. The shared memory holds the actual comment the player said and who he was in the form of a 'playercommentstruct'
other messages	other messages such as P_WAITING, P_PLAYER_SHOT_MOON, or P_SHUT_UP, should be somewhat self-evident if you understand the other messages given above. You needn't respond to any of these other messages, but you still will get them, of course.

The Protocol Itself

Note that all DDE messages between the player and Hearts are communicated with the Windows 'PostMessage' function, except the WM_DDE_INITIATE and WM_DDE_ACK communications, which are sent with 'SendMessage'. This is not due to my design, but Microsoft's.

Starting a game session.

When Hearts is started, it sends out WM_DDE_INITIATE message to all 'overlapped' windows, including your external player window. You must respond by sending (not 'posting') a WM_DDE_ACK message back to the dealer hwnd (wParam of the INITIATE message). You put the atom to "ATNHG" in the hiword of lParam and an atom to your own name in loword of lParam. Now your player will be on the list of player's that the user can select as opponents. If the user selects you in the player selection box, you will then be sent a WM_DDE_INITIATE message (via SendMessage()) with the hiword of lParam being "ATYC" (ATOM_TOPIC_YOUR_CONTACT). This means that a game is started and you are one of the players; you return 1 if this is OK and 0 if not OK (not OK could mean, for example, that you will be playing a game with someone else). The loword of lParam is the handle of the shared memory that the two of you will be using throughout the game. wParam is the handle of the window you will always be sending DDE messages to throughout the game. It will be different from the wParam that you got with the INITIATE message. You may want to save this new wParam, because it won't change and you can use it to check against any future WM_DDE_ messages to make sure that they are from this same window.

If by some chance, you are started after the Hearts program is started, you will want to send a WM_DDE_INITIATE message to all top-level windows (e.g. with a 'C' call of SendMessage(-1,,,)). Put your window handle in wParam, your name in loword of lParam, and "ATNHP" (ATOM_TOPIC_NEW_HEARTS_PLAYER) in hiword of lParam. This will cause Hearts to include you in its list of potential players. Since you can't know very easily whether Hearts is running when you start, you may want to always send this message when you start up.

It is important that you always return WM_DDE_ACK to the dealer's WM_DDE_INITIATE messages, because the dealer may: 1) not ever use you, but another copy of Hearts may be started later that sends out its own WM_DDE_INITIATE message. 2) the user may decide to switch players in the middle of the game and so you could receive a new WM_DDE_INITIATE from the dealer in the middle of a game the dealer is already playing with you. If this latter case happens, and the user removes you from your position at the card table but decides to put you in a different place at the table, you will receive a WM_DDE_TERMINATE (see below about termination) and then a WM_DDE_DATA with a hiword of lParam being "ATYC" (ATOM_TOPIC_YOUR_CONTACT).

These items I have mentioned here are the only things you need to know about initiating a game session. While technically, Hearts is the DDE 'server' and the player is the DDE 'client', these labels don't really have much meaning here since both are communicating on a nearly equal level.

Ending a session.

A session with a player can end in a number of ways. The most basic is that the user quits the Hearts game. In addition, the user can decide to switch opponents in the middle of a session. The replaced players will have to be sent termination messages, so they can clear their slates and be ready for a new session. When a player's present session is to be terminated, he receives a WM_DDE_TERMINATE message from the dealer. He must then prepare himself in whatever necessary way for receiving a new MN_DDE_INITIATE message.

During a session.

The shared memory is a structure that is always the same length and format; it is shown below. The 'notifyInformation', 'queryInformation' and 'returnInformation' fields hold structures whose type depends on the message being sent. Upon receiving a message from the dealer (either a message with a

"ATYT" (ATOM_TOPIC_YOUR_TURN topic or a "ATN" ATOM_TOPIC_NOTIFY topic), you can read the message by calling the Windows GlobalLock() function to get a long pointer to the memory (be sure to call GlobalUnlock() when done). For every 'nMessage' that there is, there is a different type of data stored in 'notifyInformation' of 'queryInformation'. You will have to 'cast' the information pointer to a structure of the type determined by the message. Below is a list of every message sent by the dealer and what structure to expect in 'information'. Note that when a player is returning an answer to a message from the dealer (specified in the nMessage part of the shared memory, the player must copy that nMessage value to the 'nReturnMessage' part of the shareed mdmory. This is done because if the player is very slow (or there is some problem) and is responding to messages that came a while ago (i.e. so long ago that there was a time out and the dealer played for the player), the dealer will be able to understand what is going on.

```
#define L_DDESHARE_MEMORY          1000   //Size in bytes of the memory to be used.
#define N_INFORMATION              200
#define N_RETURN_INFORMATION       200
struct MessageStruct{
  short   nNotifyMessage;                 //P_XXX
  char    notifyInformation[N_INFORMATION];     //holds notify info, if there is any
  short   nQueryMessage;                  //P_XXX (XXX is REGISTER, PASS, PLAY, LEAD)
  char    queryInformation[N_INFORMATION];      //message info for P_XXX message stored here.
  short   nDealerState;                   //DS_XXX
  short   nReturnMessage;                 //Player MUST set this to message he's returning
  char    returnInformation[N_RETURN_INFORMATION];  //a structure, sent to dealer
  char    szComment[MAX_COMMENT];               //you can put a comment here
  cardtype myCards[13];                   //an array to your cards
  short   nLegalCards[13];                //TRUE for each legally playable card.
};
```

The player must remember for the query messages (ATOM_TOPIC_YOUR_TURN) to copy the 'nQueryMessage' value back into 'nReturnMessage' so the dealer knows what message the player is responding to (important for if the system slows down and time's out a lot).

Sent with a ATOM_TOPIC_YOUR_TURN atom:

'nMessage'	the structure in 'notifyInformation' or 'queryInformation'	what you return in 'returnInformation'
P_DO_PASS*(star means query)	'passstruct'. Also, myCards[] will hold all your cards.	You want to put a copy of the 'passstruct' in returninfo
P_LEAD_CARD*	'trickstruct' Also, 'myCards' will hold an array of your cards presently, and myLegalCards will hold a corresponding array of TRUE's and FALSE's depending on whether the card is a legal play or not. Thus you don't have to know the rules to know what cards are legal.	Simply put your card to play (a 'cardtype') here.
P_PLAY_CARD*	"	" (just like P_LEAD_CARD)
P_REGISTER_PLAYER*	'playerinfo'. This structure is to be ignored except the 'number' parameter if you are interested in knowing which position at the table you hold.	You put a 'playerinfo' in the 'returnInformation' place. Fill in all the information except 'number'. Name is the only required answer, actually.

Sent with a ATOM_TOPIC_NOTIFY atom:

'nMessage'	the structure in 'information'	what you do
P_LOSE_CARD	'cardtype' The card you lost in the most recent play. Unless you tried an illegal card, this will be the same as the 'cardtype' you put in the 'returnInformation' in response to a P_PLAY_CARD or P_LEAD_CARD message.	You don't want to reply, but since you will probably want to maintain a hand of cards, this is where you want to subtract the card from your hand.
P_TAKE_CARD	'cardtype' The card just dealt to you.	If you are maintaining a hand, you will want to add the card to your hand here.
P_LOSE_PASS	'passstruct'. The three cards in the passstruct are the ones you should lose.	You will want to subtract the three cards from your hand here.
P_TAKE_PASS	'passstruct' The three cards are the ones passed to you	You will want to add the three cards to your hand

	from the player in the 'player' field.	here.
P_NEW_RULES	'rulestruct'	You needn't respond, but you may want to complain in some way to the user!
P_ABORT_GAME	nothing	nothing
P_ABORT_HAND	"	"
P_HEARTS_BROKEN	"	"
P_NEW_HAND	"	"
P_PLAY_ERROR	"	"
P_PASS_ERROR	"	"
P_WAITING	"	"
P_PLAYER_CHANGE	"	"
P_NEW_GAME	"	"
P_PLAYER_COMMENT	'playercommentstruct'	"
P_DONE_HAND	'donehandstruct'	"
P_END_GAME	" (donehandstruct)	"
P_REGISTER_OPPONENTS	an array of four 'playerinfo's'	"
P_CARD_SELECTION	'winmsgstruct'. This describes a card of yours on the screen that the user touched with the mouse.	"
P_PLAYER_SHOT_MOON	index of player who shot, it could be your index or opponents.	" (though you may want to complain by putting a nasty comment in the 'comment' field of the shared memory).
P_TAKE_TRICK	'trickstruct' (completed)	"
P_TRICK_DONE	'trickstruct' (same as the above P_TAKE_TRICK except all the players receive this message.)	"

The DDE Messages Sent in a Hearts Session.

wMessage (WM_DDE_)	wParam	loword of lParam	hiword of lParam
INITIATE (dlr->plr)	Handle to sending window	atom: "Hearts"	atom: "ATNHG"
INITIATE (plr->dlr)	Handle to sending window	atom: "Joe Dufus" (e.g.)	atom: "ATNHP"

ACK (plr->dlr)	Handle to plr's window	atom: "Joe Dufus" (e.g.)	atom: "ATNHG"
DATA (dlr->plr)	Handle to dlr's window	HANDLE: the global share	atom: "ATYC"
DATA (dlr->plr)	Handle to dlr's window	HANDLE: the global share	atom: "ATYT" This means that the message is one the dealer needs to get an answer to.
DATA (dlr->plr)	Handle to dlr's window	HANDLE: the global share	atom: "ATN" This means that the message is one that doesn't need a response from the player.
DATA (dlr->plr)	Handle to dlr's window	HANDLE: the global share	atom: "ATIT" This simply means that the dealer is experiencing unused time. The player can (e.g.) respond with a comment about the game slowing up...
POKE (plr->dlr)	Handle to plr's window	HANDLE: the global share (actually, this is the same handle as in the DATA messages)	atom: "ATPR" Means the player has placed his answer to the dealer's "ATYT" message in the global share.
POKE (plr->dlr)	Handle to plr's window	<unused>	atom: "ATPNR" Player can send this back to dealer when he is not responding to a message but wants to make a comment. He would then place a comment in the appropriate place in teh shared memory.

```
//*********************************************************************************************
//*********************************************************************************************
//*********************************************************************************************
//*********************************************************************************************
//The following is a complete definition of all the structures used by an external Hearts player.

//!!Important note on player indexing: Hearts uses counting that starts with 0 (zero) in almost all
//  cases.  Thus when a structure holds, for example, the index of the player who just shot the moon,
//  that index is zero-based, and the player on the left of the screen is zero, and counts
//  clockwise.  There is only one exception to this rule.  It is the index given in the 'playerinfo'
//  'number' field.  Since it generally doesn't matter too much to a player which index he, no she!,
//  is unless she is doing something like 'targeting' specific players, this anomaly won't matter
//  too much to anyone.
//*********************************************************************************************
#define TRUE        1
#define FALSE       0
#define YES         TRUE
#define NO          FALSE
#define OK          TRUE
#define NOT_OK      FALSE

//*********************************************************************************
//    These are the generic structure definitions
//*********************************************************************************

typedef struct{          //These cardstruct and cardtype structs allow you to address a card by suit
  char number;  //2-14    //  or by number (10, or 4) or by exact value (3 of Hearts).  You store a
  char suit;    //1-4     //  card as a two byte value (int, here) and you cast a cardtype to it.
}cardstruct;             //  Then you can say 'if((cardtype) my_int.specific ==_Q_SPADES){}'
                    //  Or at the same time 'if((cardtype)my_int.kind.number > 10){dump_it()}'
```

```
typedef union{            //
  short     specific;   //
  cardstruct kind;      //
}cardtype;              //



typedef struct{         //One of these is kept for each of the four players at run time.
  short points;        //How many points a player has this game.
  short queens;        //How many queens this game
  short jacks;        //This time you guess!
  short games;         //How many games this player won.
}scorestruct;



typedef struct{              //FALSE is 0 (zero) TRUE is anything else.
  short jackrule;           // _J_DIAMONDS worth -10 if you get it.
  short maxscore;            // Score that ends game.
  short shootrule;          // Whether shooting causes -26 or +26(TRUE means -)
  short maxeraserule;        // If you go back to 0 at 100, TRUE or FALSE.
  short queendumprule;       // You must dump _Q_SPADES at first chance if this is TRUE
  short pointsfirstroundrule;  // If it's illegal to drop points on the first round.
  short rotatingleadrule;     // If true, then players rotate lead instead of 2C leading.
}rulestruct;


#define MAX_COMMENT    128   //The longest number of characters you can put in a comment
#define MAX_NAME       30    //The longest a player's name can be.
#define N_PLAYERS      4     //
#define N_PASS         3     //

#define CPU            1     //Hearts internal players
#define HUMAN          2     //Users playing a hand with the mouse
#define EXTERNAL_CPU   3     //External players- they are seperate applications
#define EXTERNAL_HUMAN 4     //Same as EXTERNAL_CPU, but cards controlled by human
#define EXTERNAL_NETWORK 5   //Some Network DDE, or some other type of player

#define FACEUP       TRUE
#define FACEDOWN     FALSE

#define G_RATING       1     //This area here was originally to be used to filter out
#define PG_13_RATING   2     //  language use or other similar things.  It is not really
#define R_RATING       3     //  being used by Hearts right now.
#define NC_17_RATING   4     //Player puts one of these in the 'rating' part of playerinfo.


//The players need to fill all of the 'playerinfo' items below except the 'number' field when
//  getting a P_REGISTER_PLAYER message.
typedef struct{
  short playerID;           //Special ID Dealer may assign.           (Ignore this)
  short type;              //CPU, HUMAN, ETC
  short cardsfaceup;        //Should Hearts game draw my cards faceup?
  short givevisual;         //Show errors, messages, etc. to him.
  short correcterrors;      //Should dealer correct my errors. (TRUE for all but human)
  short rating;             //G_RATING,PG-13_RATING,R_RATING,NC-17_RATING
  short number;             //1-based player number at the board. (left is 1, bottom is 4).
  short level;             //Skill level,0-1000. Make the number up yourself. For players use.
  char  name[MAX_NAME];     //This is the name players are referred to by.
  char  comment[MAX_COMMENT]; //This is your own personal comment.
}playerinfo;



#define PASS_HOLD   0    // (player+passdirection)%4 is the destination player index.
#define PASS_LEFT   1    //  "
#define PASS_RIGHT  2    //  "
#define PASS_ACROSS 3    //  "
typedef struct{
  short   player;      //The source of the pass (0-based player)
  short   direction;    //This is 'PASS_LEFT','PASS_ACROSS', etc.
```

```
  cardtype pass[N_PASS];  //the cards passed.
}passstruct;



typedef struct{              //When a player gets a P_PLAYER_COMMENT message, this is the
  char comment[MAX_COMMENT]; //structure that is passed, so the player can, if he wants,
  short player;              //analyze who the player was and what he said. 'player'
}commentstruct;              //is 0-based.



typedef struct{              //Very common structure used throughout the game play.
  cardtype cards[N_PLAYERS]; //The cards in the trick or 0 for no card yet.
  short    suit;             //The lead suit
  short    leader;           //The player who led the trick with 'suit'.(0-based)
  short    round;            //1-based count of rounds (1 through 13)
  short    heartsbroken;     //Have they been broken yet this round.
  short    winner;           //decided later by the scorecard (b/c he governs rules)(0-based)
  short    JWon;             //If J_D was won. (TRUE or FALSE)
  short    QWon;             //If Q_S was won.
}trickstruct;


//This structure describes a mouse click that occurred on top of someone's cards.  The internal
//  players respond to one of these messages with a statement like, "Lay off my cards".
typedef struct{
  cardtype selectedcard; // Will be NULL if the card is another player's card.
  short    owner;       // PLAYER_0 - PLAYER_3
  short    place;       // CARD_POS_... HAND,TRICK,PASS,WON,DECK
  short    dealerstate;  //
  short    winmsg;      // msg that caused this call
  short    wParam;      // ditto
  long     lParam;      // ditto
}winmsgstruct;



//These are the numbers used by the dealer when a card playing error has been made by a
//  player.  The player gets one of these values sent to him with a P_PLAY_ERROR message.
#define PLAYER_DOESNT_HAVE_GIVEN_CARD  -1
#define HEARTS_NOT_BROKEN_YET          -2
#define CARD_OF_ILLEGAL_SUIT           -3
#define PLAYER_REQUIRED_TO_DUMP_QUEEN  -4
#define FIRST_RND_NO_DUMP_POINTS       -5
#define FIRST_CARD_MUST_BE_2_CLUBS     -6
#define NO_CARDS_IN_HAND               -20
#define POINTS_NOT_BROKEN_YET          HEARTS_NOT_BROKEN_YET



//Dealer State defines. They aren't very important to the players.
#define DS_INIT            401
#define DS_RESET_SCORECARD  402
#define DS_NEW_HAND         403
#define DS_SHUFFLING        404
#define DS_DEALING_ROUND    405  //The cards are being dealt.
#define DS_PLAYER_1_PASS    406  //Waiting for first player to make his pass
#define DS_PLAYER_2_PASS    407
#define DS_PLAYER_3_PASS    408
#define DS_PLAYER_4_PASS    409
#define DS_PLAYER_1_PLAY    410  //waiting for first player to play a card
#define DS_PLAYER_2_PLAY    411
#define DS_PLAYER_3_PLAY    412
#define DS_PLAYER_4_PLAY    413
#define DS_SUMMING_TRICK    414  //All four cards of trick were just played
#define DS_SUMMING_HAND     415  //All thirteen tricks have been won;new deal
#define DS_SUMMING_GAME     416  //Player just reached 100 pts. Game over.
#define DS_NEW_GAME         417  //Player should reset Everything.
```

```
//These are the messages the dealer can send to a player.  The ones with '*' are one's that
//  require a response from the player before continuing with the game.
#define P_ABORT_GAME                101         //The entire game is aborted.  Next message may be NEW_GAME
#define P_ABORT_HAND                102         //Abort only the present hand.  Game continues.
#define P_CARD_SELECTION            103         //The user touched one of your cards with the mouse.
#define P_DO_PASS                   104*        //It is your turn to do a pass.
#define P_DONE_HAND                 105         //The hand is done, all 13 tricks have been played.
#define P_END_GAME                  106         //Some player blew it and reached 100 pts.  This game is over.
#define P_HEARTS_BROKEN             107         //Simple, hearts were broken on the hand just played.
#define P_LEAD_CARD                 108*        //It is your turn to lead a card.
#define P_LOSE_CARD                 110         //You just played the given card. Now remove it from your hand.
#define P_LOSE_PASS                 111         //You just passed the given cards, remove them from your hand.
#define P_NEW_GAME                  113         //A new game is about to start.
#define P_NEW_HAND                  114         //A new hand is about to start. You'll start getting cards.
#define P_NEW_RULES                 116         //The rules have been changed by the user.
#define P_PASS_ERROR                117         //You just tried to pass cards you didn't have.
#define P_PLAY_CARD                 118*        //It is your turn to play
#define P_PLAY_ERROR                119         //You just tried to play a card illegally
#define P_PLAYER_COMMENT            121         //A player said something
#define P_PLAYER_SHOT_MOON          122         //Easy- someone made a 'run'
#define P_QUERY_NEW_RULES           124*        //Dealer wants to know if you agree with newly proposed rules.
#define P_REGISTER_OPPONENTS        125         //Dealer tells you about your opponents (names, etc.)
#define P_TAKE_CARD                 127         //This means you are dealt a card from the dealer
#define P_REGISTER_PLAYER           126*        //Dealer wants to know your personal information
#define P_TAKE_PASS                 128         //Dealer 'hands' you the three cards passed to you
#define P_TAKE_TRICK                129         //You have won the most recent trick
#define P_TRICK_DONE                130         //The most recent trick is done.
#define P_WAITING                   132         //Dealer's waiting. Info is seconds in present state.
#define P_CARD_PLAYED               133         //A card was played by player.
#define P_SHUT_UP                   134         //The player is talking too much (i.e. > 15 times a hand)

//These are the player message typedefs
typedef struct{                 //This is sent with a P_DONE_HAND message.
  short     QSwinner;           //Index of player winning the Queen this round.
  short     JDwinner;
  short     playershot;         //-1 if no shoot, else is player (0-based).
  scorestruct scores[N_PLAYERS];
  short     gamedone;           //-1 if game not done, else is winner's index.
}donehandstruct;


typedef struct{                 //This is sent with a P_END_GAME message
  scorestruct scores[N_PLAYERS];   //If this doesn't get any bigger then delete.
}endgamestruct;


typedef struct{                 //This is the struct used by dealer to pass to players
  short player;                 //  to tell them what another player said.
  char* comment;                //This is sent with a P_PLAYER_COMMENT message.
}playercommentstruct;




//*****************************************************************************
//********************** Card Defines ***************************************
//Note that a card is a two byte storage, where the first byte is the suit and
//  the second byte is the number.  Thus you can address a card by its suit,
//  its number, or its exact value (e.g. the _3_SPADES is 0x0103, which is suit
//  #1 and card number 3. See the cardtype and cardstruct definitions.
//Note that an ace is called high here, which is appropriate for Hearts. For
//  other games, people may want to call the ace '1' instead of '14'.
#define SPADES        1
#define DIAMONDS      2
#define CLUBS         3
#define HEARTS        4
#define _J           11
#define _Q           12
#define _K           13
#define _A           14
#define _2_SPADES    0x0102  /* decimal 258*/
#define _3_SPADES    0x0103  /* decimal 259*/
```

```
#define _4_SPADES     0x0104  /* decimal 260*/
#define _5_SPADES     0x0105  /* decimal 261*/
#define _6_SPADES     0x0106  /* decimal 262*/
#define _7_SPADES     0x0107  /* decimal 263*/
#define _8_SPADES     0x0108  /* decimal 264*/
#define _9_SPADES     0x0109  /* decimal 265*/
#define _10_SPADES    0x010A  /* decimal 266*/
#define _J_SPADES     0x010B  /* decimal 267*/
#define _Q_SPADES     0x010C  /* decimal 268*/
#define _K_SPADES     0x010D  /* decimal 269*/
#define _A_SPADES     0x010E  /* decimal 270*/

#define _2_DIAMONDS   0x0202  /* decimal 514 */
#define _3_DIAMONDS   0x0203  /* decimal 515 */
#define _4_DIAMONDS   0x0204  /* decimal 516 */
#define _5_DIAMONDS   0x0205  /* decimal 517 */
#define _6_DIAMONDS   0x0206  /* decimal 518 */
#define _7_DIAMONDS   0x0207  /* decimal 519 */
#define _8_DIAMONDS   0x0208  /* decimal 520 */
#define _9_DIAMONDS   0x0209  /* decimal 521 */
#define _10_DIAMONDS  0x020A  /* decimal 522 */
#define _J_DIAMONDS   0x020B  /* decimal 523 */
#define _Q_DIAMONDS   0x020C  /* decimal 524 */
#define _K_DIAMONDS   0x020D  /* decimal 525 */
#define _A_DIAMONDS   0x020E  /* decimal 526 */

#define _2_CLUBS      0x0302  /* decimal 770 */
#define _3_CLUBS      0x0303  /* decimal 771 */
#define _4_CLUBS      0x0304  /* decimal 772 */
#define _5_CLUBS      0x0305  /* decimal 773 */
#define _6_CLUBS      0x0306  /* decimal 774 */
#define _7_CLUBS      0x0307  /* decimal 775 */
#define _8_CLUBS      0x0308  /* decimal 776 */
#define _9_CLUBS      0x0309  /* decimal 777 */
#define _10_CLUBS     0x030A  /* decimal 778 */
#define _J_CLUBS      0x030B  /* decimal 779 */
#define _Q_CLUBS      0x030C  /* decimal 780 */
#define _K_CLUBS      0x030D  /* decimal 781 */
#define _A_CLUBS      0x030E  /* decimal 782 */


#define  _2_HEARTS    0x0402  /* decimal 1026 */
#define  _3_HEARTS    0x0403  /* decimal 1027 */
#define  _4_HEARTS    0x0404  /* decimal 1028 */
#define  _5_HEARTS    0x0405  /* decimal 1029 */
#define  _6_HEARTS    0x0406  /* decimal 1030 */
#define  _7_HEARTS    0x0407  /* decimal 1031 */
#define  _8_HEARTS    0x0408  /* decimal 1033 */
#define  _9_HEARTS    0x0409  /* decimal 1034 */
#define _10_HEARTS    0x040A  /* decimal 1035 */
#define  _J_HEARTS    0x040B  /* decimal 1036 */
#define  _Q_HEARTS    0x040C  /* decimal 1037 */
#define  _K_HEARTS    0x040D  /* decimal 1038 */
#define  _A_HEARTS    0x040E  /* decimal 1039 */



//*************************************************************************************************
//This section describes the information specific to external players, especially the DDE communication
//  structures and atoms.
#define L_DDESHARE_MEMORY            1000   //Size in bytes of the memory to be used.
#define N_INFORMATION                200
#define N_RETURN_INFORMATION         200

struct MessageStruct{
   short    nNotifyMessage;                    //P_XXX
   char     notifyInformation[N_INFORMATION];       //holds notify info, if there is any
   short    nQueryMessage;                     //P_XXX (XXX is REGISTER, PASS, PLAY, LEAD)
   char     queryInformation[N_INFORMATION];        //message info for P_XXX message stored here.
   short    nDealerState;                      //DS_XXX
   short    nReturnMessage;                    //Player MUST set this to message he's returning
   char     returnInformation[N_RETURN_INFORMATION];  //a structure, sent to dealer
```

```
  char    szComment[MAX_COMMENT];              //you can put a comment here
  cardtype myCards[13];                        //an array to your cards
  short   nLegalCards[13];                     //TRUE for each legally playable card.
};

#define MS_FINDER_TIME_OUT_DEBUG    600000L     //Milliseconds    // 1 hour      //If program started
#define MS_PLAY_TIME_OUT_DEBUG      400000L     //Milliseconds    //40 minutes   //with the /s switch
#define MS_FINDER_TIME_OUT          10000L      //Milliseconds    //10 seconds
#define MS_PLAY_TIME_OUT            8000L       //Milliseconds    // 8 seconds
//*********************************************************************************************************
//These are WM_DDE_XXX messages used.
//Dealer sends
#define HM_INITIATE_CONVERSATION         WM_DDE_INITIATE
#define HM_DLR_PROCESS_THIS_MESSAGE      WM_DDE_DATA     //Dealer asks player for move, etc.

//Player sends
#define HM_PLR_HERE_IS_MY_ANSWER         WM_DDE_POKE     //Player sends this to respond.

//Either Sends
#define HM_TERMINATE_CONVERSATION        WM_DDE_TERMINATE
#define HM_ACKNOWLEDGEMENT               WM_DDE_ACK

//*********************************************************************************************************
//Dealer to player.
#define ATOM_TOPIC_NEW_HEARTS_GAME       "ATNHG" //Sent to player at Very beginning
#define ATOM_TOPIC_NEW_HEARTS_PLAYER     "ATNHP" //Sent by player to dealer saying he's available
#define ATOM_TOPIC_YOUR_CONTACT          "ATYC"  //Sent to a new player
#define ATOM_TOPIC_YOUR_TURN             "ATYT"  //topic of data message
#define ATOM_TOPIC_NOTIFY                "ATN"   //Dealer notifies player of something
#define ATOM_TOPIC_IDLE_TIME             "ATIT"  //topic of data message

//Player to dealer
#define ATOM_TOPIC_PLAYER_NOT_READY      "ATPNR"  //topic of poke message
#define ATOM_TOPIC_PLAYER_READY          "ATPR"  //topic of poke nessage
```